WebSphere Event Broker

Message Flows

Version 6 Release 0

WebSphere Event Broker

Message Flows

Version 6 Release 0

Note

Before using this information and the product it supports, read the information in the Notices appendix.

Fifth Edition (December 2006)

This edition applies to IBM® WebSphere® Event Broker Version 6.0 and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

About this topic collection

This PDF has been created from the WebSphere Event Broker Version 6.0.0.3 (with Message Brokers Toolkit Version 6.0.2.0 update, December 2006) information center topics. Always refer to the WebSphere Event Broker online information center to access the most current information. The information center is periodically updated on the document update site and this PDF and others that you can download from that Web site might not contain the most current information.

The topic content included in the PDF does not include the ″Related Links″ sections provided in the online topics. Links within the topic content itself are included, but are active only if they link to another topic in the same PDF collection. Links to topics outside this topic collection are also shown, but these attempt to link to a PDF that is called after the topic identifier (for example, ac12340_.pdf) and therefore fail. Use the online information to navigate freely between topics.

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

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

Part 1. Developing message flows

Developing message flows

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

Concept topics:

- “Message flows overview” on page 4
- “Message flow projects” on page 4
- “Message flow nodes” on page 5
- “Message flow connections” on page 8
- The message tree
- Parsers
- “Properties” on page 9
- Message flow transactions
- “Broker schemas” on page 10
- “Message flow accounting and statistics data” on page 11

Task topics:

- “Designing a message flow” on page 15
- “Creating a message flow project” on page 41
- “Deleting a message flow project” on page 42
- “Creating a broker schema” on page 43
- “Creating a message flow” on page 44
- “Opening an existing message flow” on page 45
- “Defining message flow content” on page 52
- “Copying a message flow using copy” on page 46
- “Saving a message flow” on page 50
- “Renaming a message flow” on page 46
- “Moving a message flow” on page 47
- “Deleting a message flow” on page 48
- “Deleting a broker schema” on page 49
- “Defining a promoted property” on page 65
- “Collecting message flow accounting and statistics data” on page 74

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

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

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

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

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

Message flows overview

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

You define a message flow in the workbench by including a number of message flow nodes, each of which represents a set of actions that define a processing step. The connections in the flow determine which processing steps are carried out, in which order, and under which conditions. A message flow must include an input node that provides the source of the messages that are processed. You must then deploy the message flow to a broker for execution.

You can create a message flow using the built-in nodes or other message flows (known as subflows). When you want to invoke a message flow to process messages, you deploy it to a broker, where it is executed within an execution group.

The following topics describe the concepts that you need to understand to design, create, and configure a message flow and its associated resources:
- Projects
- Nodes
- Version and keywords
- "Message flow connections" on page 8
- "Properties" on page 9
- Accounting and statistics data
- "Converting data with message flows" on page 29

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

Message flow projects

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

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

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

Import one of the samples from the Samples Gallery to see how the sample's message flow resources are stored in a Message Flow project. If the sample has a message set, its message set resources are stored in a Message Set project. For example, import the Video Rental sample or the Comma Separated Value (CSV) sample; both of these samples have a Message Flow project and a Message Set project.

Message flow nodes

A message flow node is a processing step in a message flow.

It receives a message, performs a set of actions against the message, and optionally passes the message on to the next node in the message flow. A message flow node can be a built-in node , a user-defined node, or a subflow node.

A message flow node has a fixed number of input and output points known as terminals. You can make connections between the terminals to define the routes that a message can take through a message flow.

Built-in node

A built-in node is a message flow node that is supplied by WebSphere Event Broker. The built-in nodes provide input and output functions.

For information on all the built-in nodes supplied by WebSphere Event Broker, see "Built-in nodes" on page 126.

User-defined node

A user-defined node is an extension to the broker that provides a new message flow node in addition to those supplied with the product. It must be written to the user-defined node API provided by WebSphere Event Broker for both C and Java languages.

Subflow

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

The subflow, when it is embedded in a main flow, is represented by a subflow node, which has a unique icon. The icon is displayed with the correct number of terminals to represent the Input and Output nodes that you have included in the subflow definition.

The use of subflows is demonstrated in the Error Handler sample and the Coordinated Request Reply sample. The Error Handler sample uses a subflow to trap information about errors and store the information in a database. The Coordinated Request Reply sample uses a subflow to encapsulate the storage of the ReplyToQ and ReplyToQMgr values in a WebSphere MQ message so the processing logic can be reused in other message flows and to allow alternative implementations to be substituted.

A node does not always produce an output message for every output terminal: often it produces one output for a single terminal based on the message received or the result of the operation of the node.

If more than one terminal is connected, the node sends the output message on each terminal, but sends on the next terminal only when the processing has completed for the current terminal. Updates to a message are never propagated to previously-executed nodes, only to nodes following the node in which the update has been made. The order in which the message is propagated to the different output terminals is determined by the broker; you cannot change this order.

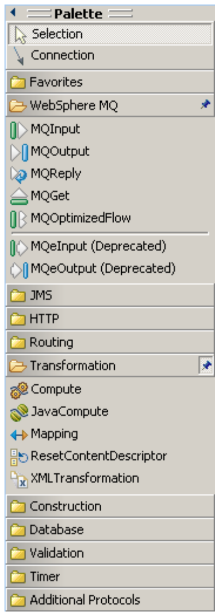
The message flow can accept a new message for processing only when all paths through the message flow (that is, all connected nodes from all output terminals) have been completed.

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

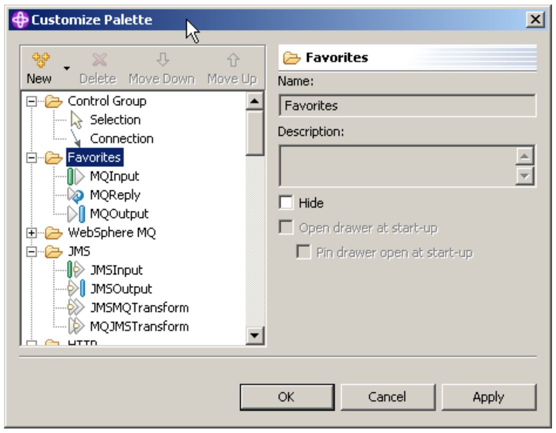
Message flow node palette

The palette contains all of the built-in nodes, which are organized into categories, or drawers. A drawer is a container for a list of icons, such as the **Favorites** drawer. You can drag the nodes that you use most often into the Favorites drawer for easy access. If you create your own nodes, you can also add them to the palette. You can drag a node from the palette onto the canvas, and create a connection between two nodes.

If you right-click the palette, you can add a selected node to the canvas, or customize the appearance and behavior of the palette. The following example shows the palette in List view, using small icons.



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



Control Group
- Selection
- Connection

Favorites
- MQInput
- MQReply
- MQOutput

WebSphere MQ

JMS
- JMSInput
- JMSOutput
- JMSMQTransform
- MQJMSTransform

HTTP

Favorites

Name: Favorites

Description:

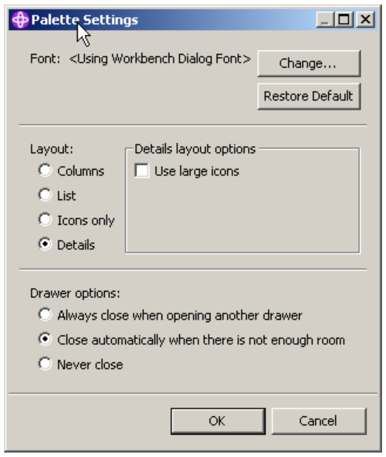
[ ] Hide

[ ] Open drawer at start-up

[ ] Pin drawer open at start-up

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

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



Font: <Using Workbench Dialog Font>

Layout: Columns, List, Icons only, [✓ Details]

Details layout options: [ ] Use large icons

Drawer options:
- Always close when opening another drawer
- [✓ Close automatically when there is not enough room]
- Never close

The following topics explain how to change the palette layout and settings:
- "Changing the palette layout" on page 53
- "Changing the palette settings" on page 53
- "Customizing the palette" on page 54

Message flow version and keywords

When you are developing a message flow, you can define the version of the message flow as well as other key information that you want to be associated with it. After the message flow has been deployed, you can view the properties of the message flow in the workbench. These properties include the deployment and modification dates and times (the default information that is displayed) as well as any additional version or keyword information that you have set.

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

Version

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

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

Keywords

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

- The label property of a Passthrough node
- ESQL comments or string literals
- The Long Description property of the message flow

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

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

The following table contains the information that the workbench shows.

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

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

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

Restrictions within keywords

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

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

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

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

Message flow connections

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

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

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

Bend points

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

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

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

Message flow node terminals

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

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

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

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

Properties

This topic discusses the following types of broker properties:

- Promoted properties: see "Promoted properties."

Promoted properties

A promoted property is a message flow node property that has been promoted to the level of the message flow in which it is included.

A message flow contains one or more message flow nodes, each of which is an instance of a message flow type (a built-in node). You can promote the properties of a message flow node to apply to the message flow to which it belongs. If you do this, any user of the message flow can set values for the properties of the nodes in this higher message flow by setting them at the message flow level, without being aware of the message flow's internal structure.

You can promote compatible properties (that is, properties that represent comparable values) from more than one node to the same promoted property; you can then set a single property that affects multiple nodes.

A subset of message flow node properties is also configurable (that is, the properties can be updated at deploy time). You can promote configurable properties: if you do so, the promoted property (which can have a different name from the property or properties that it represents) is the one that is available to update at deploy time. Configurable properties are those associated with system resources, for example queues: they can be set at deploy time by an administrator rather than a message flow developer.

Broker schemas

A broker schema is a symbol space that defines the scope of uniqueness of the names of resources (message flows) defined within it.

The broker schema is defined as the relative path from the project source directory to the flow name. When you first create a message flow project, a default broker schema named `(default)` is created within the project.

You can create new broker schemas to provide separate symbol spaces within the same message flow project. A broker schema is implemented as a folder, or subdirectory, within the project, and provides organization within that project. You can also use project references to spread the scope of a single broker schema across multiple projects to create an application symbol space that provides a scope for all resources associated with an application suite.

A broker schema name must be a character string that starts with a Unicode character followed by zero or more Unicode characters or digits, and the underscore. You can use the period to provide a structure to the name, for example `Stock.Common`. A directory is created in the project directory to represent the schema, and if the schema is structured using periods, further subdirectories are defined. For example, the broker schema `Stock.Common` results in a directory Common within a directory Stock within the message flow project directory.

If you create a message flow resource in the default broker schema within a project, the file or files associated with that resource are created in the directory that represents the project. If you create a resource in another broker schema, the files are created within the schema directory.

For example, if you create a message flow Update in the default schema in the message flow project Project1, its associated files are stored in the Project1 directory. If you create another message flow in the `Stock.Common` broker schema within the project Project1, its associated files are created in the directory Project1\Stock\Common.

Because each broker schema represents a unique name scope, you can create two message flows that share the same name within two broker schemas. The broker schemas ensure that these two message flows are recognized as separate resources. The two message flows, despite having the same name, are considered unique.

If you move a message flow from one project to another, you can continue to use the message flow within the original project if you preserve the broker schema. If you do this, you must update the list of dependent projects for the original project by adding the target project. If, however, you do not preserve the broker schema, the flow becomes a different flow because the schema name is part of the fully qualified message flow name, and it is no longer recognized by other projects. This action results in broken links that you must manually correct. For further information about correcting errors after moving a message flow, see “Moving a message flow” on page 47.

Do not move resources by moving their associated files in the file system; you must use the workbench to move resources to ensure that all references are corrected to reflect the new organization.

Message flow accounting and statistics data

Message flow accounting and statistics data is the information that can be collected by a broker to record performance and operating details of message flow execution.

These reports are not the same as the publish/subscribe statistics reports that you can generate. The publish/subscribe statistics provide information about the performance of brokers, and the throughput between the broker and clients that are connected to the broker. Message flow accounting and statistics reports provide information about the performance and operating details of a message flow execution.

Message flow accounting and statistics data records dynamic information about the runtime behavior of a message flow. For example, it indicates how many messages are processed and how large those messages are, as well as CPU usage and elapsed processing times. The broker collects the data and records it in a specified location when one of a number of events occurs (for example, when a snapshot interval expires or when the execution group you are recording information about stops).

Accounting and statistics data is collected only for message flows that start with an MQInput, HTTPInput, or user-defined input node. If you start data collection for a message flow that starts with this node, the data is collected for all built-in and user-defined nodes, including those in subflows. If the message flow starts with another input node (for example, a Real-timeInput node), no data is collected (and no error is reported).

Collecting message flow accounting and statistics data is optional; by default it is switched off. To use this facility, request it on a message flow or execution group basis. The settings for accounting and statistics data collection are reset to the defaults when an execution group is redeployed. Previous settings for message flows in an execution group will not be passed on to the new message flows deployed to that execution group. Data collection is started and stopped dynamically when you issue the mqsichangeflowstats command; you do not need to make any change to the broker or to the message flow, or redeploy the message flow, to request statistics collection.

You can activate data collection on both your production and test systems. If you collect the default level of statistics (message flow), the impact on broker performance is minimal. However, collecting more data than the default message flow statistics can generate high volumes of report data that might cause a small but noticeable performance overhead.

When you plan data collection, consider the following points:
- Collection options

- Accounting origin
- Output formats

You can find more information on how to use accounting and statistics data to improve the performance of a message flow in this developerWorks article on message flow performance.

The following SupportPac provides additional information about using accounting and statistics:
- Using statistics and accounting SupportPac (IS11)

Message flow accounting and statistics collection options

The options that you specify for message flow accounting and statistics collection determine what information is collected. You can request the following types of data collection:

- Snapshot data is collected for an interval of approximately 20 seconds. The exact length of the interval depends on system loading and the level of current broker activity. You cannot modify the length of time for which snapshot data is collected. At the end of this interval, the recorded statistics are written to the output destination and the interval is restarted.
- Archive data is collected for an interval that you have set for the broker on the mqsicreatebroker or mqsichangebroker command. You can specify an interval of between 10 and 14400 minutes, the default value is 60 minutes. At the end of this interval, the recorded statistics are written to the output destination and the interval is restarted.

  An interval is prematurely expired and restarted when any of the following events occur:
  - The message flow is redeployed.
  - The set of statistics data to be collected is modified.
  - The broker is shut down.

  This preserves the integrity of the data already collected when that event occurs.

  z/OS On z/OS, you can set the command parameter to 0, which means that the interval is controlled by an external timer mechanism. This support is provided by the Event Notification Facility (ENF), which you can use instead of the broker command parameter if you want to coordinate the expiration of this timer with other system events.

You can request snapshot data collection, archive data collection, or both. You can activate snapshot data collection while archive data collection is active. The data recorded in both reports is the same, but is collected for different intervals. If you activate both snapshot and archive data collection, be careful not to combine information from the two different reports, because you might count information twice.

You can use the statistics generated for the following purposes:

- You can record the load that applications, trading partners, or other users put on the broker. This allows you to record the relative use that different users make of the broker, and perhaps to charge them accordingly. For example, you could levy a nominal charge on every message that is processed by a broker, or by a specific message flow.

  Archive data provides the information that you need for a use assessment of this kind.
- You can assess the execution of a message flow to determine why it, or a node within it, is not performing as you expect.

Snapshot data is appropriate for performance assessment.

- You can determine the route that messages are taking through a message flow. For example, you might find that an error path is taken more frequently than you expect and you can use the statistics to understand when the messages are routed to this error path.

  Check the information provided by snapshot data for routing information; if this is insufficient for your needs, use archive data.

Message flow accounting and statistics accounting origin

Accounting and statistics data can be identified by the account identifier of the originator. The accounting origin for all accounting and statistics data for all message flows is set to `Anonymous`. You cannot change this value.

Output formats for message flow accounting and statistics data

When you collect message flow statistics, you can choose the output destination for the data:

- User trace
- XML publication
- SMF

Statistics data is written to the specified output location in the following circumstances:

- When the archive data interval expires.
- When the snapshot interval expires.
- When the broker shuts down. Any data that has been collected by the broker, but not yet written to the specified output destination, is written during shutdown. It might therefore represent data for an incomplete interval.
- When any part of the broker configuration is redeployed. Redeployed configuration data might contain an updated configuration that is not consistent with the existing record structure (for example, a message flow might include an additional node, or an execution group might include a new message flow). Therefore the current data, which might represent an incomplete interval, is written to the output destination. Data collection continues for the redeployed configuration until you change data collection parameters or stop data collection.
- When data collection parameters are modified. If you update the parameters that you have set for data collection, all data collected for the message flow (or message flows) is written to the output destination to retain data integrity. Statistics collection is restarted according to the new parameters.
- When an error occurs that terminates data collection. You must restart data collection yourself in this case.

User trace

You can specify that the data collected is written to the user trace log. The data is written even if trace is currently switched off. The default output destination for accounting and statistics data is the user trace log. The data is written to one of the following locations:

- Windows On Windows systems, if the broker workpath has been set using the -w option of the mqsicreatebroker command, data is written to workpath\log. If the broker workpath has not been specified, data is written to *install_dir*\log, where *install_dir* is the directory in which WebSphere Event Broker is installed.
- UNIX On UNIX systems, data is written to /var/wmqi/log.

- Linux On Linux systems, data is written to /var/wmqi/log.
- z/OS On z/OS systems, data is written to /component_filesystem/log.

XML publication

You can specify that the data collected is published. The publication message is created in XML format and is available to subscribers registered in the broker network that subscribe to the correct topic.

The topic on which the data is published has the following structure:

```
$SYS/Broker/brokerName/StatisticsAccounting/recordType/executionGroupLabel/messageFlowLabel
```

The variables correspond to the following values:

brokerName
The name of the broker for which statistics are collected.

recordType
Set to `Snapshot` or `Archive`, depending on the type of data to which you are subscribing. Alternatively, use # to register for both snapshot and archive data if it is being produced.

executionGroupLabel
The name of the execution group for which statistics are collected.

messageFlowLabel
The label on the message flow for which statistics are collected.

Subscribers can include filter expressions to limit the publications that they receive. For example, they can choose to see only snapshot data, or to see data collected for a single broker. Subscribers can specify wild cards (+ and #) to receive publications that refer to multiple resources.

The following examples show the topic with which a subscriber should register to receive different sorts of data:

- Register the following topic for the subscriber to receive data for all message flows running on *BrokerA*:

  $SYS/Broker/*BrokerA*/StatisticsAccounting/#
- Register the following topic for the subscriber to receive only archive statistics relating to a message flow *Flow1* running on execution group *Execution* on broker *BrokerA*:

  $SYS/Broker/*BrokerA*/StatisticsAccounting/Archive/*Execution*/*Flow1*
- Register the following topic for the subscriber to receive both snapshot and archive data for message flow *Flow1* running on execution group *Execution* on broker *BrokerA*

  $SYS/Broker/*BrokerA*/StatisticsAccouting/#/*Execution*/*Flow1*

Message display, test and performance utilities SupportPac (IH03) can help you with registering your subscriber.

SMF

On z/OS, you can specify that the data collected is written to SMF. SMF supports the collection of data from multiple subsystems, and you might therefore be able to synchronize the information recorded. When you want to interpret the information recorded, you can use any utility program that processes SMF records. Accounting and statistics data uses SMF type 117 records.

Designing a message flow

Before you start:

Read the concept topic about message flow nodes.

When you design a message flow, consider several design factors which include some or all of the following options:

- Which nodes provide the function that you require. In many cases, you can choose between several nodes that provide a suitable function. You might have to consider other factors listed here to determine which node is best for your overall needs. You can include built-in nodes, user-defined nodes, and subflow nodes. For more information, see “Deciding which nodes to use.”
- Whether it is appropriate to include more than one input node. For more information, see “Using more than one input node” on page 18.
- Whether you can use subflows; for example, to define a specific output node with common properties. For more information, see “Using subflows” on page 24.
- What response times your applications expect from the message flow. This factor is influenced by several aspects of how you configure your nodes and the message flow. For more information, see “Optimizing message flow response times” on page 25.
- Whether to use WebSphere MQ cluster queues. For more information, see “Using WebSphere MQ cluster queues for input and output” on page 27.
- Whether to use WebSphere MQ shared queues on z/OS. The use of shared queues is described further in “Using WebSphere MQ shared queues for input and output (z/OS)” on page 29.
- Whether you want your messages to go through data conversion. The available options are described in “Configuring message flows for data conversion” on page 30.
- What steps to take to ensure that messages are not lost. For more information, see “Ensuring that messages are not lost” on page 32.
- How errors are handled within the message flow. You can use the facilities provided by the broker to handle any errors that are encountered during message flow execution (for example, if the input node fails to retrieve an input message, or if writing to a database results in an error). However, you might prefer to design your message flow to handle errors in a specific way. For more information, see “Handling errors in message flows” on page 34.

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

Deciding which nodes to use

Before you start:

Read the concept topic about message flow nodes.

WebSphere Event Broker includes a large number of message processing nodes that you can use within your message flows. You can also choose from user-defined nodes that have been created and supplied by users, or other vendors and companies.

Your decision about which nodes to use depends on the processing that you want to perform on your messages.

Input and output nodes

Input and output nodes define points in the message flow to which clients send messages (input nodes such as MQInput) and from which clients receive messages (output nodes such as MQOutput). Client applications interact with these nodes by putting messages to, or getting messages from, the I/O resource that is specified by the node as the source or target of the messages. Although a message flow must include at least one input node, it does not have to include an output node.

- If you are creating a message flow that you want to deploy to a broker, you must include at least one input node to receive messages. The input node that you choose depends on the source of the input messages and where in the flow you want to receive the messages:

  MQInput

  If the messages arrive at the broker on a WebSphere MQ queue and the node is to be at the start of a message flow.

  The use of message flows that contain MQeInput nodes in WebSphere Event Broker Version 6.0 is deprecated. Redesign your message flows to remove the MQe nodes and replace them with MQ nodes that are configured to your own specifications and coordinated with your MQe Gateway configuration. For more details, see Migrating a message flow that contains WebSphere MQ Everyplace nodes.

  MQGet

  If the messages arrive at the broker on a WebSphere MQ queue and the node is not to be at the start of a message flow.

  SCADAInput

  If the messages are sent by a telemetry device.

  Real-timeInput or Real-timeOptimizedFlow

  If the messages are sent by a JMS or multicast application.

  The Real-timeInput node is an input node and the Real-timeOptimizedFlow node is a complete message flow that provides a high performance publish/subscribe message flow.

  JMSInput

  If the messages are sent by a JMS application.

  User-defined input node

  If the message source is a client or application that uses a different protocol or transport.

  Input node

  If you are creating a message flow that you want to embed in another message flow (a subflow) that you will not deploy as a standalone message flow, you must include at least one Input node to receive messages into the subflow.

An instance of the Input node represents an in terminal. For example, if you have included one instance of the Input node, the subflow icon shows one in terminal that you can connect to other nodes in the main flow in the same way that you connect any other node.

You can deploy only message flows that have at least one input node. If your message flow does not contain an input node, you are prevented from adding it to the broker archive file. The input node can be in the main flow, or in a message flow that is embedded in the main flow.

You can use more than one input node in a message flow. For more information, see "Using more than one input node" on page 18.

- If you want to send the messages produced by the message flow to a target application, you can include one or more output nodes. The one that you choose depends on the transport across which the target application expects to receive those messages:

  Publication
  If you want to distribute the messages using the publish/subscribe network for applications that subscribe to the broker across all supported protocols. A Publication node is an output node that use output destinations that are identified by subscribers whose subscriptions match the characteristics of the current message.

  MQOutput
  If the target application expects to receive messages on a WebSphere MQ queue, or on the WebSphere MQ reply-to queue specified in the input message MQMD.

  The use of message flows that contain MQeOutput nodes in WebSphere Event Broker Version 6.0 is deprecated. Redesign your message flows to remove the MQe nodes and replace them with MQ nodes that are configured to your own specifications and coordinated with your MQe Gateway configuration. For more details, see Migrating a message flow that contains WebSphere MQ Everyplace nodes.

  MQReply
  If the target application expects to receive messages on the WebSphere MQ reply-to queue specified in the input message MQMD.

  SCADAOutput
  If a telemetry device is the target of the output messages, and the Publication node is not suitable.

  Real-timeOptimizedFlow
  If the target application is a JMS or multicast application.

  JMSOutput
  If the messages are for a JMS destination.

  User-defined output node
  If the target is a client or application that uses a different protocol or transport.

Output node

If you are creating a message flow that you want to embed in another message flow (a subflow) that you will not deploy as a standalone message flow, you must include at least one Output node to propagate messages to subsequent nodes that you connect to the subflow.

An instance of the Output node represents an out terminal. For example, if you have included two instances of the Output node, the subflow icon shows two out terminals that you can connect to other nodes in the main flow in the same way that you connect any other node.

Using more than one input node

Before you start:

Read the concept topic about message flow nodes.

You can include more than one input node in a single message flow. You might find this useful in the following situations:

- The message flow provides common processing for messages that are received across multiple transports. For example, a single message flow might handle:
  - Data in messages received across WebSphere MQ, and therefore through a WebSphere MQ queue and an MQInput node
  - Messages that are received across native IP connections (a Real-timeInput node)
- You need to set standard properties on the MQInput node if input messages:
  - are all received across WebSphere MQ, and
  - do not include an MQRFH2 header.

  If the required standard properties are not always the same for every message, you can include more than one input node and configure each to handle a particular set of properties.
- Each input node in a message flow causes the broker to start a separate thread of execution. Including more than one input node might improve the message flow performance. However, if you include multiple input nodes that access the same input source (for example, a WebSphere MQ queue), the order in which the messages are processed cannot be guaranteed. If you want the message flow to process messages in the order in which they are received, this option is not appropriate.

  If you are not concerned about message order, consider using additional instances of the same message flow rather than multiple input nodes. If you set the *Additional Instances* property of the message flow when you deploy it to the broker, multiple copies of the message flow are started in the execution group. This is the most efficient way of handling multiple instances.

The Scribble sample uses two input nodes: an MQInput node and a Real-timeInput node. This enables the sample's message flow to accept input across both WebSphere MQ transport and native IP connections.

Configuring JMSInput and JMSOutput nodes to support global transport

If you require additional configuration to enable global transaction support for the JMSInput and JMSOutput node, you must complete the following steps:

1. Set the Message Flow property *Coordinated Transaction* to *yes*.
2. For each JMSInput or JMSOutput node required in the global transaction, set the Advanced property *Transaction Mode* to *Global*.
3. Create a Queue Connection Factory and supply either a default name, *recoverXAQCF* or supply a user defined name. See JMSInput or JMSOutput node for further details on creating JNDI administered objects.
4. On platforms other than z/OS, you must set up a stanza for each JMS provider that you want to use, prior to deployment. Select the appropriate link for details of this task on the platform, or platforms, that your enterprise uses:
   - "Linux and UNIX systems"
   - "Windows systems" on page 21

   For further information, see:
   - "Configuring for coordinated transactions" on page 132 within the JMSInput node topic
   - "Configuring for coordinated transactions" on page 141 within the JMSOutput node topic.

   On WebSphere Event Broker for z/OS, the only JMS provider supported is the IBM WebSphere MQ Java Client and the only transport mode supported for that client is BIND mode; no further configuration steps are required

The JMS provider can supply additional jar files that are required for transactional support; see the documentation supplied with the JMS provider for more information. For example, on platforms other than z/OS, the WebSphere MQ JMS provider supplies an extra jar file `com.ibm,mqetclient.jar`.

You must also add any additional jar files to the broker `shared_classes` directory. On Windows, this directory is C:\Documents and Settings\All Users\Application Data\IBM\MQSI\shared-classes. For more information, see the section on making the JMS provider client available to the JMS Nodes in "JMSInput node" on page 128.

Choice of JMS Provider

Any JMS provider that conforms to the Java Message Service Specification, version 1.1 and that supports the JMS XAResource API through the JMS session can be used if transaction coordination is required.

If the message designer has specified a non XA-compliant provider, the non transactional mode only is supported. In this case, you must set the *Transaction mode* property to *None* for all JMSInput and JMSOutput nodes.

Linux and UNIX systems

For the broker's queue manager, place a stanza entry in an initialization file, for example `qm.ini`, for each JMS provider that can be used by a JMSInput node.

You must specify a stanza in the broker's queue manager `.ini` file for each JMS provider that you want to use. There must be one stanza for each new JMS provider, where the JMS provider can be specified by any JMSInput or JMSOutput node included in a message flow that is running on a broker.

The parameters that are supplied on `XAOpenString` are comma delimited and positional. Any missing optional parameter must be represented by a comma if other parameters are provided later in the string.

The following stanza entry is an example you can add when using WebSphere MQ Java as the JMS provider:

```
XAResourceManager:
 Name=WBIWMQJMS
    SwitchFile=/<Installation Path>/lib/JMSSwitch.so
    XAOpenString=<Initial Context Factory>,
              <location of JNDI bindings>'
              <LDAP Principal>,
              <LDAP Credentials>,
              <Recovery Connection Factory Name>,
              <JMS Principal>,
              <JMS Credentials>
              ThreadOfControl=THREAD
```

where:

<Installation Path>
: Is the location of the WebSphere Event Broker installation. This value is mandatory where the LDAP parameters are omitted, but a user defined Queue Connection Factory is specified for recovery.

<Initial Context Factory>
: Is the Initial Context Factory identifier for the JMS provider; this value is required.

<Location of JNDI bindings>
: Is either the file path to the bindings file, or the LDAP directory location of the JNDI administered objects that can be used to create an initial context factory for the JMS connection. When supplying the file path to the bindings file, do not include the file name. See the JMSInput or JMSOutput node for further details on creating the JNDI administered objects; this value is required.

<LDAP Principal>
: Is an optional parameter used to specify the principal (user ID) that might be required when an LDAP database is used to hold the JNDI administered objects.

<LDAP Credentials>
: Is an optional parameter used to specify the Credentials (password) that might be required if a password protected LDAP database is used to hold the JNDI administered objects.

<Recovery Connection Factory Name>
: Is an optional parameter used to specify the name of a Queue Connection Factory object in the JNDI administered objects for recovery purposes, when the non default name is required.

<JMS Principal>
: Is an optional parameter for the user ID required to connect to a JMS provider, using a secure JMS Connection Factory.

<JMS Credentials>
Is an optional parameter for the password required to connect to the same JMS provider in conjunction with the JMS principal.

The values for the Initial Context factory and Location of JNDI bindings in the stanza must match those specified in the JMSInput or JMSOutput nodes in the message flows.

Any LDAP parameters must match those that have been specified by using the mqsicreatebroker or mqsichangebroker command.

The Recovery Factory Name must match a Queue Connection Factory name that is created in the JNDI administered objects. If this is omitted, a default factory called `recoverXAQCF` is used. In either case this value must refer to a JNDI administered object that has already been created.

The JMS Principal and JMS Credentials must be configured together

Following is an example format of a stanza in the `qm.ini` file that describes a JMS provider for global transactions:

```
XAResourceManager:
    Name=XAJMS_PROVIDER1
    SwitchFile=/opt/var/mqsi/lib/JMSSwitch.so
    XAOpenString= com.sun.jndi.fscontext.RefFSContextFactory,
              /Bindings/JMSProvider1_Bindings_Directory,
              ,
              ,
              ,
              myJMSuser1,
              passwd
              ThreadOfControl=THREAD
```

where:

XAJMS_PROVIDER1
Is the user-defined name for the resource manager

/opt/var/mqsi
Is the <Installation Path>

com.sun.jndi.fscontext.RefFSContextFactory
Is the <Initial Context Factory>

/Bindings/JMSProvider1_Bindings_Directory
Is the location of the bindings

myJMSuser1
Is the <JMS Principal>

passwd
Is the password used in <JMS Credentials>

In this example the optional fields `<LDAP Principal>`, `<LDAP Credentials>`, and `<Recovery Connection Factory Name>` are not required, so only the positional comma delimiters are configured in the `XAOpenString` stanza.

Windows systems

On Windows you use the WebSphere MQ Explorer or WebSphere MQ Services snap-in, depending on which version of WebSphere MQ you are using, to configure the stanza.

The Switch file is called JMSSwitch.dll; see Refer to the WebSphere MQ System Administration Guide for details on how to update the qm.ini file.

The extra entry, called the XACloseString, should match the values provided for the XAOpenString.

For the broker's queue manager, place a stanza entry in an initialization file, for example qm.ini, for each JMS provider that can be used by a JMSInput node.

You must specify a stanza in the broker's queue manager .ini file for each JMS provider that you want to use. There must be one stanza for each new JMS provider, where the JMS provider can be specified by any JMSInput or JMSOutput node included in a message flow that is running on a broker.

The parameters that are supplied on XAOpenString are comma delimited and positional. Any missing optional parameter must be represented by a comma if other parameters are provided later in the string.

The following stanza entry is an example you can add when using WebSphere MQ Java as the JMS provider:

```
XAResourceManager:
 Name=WBIWMQJMS
    SwitchFile=/<Installation Path>/lib/JMSSwitch.so
    XAOpenString=<Initial Context Factory>,
              <location of JNDI bindings>'
              <LDAP Principal>,
              <LDAP Credentials>,
              <Recovery Connection Factory Name>,
              <JMS Principal>,
              <JMS Credentials>
              ThreadOfControl=THREAD
```

where:

<Installation Path>
Is the location of the WebSphere Event Broker installation. This value is mandatory where the LDAP parameters are omitted, but a user defined Queue Connection Factory is specified for recovery.

<Initial Context Factory>
Is the Initial Context Factory identifier for the JMS provider; this value is required.

<Location of JNDI bindings>
Is either the file path to the bindings file, or the LDAP directory location of the JNDI administered objects that can be used to create an initial context factory for the JMS connection. When supplying the file path to the bindings file, do not include the file name. See the JMSInput or JMSOutput node for further details on creating the JNDI administered objects; this value is required.

<LDAP Principal>
Is an optional parameter used to specify the principal (user ID) that might be required when an LDAP database is used to hold the JNDI administered objects.

<LDAP Credentials>
Is an optional parameter used to specify the Credentials (password) that might be required if a password protected LDAP database is used to hold the JNDI administered objects.

<Recovery Connection Factory Name>
Is an optional parameter used to specify the name of a Queue Connection Factory object in the JNDI administered objects for recovery purposes, when the non default name is required.

<JMS Principal>
Is an optional parameter for the user ID required to connect to a JMS provider, using a secure JMS Connection Factory.

<JMS Credentials>
Is an optional parameter for the password required to connect to the same JMS provider in conjunction with the JMS principal.

The values for the Initial Context factory and Location of JNDI bindings in the stanza must match those specified in the JMSInput or JMSOutput nodes in the message flows.

Any LDAP parameters must match those that have been specified by using the mqsicreatebroker or mqsichangebroker command.

The Recovery Factory Name must match a Queue Connection Factory name that is created in the JNDI administered objects. If this is omitted, a default factory called `recoverXAQCF` is used. In either case this value must refer to a JNDI administered object that has already been created.

The JMS Principal and JMS Credentials must be configured together

Following is an example format of a stanza in the `qm.ini` file that describes a JMS provider for global transactions:

```
XAResourceManager:
    Name=XAJMS_PROVIDER1
    SwitchFile=/opt/var/mqsi/lib/JMSSwitch.so
    XAOpenString= com.sun.jndi.fscontext.RefFSContextFactory,
              /Bindings/JMSProvider1_Bindings_Directory,
              ,
              ,
              ,
              myJMSuser1,
              passwd
              ThreadOfControl=THREAD
```

where:

XAJMS_PROVIDER1
Is the user-defined name for the resource manager

/opt/var/mqsi
Is the <Installation Path>

com.sun.jndi.fscontext.RefFSContextFactory
Is the <Initial Context Factory>

/Bindings/JMSProvider1_Bindings_Directory
Is the location of the bindings

myJMSuser1
Is the <JMS Principal>

passwd
Is the password used in <JMS Credentials>

In this example the optional fields `<LDAP Principal>`, `<LDAP Credentials>`, and `<Recovery Connection Factory Name>` are not required, so only the positional comma delimiters are configured in the `XAOpenString` stanza.

Using subflows

You can include subflows in your message flows in exactly the same way as you include built-in or user-defined nodes.

You can also connect subflows to other nodes in the same way. You can define a subflow once, and use it in more than one message flow (and even in more than one message flow project), so a subflow can provide the following benefits:

- Reuse and reduced development time.
- A consistent way of achieving a particular function, and increased maintainability of your message flows (consider a subflow as analogous to a programming macro, or to inline code that is written once but used in many places).
- Flexibility. If you promote some or all of the properties of the nodes in the subflow, you can tailor a subflow to a specific context (for example, by updating the output queue information).

However, remember that a subflow is not a single node, and its inclusion increases the number of nodes in the message flow, which might affect its performance.

Consider these examples of subflow use:

- You can define a subflow that provides a common destination for messages that result in an error within the message flow.

Use the Passthrough node to enable versioning of a subflow at run time. The Passthrough node allows you to add a label to your message flow or subflow. By combining this label with keyword replacement from your version control system, you can identify which version of a subflow is included in a deployed message flow. You can use this label for your own purposes. If you have included the correct version keywords in the label, you can see the value of the label:

- Stored in the broker archive (bar) file, using the **mqsireadbar** command
- As last deployed to a particular broker, on the properties of a deployed message flow in the Message Brokers Toolkit
- In the run time, if you enable user trace for that message flow

The message that it propagates on its Out terminal is the same message that it received on its In terminal; for example, if you develop an error processing subflow to include in several message flows, you might want to modify that subflow. However, you might want to introduce the modified version initially to just a subset of the message flows in which it is included. Set a value for the instance of the Passthrough node that identifies which version of the subflow you have included.

The use of subflows is demonstrated in the Error Handler sample and the Coordinated Request Reply sample. The Error Handler sample uses a subflow to trap information about errors and store the information in a database. The Coordinated Request Reply sample uses a subflow to encapsulate the storage of the ReplyToQ and ReplyToQMgr values in a WebSphere MQ message so that the processing logic can be easily reused in other message flows and to allow alternative implementations to be substituted.

Adding keywords to subflows

You can embed keywords in each subflow that you use in a message flow. A different keyword must be used in each instance of a subflow. This is because only the first recorded instance of each keyword within the message flow `.cmf` file is available to Configuration Manager Proxy applications and to the toolkit. The order that subflows appear in the `.cmf` file is not guaranteed.

Optimizing message flow response times

Before you start:

Read the concept topic about message flow nodes.

When you design a message flow, the flexibility and richness of the built-in nodes often means that there are several ways to achieve the processing and therefore the end results that you require. However, you can also find that these different solutions deliver different performance and, if this is an important consideration, you must design for performance as well as function.

There are two ways in which your applications can perceive performance:

1. Response time. This indicates how quickly each message is processed by the message flow. This is particularly influenced by how you design your message flows. This is further discussed in this topic.
2. Throughput. This indicates how many messages of particular sizes can be processed by a message flow in a given time. This is mainly affected by configuration and system resource factors, and is therefore discussed in Optimizing message flow throughput with other domain configuration information.

There are several aspects that influence message flow response times. However, as you create and modify your message flow design to arrive at the best results that meet your specific business requirements, you must also consider the eventual complexity of the message flow. The most efficient message flows are not necessarily the easiest to understand and maintain; experiment with the solutions available to arrive at the best balance for your needs.

Several factors influence message flow response times:

The number of nodes that you include in the message flow

Every node causes some processing overhead, so consider the content of the message flow carefully, including the use of subflows.

Use as few nodes as possible in a message flow; every node that you include in the message flow increases the overhead in the broker. There is an upper limit to the number of nodes within a single flow. This limit is governed by system resources, particularly the stack size.

For more information about stack sizes, see “System considerations for message flow development” on page 26.

The use of persistent and transactional messages

Persistent messages are saved to disk during message flow processing. This is avoided if you can specify that messages either on input, output, or both, are non-persistent. If your message flow is handling only non-persistent messages, check the configuration of the nodes and the message flow itself; if your messages are non-persistent, transactional support might be unnecessary. The default configuration of some nodes enforces transactionality; if you update these properties and redeploy the message flow, response times might improve.

Message size
A larger message takes longer to process. If you can split large messages into smaller chunks of information, you might be able to improve the speed at which they are handled by the message flow. The Large Messaging sample demonstrates how to minimise the virtual memory requirements for the message flow to improve a message flow's performance when processing potentially large messages.

You can find more information on improving the performance of a message flow in this developerWorks article on message flow performance.

System considerations for message flow development

Default stack size

When a message flow thread starts, it requires storage to perform the instructions that are defined by the logic of its connected nodes. This storage comes from the execution group's heap and stack size. The default stack size that is allocated to a message flow thread depends on the operating system that is used.

Windows Each message flow thread is allocated 1 MB of stack space.

Linux Each message flow thread is allocated 8 MB of stack space.

UNIX Each message flow thread is allocated 1 MB of stack space.

z/OS Each message flow thread is allocated 512 KB of downward stack space and 50 KB of upward stack space.

In a message flow, a node typically uses 2 KB of the stack space. A typical message flow can therefore include 250 nodes on z/OS, 500 nodes on UNIX systems and 500 nodes on Windows. This amount can be higher or lower depending on the type of nodes used and the processing that they perform.

Increasing the stack size on Windows and UNIX systems

You can increase the stack size by setting the **MQSI_THREAD_STACK_SIZE** environment variable to an appropriate value. When you restart brokers that are running on the system, they use the new value.

The value of **MQSI_THREAD_STACK_SIZE** that you set is used for every thread that is created within a DataFlowEngine process. If the execution group has a large number of message flows assigned to it and you set a large value for **MQSI_THREAD_STACK_SIZE**, the DataFlowEngine process therefore needs a large amount of storage for the stack.

Increasing the stack size on z/OS

Integrator components on z/OS are compiled using the XPLINKage (extra performance linkage), which adds optimization to the runtime code. However, if the initial stack size is not large enough, then stack extents are used. 128 KB is used in each extent. Ensure that you choose a large enough downward stack size because the performance of XPLINK degrades when stack extents are used.

To determine suitable stack sizes, a component administrator for z/OS can use the LE (Language Environment®) Report Storage tool. To use this tool, you must test a message flow using the **RPTSTG** option with the **_CEE_RUNOPTS** environment variable. Set this option in the component profile (BIPBPROF for a broker) during the development and test of message flows that are intended for production. For example:

```
export _CEE_RUNOPTS=XPLINK\(ON\),RPTSTG(ON)
```

You can then override the default values for the stack sizes on z/OS by altering or adding the **LE_CEE_RUNOPTS** environment variable in the component profile.

When you update the component profile, perform the following steps:

1. Stop the component.
2. Make the necessary changes to the profile.
3. Submit BIPGEN to re-create the ENVFILE.
4. Restart the component.

For example, you can change the default values of 50 K and 512 K in the following line to suit your needs:

```
export _CEE_RUNOPTS=XPLINK(ON),THREADSTACK(ON,50K,15K,ANYWHERE,KEEP,512K,128K)
```

When you use **RPTSTG**, it increases the time that an application takes to run. You should therefore use it as an aid to the development of message flows only, and not in your final production environment. When you have determined the correct stack sizes needed you should remove this option from the **_CEE_RUNOPTS** environment variable.

XPLINK stacks grow downward in virtual storage while the old standard linkage grows upward. If your message flow uses user-defined nodes that have been compiled with the standard linkage convention, set a suitable value for the upward stack size.

Determining the correct stack size

In WebSphere Event Broker, any processing that involves nested or recursive processing can cause extensive usage of the stack. For example, in the following situations you might need to increase the stack size:

- When a message flow is processing a message that contains a large number of repetitions or complex nesting
- When a message flow is executing ESQL that calls the same procedure or function recursively, or when an operator (for example, the concatenation operator) is used repeatedly in an ESQL statement

Using WebSphere MQ cluster queues for input and output

When you design the WebSphere MQ network that underlies your WebSphere Event Broker broker domain, consider whether to use clusters.

The use of queue manager clusters brings the following significant benefits:

1. Reduced system administration

   Clusters need fewer definitions to establish a network; you can set up and change your network more quickly and easily.
2. Increased availability and workload balancing

You can benefit by defining instances of the same queue to more than one queue manager, thus distributing the workload through the cluster.

If you use clusters with WebSphere Event Broker, consider the following:

For SYSTEM.BROKER queues:
The SYSTEM.BROKER queues are defined for you when you create WebSphere Event Broker components, and are not defined as cluster queues. Do not change this attribute.

For broker, Configuration Manager, and User Name Server connectivity:
If you define the queue managers that support your brokers, the Configuration Manager, and the User Name Server to a cluster, you can benefit from the simplified administration provided by WebSphere MQ clusters. You might find this particularly relevant for the brokers in a collective, which must all have WebSphere MQ interconnections.

For message flow input queues:
If you define an input queue as a cluster queue, consider the implications for the order of messages or the segments of a segmented message. The implications are the same as they are for any WebSphere MQ cluster queue. In particular, the application must ensure that, if it is sending segmented messages, all segments are processed by the same target queue, and therefore by the same instance of the message flow at the same broker.

For message flow output queues:

- WebSphere Event Broker always specifies MQOO_BIND_AS_Q_DEF when it opens a queue for output. If you expect segmented messages to be put to an output queue, or want a series of messages to be handled by the same process, you must specify DEFBIND(OPEN) when you define that queue. This ensures that all segments of a single message, or all messages within a sequence, are put to the same target queue and are processed by the same instance of the receiving application.
- If you create your own output nodes, specify MQOO_BIND_AS_Q_DEF when you open the output queue, and DEFBIND(OPEN) when you define the queue, if you need to guarantee message order, or to ensure a single target for segmented messages.

For publish/subscribe:

- If the target queue for a publication is a cluster queue, you must deploy the publish/subscribe message flow to all the brokers on queue managers in the cluster. However, the cluster does not provide any of the failover function to the broker domain topology and function. If a broker to which a message is published, or a subscriber registers, is unavailable, the distribution of the publication or registration is not taken over by another broker.
- When a client registers a subscription with a broker that is running on a queue manager that is a member of a cluster, the broker forwards a proxy registration to its neighbors within the broker domain; the registration details are not advertised to other members of the cluster.
- A client might choose to become a clustered subscriber, so that its subscriber queue is one of a set of clustered queues that receive any given publication. In this case, when registering a subscription, use the name of an ″imaginary″ queue manager that is associated with the cluster; this is not the queue manager to which the publication will be sent, but an alias for the broker to use. As an administrative activity, a blank queue manager alias definition is made for this queue manager on

the broker that satisfies this subscription for all clustered subscribers. When the broker publishes to a subscriber queue that names this queue manager, resolution of the queue manager name results in the publication being sent to any queue manager that hosts the subscriber cluster queue, and only one clustered subscriber receives the publication.

For example, if the clustered subscriber queue was SUBS_QUEUE and the ″imaginary″ subscriber queue manager was CLUSTER_QM, the broker definition would be:

```
DEFINE QREMOTE(CLUSTER_QM) RQMNAME(' ') RNAME(' ')
```

This sends broker publications for SUBS_QUEUE on CLUSTER_QM to one instance of the cluster queue named SUBS_QUEUE anywhere in the cluster.

To understand more about clusters, and the implications of using cluster queues, see the *WebSphere MQ Queue Manager Clusters* book.

Using WebSphere MQ shared queues for input and output (z/OS)

On z/OS systems you can define WebSphere MQ shared queues as input and output queues for message flows.

Use the WebSphere MQ for z/OS product facilities to define these queues and specify that they are shared.

For more information about configuring on z/OS, refer to the *WebSphere MQ for z/OS Concepts and Planning Guide*.

Using shared queues helps to provide failover support between different images running WebSphere Event Broker on a sysplex.

You cannot use shared queues for broker or User Name Server component queues such as SYSTEM.BROKER.CONTROL.QUEUE.

Shared queues are available only on z/OS.

Converting data with message flows

Data conversion is the process by which data is transformed from the format recognized by one operating system into that recognized by a second operating system with different characteristics such as numeric order.

If you are using a network of systems that use different methods for storing numeric values, or you need to communicate between users who view data in different code pages, you must consider how to implement data conversion.

Numeric order

For numeric and encoding aspects, consider:

- Big Endian versus Little Endian
- Encoding values in WebSphere MQ (the Encoding field in the MQMD)

  Encoding values are system specific. For example, Windows usually has an encoding of 546, hexadecimal value X’00000222’. The three final hexadecimal digits identify:

  1. The float number format

This value can be 1 (IEEE format byte order normal), 2 (IEEE format byte order reversed), or 3 (zSeries format byte order normal). Note that operations on floating point numbers, whether IEEE or z/Series (S/390) format, are subject to rounding error.

2. The packed decimal number format

   This value can be 1 (byte order normal) or 2 (byte order reversed).

3. The hexadecimal number format

   This value can be 1 (byte order normal) or 2 (byte order reversed).

The bit order within a byte is never reversed. Byte order normal means that the least significant digit occupies the highest address.

Systems that process numbers in normal byte order are Big Endian (z/Series, iSeries, Linux, and UNIX). Systems that process numbers in reversed byte order are Little Endian (mainly PCs).

For further details about numeric order, see Appendix D, Machine Encodings, in the *WebSphere MQ Application Programming Reference*.

Code page conversions

Code page conversion might be required for any of the following reasons:
- ASCII versus EBCDIC
- National languages
- Operating system specific code pages

For more information about code page support in WebSphere MQ, see the *WebSphere MQ Application Programming Reference* book.

When you use WebSphere Event Broker, you can use the data conversion facilities of WebSphere MQ.

WebSphere MQ facilities

Headers and message body are converted according to the MQMD values, and other header format names. You might have to set up data conversion exits to convert the body of your messages.

When you use WebSphere MQ facilities, the whole message is converted to the specified encoding and CCSID, according to the setting of the format in the WebSphere MQ header.

For more detail about data conversion using WebSphere MQ facilities, see Appendix F, Data Conversion, in the *WebSphere MQ Application Programming Reference*.

Configuring message flows for data conversion

If you exchange messages between applications that run on systems that are incompatible in some way, you can configure your system to provide data conversion as the message passes through the broker. Data conversion might be necessary if either of the following two values are different on the sending and receiving systems:

1. CCSID. The Coded Character Set Identifier refers to a set of coded characters and their code point assignments. WebSphere Event Broker can process and construct application messages in any code page for which WebSphere MQ provides conversion to and from Unicode, on all operating systems. For more information about code page support, see the *WebSphere MQ Application Programming Reference*.

This behavior might be affected by the use of other products in conjunction with WebSphere Event Broker. Check the documentation for other products, including any databases that you use, for further code page support information.

2. Encoding. This defines the way in which a machine encodes numbers, that is binary integers, packed-decimal integers, and floating point numbers. Numbers that are represented as characters are handled in the same way as all other string data.

If the native CCSID and encoding on the sending and receiving systems are the same, you do not need to invoke data conversion processes.

WebSphere Event Broker and WebSphere MQ provide data conversion facilities to support message exchange between unlike systems. Your choice of which facilities to use depends on the characteristics of the messages that are processed by the message flow:

- Messages that contain text only
- Message that include numerics

Messages that contain text only

Read this section if your messages are WebSphere MQ messages that contain all text (character data or string). If WebSphere MQ supports the systems on which both sending and receiving applications are running for data conversion, use WebSphere MQ facilities. This provides the most efficient data conversion option.

The default behavior of WebSphere MQ is to put messages to queues specifying the local system CCSID and encoding. Applications issuing MQGET can request that the queue manager provides conversion to their local CCSID and encoding as part of get processing.

To use this option:

1. Design messages to be text-only. If you are using COBOL, move numeric fields to USAGE DISPLAY to put them into string form.
2. Set the Format field in the MQMD to MQFMT_STRING (value MQSTR).
3. Issue MQGET with MQGMO_CONVERT in the receiving application. If you prefer, you can convert when the message is received by the broker, by setting the *Convert* property of the MQInput node to yes (by selecting the check box).

If you require more sophisticated data conversion than WebSphere MQ provides in this way (for example, to an unsupported code page), use WebSphere MQ data conversion exits. For more information about these, see the *WebSphere MQ Application Programming Reference*.

Messages that include numerics

Read this section if your messages include numeric data, or are text only but are not WebSphere MQ messages. If your messages are WebSphere MQ messages that include numeric data, you can use WebSphere MQ data conversion exits. If the messages are not WebSphere MQ messages and are text only, or text and numeric, you must use procedures invoked by your own sending or receiving applications.

1. Define the output message in the MRM domain. You can create this definition in one of the following ways:

- Import an external message definition (for example a C header or COBOL copybook).
- Create the message model in the message definition editor.

2. Configure a message flow to receive and process this message:
   a. If you include an MQInput node, do not request conversion by this node.
   b. Include a Compute node in the message flow to create the output message with the required content:
      - If the output message is a WebSphere MQ message, code ESQL in the Compute node to set the CCSID and encoding for the target system in the MQMD.

        For example, to set values for a target z/OS system running with CCSID of 37 and encoding of 785:

        ```
        SET OutputRoot.MQMD.CodedCharSetId = 37;
        SET OutputRoot.MQMD.Encoding = 785;
        ```
      - If the output message is not a WebSphere MQ message, code ESQL in the Compute node to set the CCSID and encoding for the target system in the Properties folder.

Ensuring that messages are not lost

It is important to safeguard that messages that flow through your broker domain. This is true of both application-generated messages and those used internally for inter-component communication. Messages used internally between components always use the WebSphere MQ protocol. Application messages can use all supported transport protocols.

For application and internal messages travelling across WebSphere MQ, two techniques protect against message loss:

- Message persistence

  If a message is persistent, WebSphere MQ ensures that it is not lost when a failure occurs, by copying it to disk.
- Syncpoint control

  An application can request that a message is processed in a synchronized unit-of-work (UOW)

For more information about how to use these options, refer to the *WebSphere MQ System Administration Guide*.

Internal messages

WebSphere Event Broker components use WebSphere MQ messages to communicate events and data between broker processes and subsystems, and the Configuration Manager and User Name Server. The components ensure that the WebSphere MQ features are exploited to protect against message loss. You do not need to take any additional steps to configure WebSphere MQ or WebSphere Event Broker to protect against loss of internal messages.

Application messages

If delivery of application messages is critical, you must design application programs and the message flows that they use to ensure that messages are not lost. The techniques used depend on the protocol used by the applications.

WebSphere MQ Enterprise Transport and WebSphere MQ Mobile Transport

If you are using the built-in input nodes that accept messages across the WebSphere MQ or WebSphere MQ Everyplace protocols, you can use the following guidelines and recommendations:

- Using persistent messages

  WebSphere MQ messaging products provide *message persistence*, which defines the longevity of the message in the system and guarantees message integrity. Nonpersistent messages are lost in the event of system or queue manager failure. Persistent messages are always recovered if a failure occurs.

  You can control message persistence in the following ways:

  – Program your applications that put messages to a queue using the MQI or AMI to indicate that the messages are persistent.

  – Define the input queue with message persistence as the default setting.

  – Configure the output node to handle persistent messages.

  – Program your subscriber applications to request message persistence.

  When an input node reads a message is read from an input queue, the default action is to use the persistence defined in the WebSphere MQ message header (MQMD), that has been set either by the application creating the message, or by the default persistence of the input queue. The message retains this persistence throughout the message flow, unless it is changed in a subsequent message processing node.

  You can override the persistence value of each message when the message flow terminates at an output node. This node has a property that allows you to specify the message persistence of each message when it is put to the output queue, either as the required value, or as a default value. If you specify the default, the message takes the persistence value defined for the queues to which the messages are written.

  If a message passes through a Publication node, the persistence of messages sent to subscribers is determined by the subscribers' registration options. If a subscriber has requested persistent message delivery, and is authorized to do so by explicit or implicit (inherited) ACL, the message is delivered persistently regardless of its existing persistence property. Also, if the user has requested nonpersistent message delivery, the message is delivered nonpersistent regardless of its existing persistence property.

- Processing messages under syncpoint control

  The default action of a message flow is to process incoming messages under syncpoint in a broker-controlled transaction. This means that a message that fails to be processed for any reason is backed out by the broker. Because it was received under syncpoint, the failing message is reinstated on the input queue and can be processed again. If the processing fails, the error handling procedures that are in place for this message flow (defined either by how you have configured the message flow, or by the broker) are executed.

  For full details of input node processing, see "Managing errors in the input node" on page 36.

WebSphere MQ Telemetry Transport

If you are using the built-in input node SCADAInput that accepts messages from telemetry devices across the MQIsdp protocol, this protocol does not have a concept of queues. Clients connect to a SCADAInput node by specifying the port number on which the node is listening. Messages are sent to clients using a `clientId`. However, you can specify a maximum `QoS` (Quality of Service) within a SCADA subscription message, which is similar to persistence:

- **`QoS0`** Nonpersistent.
- **`QoS1`** Persistent, but might be delivered more than once
- **`QoS2`** Once and once only delivery

If a persistent SCADA message is published, it might be downgraded to the highest level that the client can accept. In some circumstances this might mean that the message becomes nonpersistent.

WebSphere MQ Real-time Transport and WebSphere MQ Multicast Transport
If you are using the built-in input nodes Real-timeInput and Real-timeOptimizedFlow that accept messages from JMS and multicast applications, no facilities are available to protect against message loss. You can, however, provide recovery procedures by configuring the message flow to handle its own errors.

Other transports and protocols
For user-defined input nodes that receive messages from another transport protocol, you must rely on the support provided by that transport protocol, or use the recovery procedures provided by the supplier of the user-defined nodes.

Handling errors in message flows

The broker provides basic error handling for all your message flows. If basic processing is not sufficient, and you want to take specific action in response to certain error conditions and situations, you can enhance your message flows to provide your own error handling.

The options that you can use to do this are quite complex in some cases. The options that are provided for MQInput nodes are extensive because these nodes deal with persistent messages and transactions. MQInput is also affected by configuration options for WebSphere MQ.

Because you can decide to handle different errors in different ways, there are no fixed procedures to describe. This section provides information about the principles of error handling, and the options that are available, and you must decide what combination of choices that you need in each situation based on the details that are provided in this section.

You can choose one or more of these options in your message flows:

- Connect the failure terminal of any node to a sequence of nodes that processes the node’s internal exception (the fail flow).
- Connect the catch terminal of the input node to a sequence of nodes that processes exceptions that are generated beyond it (the catch flow).
- Ensure that all messages received by an MQInput node are processed within a transaction, or are not.
- Ensure that all messages received by an MQInput node are persistent, or are not.

If you include user-defined nodes in your message flow, you must refer to the information provided with the node to understand how you might handle errors with these nodes. The descriptions in this section cover only the built-in nodes.

When you design your error handling approach, consider the following factors:

- Most of the built-in nodes have failure terminals. The exceptions are Input, Output, Passthrough, Publication, Real-timeInput, and Real-timeOptimizedFlow.

  When an exception is detected within a node, the message and the exception information are propagated to the node's failure terminal. If the node does not have a failure terminal, or it is not connected, the broker throws an exception and returns control to the input node.

  If an MQinput node detects an internal error, its behavior is slightly different; if the failure terminal is not connected, it attempts to put the message to the input queue's backout requeue queue, or (if that is not defined) to the dead letter queue of the broker's queue manager,

  For more information, see the following:
  - "Handling MQInput errors" on page 38
- The MQInput, and SCADAInput nodes have catch terminals.

  A message is propagated to a catch terminal only if it has first been propagated beyond the node (for example, to the nodes connected to the out terminal).
- When a message is propagated to the failure or catch terminal, the node creates and populates a new ExceptionList with an exception that represents the error that has occurred. The ExceptionList is propagated as part of the message tree.
- The MQInput node has additional processing for transactional messages (other input nodes do not handle transactional messages).
- If you include a Trace node that specifies `$Root` or `$Body`, the complete message is parsed. This might generate parser errors that are not otherwise detected.

The general principles of error handling are:

- If you connect the catch terminal of the input node, you are indicating that the flow handles all exceptions that are generated anywhere in the out flow. The broker performs no rollback and takes no action unless there is an exception on the catch flow. If you want any rollback action after an exception has been raised and caught, you must provide this in the catch flow.
- If you do not connect the catch terminal of the MQInput node, you can connect the failure terminal and provide a fail flow to handle exceptions generated by the node. The fail flow is invoked immediately when an exception occurs in the node.

  The fail flow is also invoked if an exception is generated beyond the MQInput node (in either out or catch flows), the message is transactional, and the reinstatement of the message on the input queue causes the backout count to reach the backout threshold.

  The SCADAInput node does not propagate the message to the failure terminal if an exception is generated beyond the node and you have not connected its catch terminal.
- If a node propagates a message to a catch flow, and another exception occurs that returns control to the same node again, the node handles the message as though the catch terminal is not connected.
- If you do not connect either failure or catch terminals of the input node, the broker provides default processing (which varies with the type of input node).
- If you have a common procedure for handling particular errors, you might find it appropriate to create a subflow that includes the sequence of nodes required. Include this subflow wherever you need that action to be taken.

The Error Handler sample demonstrates how to use an error handling routine to trap information about errors and to store that information in a database. The error handling routine is a subflow that you can add, unchanged, to any message flow. The sample also demonstrates how to configure message flows to control transactionality; in particular, the use of globally coordinated transactions to ensure overall data integrity.

Connecting failure terminals

When a node that has a failure terminal detects an internal error, it propagates the message to that terminal. If it does not have a failure terminal, or if you have not connected the failure terminal, the broker generates an exception.

The nodes sometimes generate errors that you can predict, and it is in these cases that you might want to consider connecting the failure terminal to a sequence of nodes that can take sensible actions in response to the expected errors.

Examples of expected errors are:
- Temporary errors when the input node retrieves the message.
- Messages with an internal or format error that cannot be recognized or processed by the input node.

You can also connect the failure terminal if you do not want WebSphere MQ to retry a message or put it to a backout or dead letter queue.

Managing errors in the input node

When you design your message flow, consider which terminals on the input node to connect:
- If the node detects an error, it always propagates the message to the failure terminal if the node has one and if you have connected a fail flow.
- If you connect the catch terminal (if the node has one), this indicates that you want to handle all exceptions that are generated in the out flow. This handles errors that can be expected in the out flow. The broker does not take any action unless there is an exception on the catch flow and the message is transactional. Connect the failure terminal to handle this case if you choose.
- If you do not connect the catch terminal, or the node does not have a catch terminal, the broker provides default processing. This depends on the node and whether the message is transactional. Processing for non-transactional messages is described in this topic. Refer to “Handling MQInput errors” on page 38 for details of how these nodes handle transactional messages (other input nodes do not support transactional messages).

All input nodes process non-transactional, non-persistent messages. The built-in input nodes handle failures and exceptions associated with these messages in this way:
- If the node detects an internal error:
  - If you have not connected the failure terminal, the node logs the error in the local error log and discards the message.

    The Real-timeInput and Real-timeOptimizedFlow nodes retry once before they discard the message; that is, they retrieve the message again and attempt to process it.
  - If you have connected the failure terminal, you are responsible for handling the error in the fail flow. The broker creates a new ExceptionList to represent

the error and this is propagated to the failure terminal as part of the message tree, but neither the node nor the broker provide any further failure processing.

- If the node has successfully propagated the message to the out terminal and a later exception results in the message being returned to the input node:
  - If you have not connected the catch terminal or the node does not have a catch terminal, the node logs the error in the local error log and discards the message.
  - If you have connected the catch terminal, you are responsible for handling the error in the catch flow. The broker creates a new ExceptionList to represent the error and this is propagated to the catch terminal as part of the message tree, but neither the node nor the broker provide any further exception processing.
- If the node has already propagated the message to the catch terminal and an exception is thrown in the catch flow:
  - If you have not connected the failure terminal, or the input node does not have a failure terminal, the node logs the error in the local error log and discards the message.
  - If you have connected the failure terminal, you are responsible for handling the error in the fail flow. The broker creates a new ExceptionList to represent the error and this is propagated to the failure terminal as part of the message tree, but neither the node nor the broker provide any further failure processing.

    The SCADAInput node does not propagate the message to the failure terminal if an exception is generated in the catch flow. The node logs the error in the local error log and discards the message.
- If the node has propagated the message to the failure terminal and an exception is thrown in the fail flow, the node logs the error in the local error log and discards the message.

This action is summarized in the table below:

Error event	Failure terminal connected	Failure terminal not connected	Catch terminal connected	Catch terminal not connected
Node detects internal error	Fail flow handles the error	Node logs the error and discards the message	Not applicable	Not applicable
Node propagates message to out terminal, exception occurs in out flow	Not applicable	Not applicable	Catch flow handles the error	Node logs the error and discards the message
Node propagates message to catch terminal, exception occurs in catch flow	Fail flow handles the error (not SCADAInput)	Node logs the error and discards the message	Not applicable	Not applicable
Node propagates message to failure terminal, exception occurs in fail flow	Not applicable	Not applicable	Node logs the error and discards the message	Node logs the error and discards the message

Handling MQInput errors:

The MQInput node takes the following actions when it handles errors with persistent and transactional messages. Errors encountered with non-transactional messages are handled as described in "Managing errors in the input node" on page 36.

- The MQInput node detects an internal error in the following situations:
  - A message validation error occurs when the associated message parser is initialized.
  - A warning is received on an MQGET.
  - The backout threshold is reached when the message is rolled back to the input queue.
- If the MQInput node detects an internal error, one of the following actions occur:
  - If you have not connected the Failure terminal, the MQInput node attempts to put the message to the input queue's backout requeue queue, or (if that is not defined) to the dead letter queue of the broker's queue manager. If the put attempt fails, the message is rolled back to the input queue. The MQInput node writes the original error and the MQPUT error to the local error log. The MQInput node now invokes the retry logic, described in "Retry processing" on page 39.
  - If you have connected the Failure terminal, you are responsible for handling the error in the flow connected to the Failure terminal. The broker creates a new ExceptionList to represent the error and this is propagated to the Failure terminal as part of the message tree, but neither the MQInput node nor the broker provide any further failure processing.
- If the MQInput node has successfully propagated the message to the out terminal and an exception is thrown in the out flow, the message is returned to the MQInput node:
  - If you have not connected the Catch terminal, the message is rolled back to the input queue. The MQInput node writes the error to the local error log and invokes the retry logic, described in "Retry processing" on page 39.
  - If you have connected the Catch terminal, you are responsible for handling the error in the flow connected to the Catch terminal. The broker creates a new ExceptionList to represent the error and this is propagated to the Catch terminal as part of the message tree, but neither the MQInput node nor the broker provide any further failure processing.
- If the MQInput node has already propagated the message to the Catch terminal and an exception is thrown in the flow connected to the Catch terminal, the message is returned to the MQInput node:
  - The MQInput node writes the error to the local error log.
  - The message is rolled back to the input queue.
- If the MQInput node has already propagated the message to the Failure terminal and an exception is thrown in the flow connected to the Failure terminal, the message is returned to the MQInput node and rolled back to the input queue. The MQInput node writes the error to the local error log and invokes the retry logic, described in "Retry processing" on page 39. The message is not propagated to the Catch terminal, even if that is connected.

This action is summarized in the table below:

Error event	**Failure terminal connected**	**Failure terminal not connected**	**Catch terminal connected**	**Catch terminal not connected**
Node detects internal error	Flow connected to the Failure terminal handles the error	Message put to alternative queue; node retries if the put fails	Not applicable	Not applicable
Node propagates message to out terminal, exception occurs in out flow	Not applicable	Not applicable	Flow connected to the Catch terminal handles the error	Node retries
Node propagates message to Catch terminal, exception occurs in flow connected to the Catch terminal	Error logged, message rolled back	Error logged, message rolled back	Not applicable	Not applicable
Node propagates message to Failure terminal, exception occurs in flow connected to the Failure terminal	Not applicable	Not applicable	Node retries	Node retries

Retry processing:

The node attempts retry processing when a message is rolled back to the input queue. It checks whether the message has been backed out before, and if it has, whether the backout count has reached (equalled) the backout threshold. The backout count for each message is maintained by WebSphere MQ in the MQMD.

You specify (or allow to default to 0) the backout threshold attribute *BOTHRESH* when you create the queue. If you accept the default value of 0, the node increases this to 1. The node also sets the value to 1 if it cannot detect the current value. This means that if a message has not been backed out before, it is backed out and retried at least once.

1. If the node has propagated a message to the out terminal many times following repeated failed attempts in the out flow, and the number of retries has reached the backout threshold limit, it attempts to propagate the message through the Failure terminal if that is connected. If you have not connected the Failure terminal, the node attempts to put the message to another queue.

   If a failure occurs beyond the Failure terminal, further retries are made until the backout count field in the MQMD reaches twice the backout threshold set for the input queue. When this limit is reached, the node attempts to put the message to another queue.
2. If the backout threshold has not been reached, the node gets the message from the queue again. If this fails, this is handled as an internal error (described above). If it succeeds, the node propagates the message to the out flow.
3. If the backout threshold has been reached:

- If you have connected the Failure terminal, node propagates the message to that terminal. You must handle the error on the flow connected to the Failure terminal.
- If you have not connected the Failure terminal, the node attempts to put the message on an available queue, in order of preference:
  a. The message is put on the input queue's backout requeue name (queue attribute *BOQNAME*), if one is defined.
  b. If the backout queue is not defined, or it cannot be identified by the node, the message is put on the dead letter queue (DLQ), if one is defined. (If the broker's queue manager has been defined by the **mqsicreatebroker** command, a DLQ with a default name of SYSTEM.DEAD.LETTER.QUEUE has been defined and is enabled for this queue manager.)
  c. If the message cannot be put on either of these queues because there is an MQPUT error (including queue does not exist), or because they cannot be identified by the node, it cannot be handled safely without risk of loss.

     The message cannot be discarded, therefore the message flow continues to attempt to backout the message. It records the error situation by writing errors to the local error log. A second indication of this error is the continual incrementing of the *BackoutCount* of the message in the input queue.

     If this situation has occurred because neither queue exists, you can define one of the backout queues mentioned above. If the condition preventing the message from being processed has cleared, you can temporarily increase the value of the *BOTHRESH* attribute. This forces the message through normal processing.

4. If twice the backout threshold has been reached or exceeded, the node attempts to put the message on an available queue, in order of preference, as defined in the previous step.

Handling message group errors:

WebSphere MQ supports message groups. You can specify that a message belongs to a group and its processing is then completed with reference to the other messages in the group (that is, either all messages are committed or all messages are rolled back). When you send grouped messages to a broker, this condition is upheld if you have configured the message flow correctly, and errors do not occur during group message processing.

To configure the message flow to handle grouped messages correctly, follow the actions described in the "MQInput node" on page 154. However, correct processing of the message group cannot be guaranteed if an error occurs while one of the messages is being processed.

If you have configured the MQInput node as described, under normal circumstances all messages in the group are processed in a single unit of work which is committed when the last message in the group has been successfully processed. However, if an error occurs before the last message in the group is processed, the unit of work that includes the messages up to and including the message that generates the error is subject to the error handling defined by the rules documented here, which might result in the unit of work being backed out.

However, any of the remaining messages within the group might be successfully read and processed by the message flow, and therefore are handled and committed in a new unit of work. A commit is issued when the last message is encountered and processed. Therefore if an error occurs within a group, but not on the first or last message, it is possible that part of the group is backed out and another part committed.

If your message processing requirements demand that this situation is handled in a particular way, you must provide additional error handling to handle errors within message groups.

Managing message flows

This section contains information on managing message flows:
- "Creating a message flow project"
- "Deleting a message flow project" on page 42
- "Creating a broker schema" on page 43
- "Creating a message flow" on page 44
- "Opening an existing message flow" on page 45
- "Copying a message flow using copy" on page 46
- "Renaming a message flow" on page 46
- "Moving a message flow" on page 47
- "Deleting a message flow" on page 48
- Displaying version and keyword information
- "Saving a message flow" on page 50

To learn more about message flows, try importing the Airline Reservations sample (or another sample from the Samples Gallery) and explore the samples message flow resources; try creating, deleting, or renaming the resources.

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

Creating a message flow project

Before you start:

Read the concept topic about message flow projects.

A message flow project is a container for message flows; you must create a project before you can create a message flow.

The project and its resources are stored in a file system or in a shared repository. If you are using a file system, this can be the local file system or a shared drive. If you store files in a repository, you can use any of the available repositories that are supported by Eclipse, for example CVS.

To create a message flow project:
1. Switch to the Broker Application Development perspective.
2. Click **File** → **New** → **Message Flow Project** or right-click any resource in the Navigator view and click **New** → **Message Flow Project**.

   You can also press Ctrl+N. This displays a dialog that allows you to select the wizard to create a new object. Click Message Brokers in the left view; the right view displays a list of objects that you can create for WebSphere Event Broker. Click Message Flow Project in the right view, then click **Next**. The New Message Flow Project wizard displays.

3. Enter a name for the project. Choose a project name that reflects the message flows that it contains. For example, if you want to use this project for financial processing message flows, you might give it the name Finance_Flows.
4. Leave the *Use default* check box checked (it is checked when the dialog opens) This applies if you want to use the default location for the new message project directory, that is, in the \workspace subdirectory of your current installation. You cannot edit the *Directory* entry field.
   a. Alternatively, clear the *Use default* check box and specify a location for the new message flow project files in the *Directory* entry field. This applies if you do not want to use the default location.
   b. Use the **Browse** button to find the desired location or type the location in.
5. Click **Next** if you want to specify that this message flow project depends on other message flow projects, or on message set projects, You are presented with a list of current projects. Select one or more message flow projects from the list to indicate this new message flow project's dependencies.

   This message flow project has a dependency on another message flow project if you intend to use subflows within it that are defined in another project.

   You can add dependencies after you have created the message flow project by right-clicking the project in the Resource Navigator and clicking **Properties**. Click **References** and select the dependent message flow or message set project from the list of projects displayed.
6. Click **Finish** to complete the task.

The project file is created within a directory that has the same name as your message flow project in the specified location. All other files that you create (or cause to be created) related to this message flow project are created in this same directory.

A default broker schema (`default`) is also created within the project. You can create and use different schemas within a single project to organize message flow resources, and to provide the scope of resource names to ensure uniqueness.

Next: create a message flow

Deleting a message flow project

A message flow project is the container in which you create and maintain all the resources associated with one or more message flows. These resources are created as files, and are displayed within the project in the Resource Navigator view. If you do not want to retain a message flow project, you can delete it.

Before you start:
- Create a message flow project
- Read the concept topic about message flow projects

Deleting a message flow project in the workbench deletes the project and its resources; the Configuration Manager does not hold a copy. If you are using a shared repository, the repository might retain a copy of a deleted resource.

In previous releases you could remove resources from the Control Center, which removed the reference in your workspace, but retained the resource in the Configuration Manager repository.

To delete a message flow project:

1. Switch to the Broker Application Development perspective.
2. Highlight the message flow project that you want to delete and click **Edit** → **Delete** You can also press Del, or right-click the project in the Navigator view and click **Delete**
3. You must choose if you want the contents of the message flow project folder deleted with this action on the displayed confirmation dialog. The dialog contains two buttons:
   a. The first confirms that all contents are to be deleted.
   b. The second requests that the directory contents are not deleted. The default action is not to delete the contents, and the second button is selected by default when the dialog is initially displayed.
   a. Select the appropriate button. If you choose not to delete the contents of the message flow project directory, all the files and the directory itself are retained.

      If you later create another project with the same name, and specify the same location for the project (or accept this as the default value), you can access the files previously created.

      If you choose to delete all the contents, all files and the directory itself are deleted.
4. Click **Yes** to complete the delete request, or **No** to terminate the delete request.

When you click **Yes**, the requested objects are deleted.

If you maintain resources in a shared repository, a copy is retained in that repository. You can follow the instructions provided by the repository supplier to retrieve the resource if required.

If you are using the local drive or a shared drive to store your resources, no copy of the resource is retained. Be very careful to select the correct resource when you complete this task.

Creating a broker schema

If you want to organize your message flow project resources, and to define the scope of resource names to ensure uniqueness, you can create broker schemas. A default schema is created when you create the message flow project, but you can create additional schemas if you choose.

Before you start:

- Create a message flow project
- Read the concept topic about broker schemas

To create a broker schema:

1. Switch to the Broker Application Development perspective.
2. Click **File** → **New** → **BrokerSchema** or right-click any resource in the Navigator view and click **New** → **BrokerSchema**.

You can also press Ctrl+N. This displays a dialog that allows you to select the wizard to create a new object. Click Message Brokers in the left view. The right view displays a list of objects that you can create for WebSphere Event Broker. Click Broker Schema in the right view, then click **Next**. The New Broker Schema wizard displays.

3. Enter the message flow project in which you want the new schema to be created. If you have a message flow project or one of its resources highlighted when you invoke the wizard, that project name appears in the dialog. If a name does not appear in this field, or if you want to create the schema in another project, click **Browse** and select the correct project from the displayed list.

   You can type the project name in, but you must enter a valid name. The dialog displays a red cross and the error message `The specified project does not exist` if your entry is not a valid project.
4. Enter a name for the schema. Choose a name that reflects the resources that it contains. For example, if you want to use this schema for message flows for retail applications, you might give it the name Retail.

   A broker schema name must be a character string that starts with a Unicode character followed by zero or more Unicode characters or digits, and the underscore. You can use the period to provide a structure to the name, for example `Stock.Common`.
5. Click **Finish** to complete the task.

The schema directory is created in the project directory. If the schema is structured using periods, further subdirectories are defined. For example, the broker schema `Stock.Common` results in a directory `Common` within a directory `Stock` within the message flow project directory.

Creating a message flow

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

Before you start:
- Create a message flow project
- Read the concept topic about broker schemas

The message flow and its resources are stored in a file system or in a shared repository. If you are using a file system, this can be the local drive or a shared drive. If you store files in a repository, you can use any of the available repositories that are supported by Eclipse, for example CVS.

Use this process to create a complete message flow that you can deploy, or a subflow that provides a subset of function (for example, a reusable error processing routine) that you cannot deploy on its own.

To create a message flow:
1. Switch to the Broker Application Development perspective.
2. Check that you have already created the message flow project in which you want to create the message flow. You can only create a message flow in an existing project. The project can be empty, or can already have message flows defined in it.
3. Click **File** → **New** → **Message Flow** or right-click any resource in the Navigator view and click **New** → **Message Flow**.

You can also press Ctrl+N. This displays a dialog that allows you to select the wizard to create a new object. Click Message Brokers in the left view. The right view displays a list of objects that you can create for WebSphere Event Broker. Click Message Flow in the right view, then click **Next**. The New Message Flow wizard displays.

4. Identify the project in which you want to define the message flow. If you have a resource selected in the Navigator view, the name of the corresponding project is displayed in the first entry field, **Project**.

   If you do not have a resource selected, the first field is blank. Click **Browse** to select the appropriate project for this message flow. A dialog containing a list of valid projects is displayed. Select the correct project and click **OK**.

   You can type the project name in, but you must enter a valid name. The dialog displays a red cross and the error message `The specified project does not exist` if your entry is not a valid project.
5. Complete the **Schema** and **Name** fields when the project is correct:
   a. In **Schema**, enter the identifier of the broker schema in which the message flow is defined. When you create a message flow project, a default schema is created within it, and this default value is always assumed if you do not enter a value in this field, or do not select a value using the **Browse** button.

      You can create and use different schemas within a single project to organize message flow resources, and to provide the scope of resource names to ensure uniqueness.
   b. In **Name**, enter the name of the message flow. You can use any valid character for the name; choose a name that reflects its function, for example OrderProcessing.
6. Click **Finish**.

The new message flow (<message_flow_name>.msgflow) is displayed within its project in the Navigator view. The editor view is empty and ready to receive your input.

Next: save the message flow or define message flow content.

Opening an existing message flow

Open an existing message flow to change or update its contents, or to add or remove nodes.

Before you start

To complete this task, you must have completed the following tasks:
- "Creating a message flow" on page 44

To open an existing message flow:
1. Switch to the Broker Application Development perspective. The Navigator view is populated with all the message flow and message set projects that you have access to. A message flow is contained in a file called <message_flow_name>.msgflow.
2. Right-click the message flow that you want to work with, and click **Open**. Alternatively you can double-click the message flow in the Navigator view.

   The graphical view of the message flow is displayed in the editor view. You can now work with this message flow, for example, adding or removing nodes, changing connections, or modifying properties.

Copying a message flow using copy

You might find it useful to copy a message flow as a starting point for a new message flow that has similar function. For example, you might want to replace or remove one or two nodes to process messages in a different way.

Before you start

To complete this task, you must have completed the following task:
- "Creating a message flow" on page 44

To copy a message flow:
1. Switch to the Broker Application Development perspective.
2. Select the message flow (<message_flow_name>.msgflow) that you want to copy in the Navigator view.
   a. Right-click the file and click **Copy** from the menu.
3. Right-click the broker schema within the message flow project to which you want to copy the message flow and click **Paste**. You can copy the message flow within the same broker schema within the same message flow, or to a different broker schema within the same message flow project, or to a broker schema in a different message flow project.

The message flow is copied with all property settings intact. If you intend to use this copy of the message flow for another purpose, for example to retrieve messages from a different input queue, you might have to modify its properties.

You can also use **File** → **Save As** to copy a message flow. This is described in "Saving a message flow" on page 50.

Renaming a message flow

You can rename a message flow. You might want to do this if you have modified the message flow to provide a different function and you want the name of the message flow to reflect this new function.

Before you start

To complete this task, you must have completed the following task:
- "Creating a message flow" on page 44

To rename a message flow:
1. Switch to the Broker Application Development perspective.
2. Right-click the message flow that you want to rename (<message_flow_name>.msgflow) in the Navigator view, and click **Rename**, or click **File** → **Rename**. If you have the message flow selected, you can also press F2. The Rename Resource dialog is displayed.
3. Type in the new name for the message flow.
4. Click **OK** to complete this action, or **Cancel** to cancel the request. If you click **OK**, the message flow is renamed.

   After you have renamed the message flow, any references that you have to this message flow (for example, if it is embedded in another message flow) are no longer valid.

5. You must open the affected message flows and correct the references if you are not sure where you have embedded this message flow.
   a. Click **File** → **Save All** The save action saves and validates all resources. Unresolved references are displayed in the Tasks view, and you can click each error listed.

      This opens the message flow that makes a non-valid reference in the editor view
   b. Right click the subflow icon and click **Locate Subflow**. The Locate Subflow dialog is displayed, listing the available message flow projects.
   c. Expand the list and explore the resources available to locate the required subflow.
   d. Select the correct subflow and click **OK**. All references in the current message flow are updated for you and the errors removed from the Tasks view.

Moving a message flow

You can move a message flow from one broker schema to another within the same project or to a broker schema in another project. You might want to do this, for example, if you are reorganizing the resources in your projects.

Before you start

To complete this task, you must have completed the following task:
- “Creating a message flow” on page 44

To move a message flow:
1. Switch to the Broker Application Development perspective.
2. Drag and drop the message flow that you want to move from its current location to a broker schema within the same or another message flow project. If the target location that you have chosen is not valid, a black no-entry icon appears over the target, an error dialog is displayed, and the message flow is not moved.

   You can move a message flow to another schema in the same project or to a schema in another message flow project.
3. If you prefer, you can:
   a. Right-click the message flow that you want to move (<message_flow_name>.msgflow) in the Navigator view and click **Move**, or **File** → **Move** The Move dialog is displayed. This contains a list of all valid projects to which you can move this message flow.
   b. Select the project and the broker schema within the project to which you want to move the message flow. You can move a message flow to another schema in the same project or to a schema in another message flow project.
   c. Click **OK** to complete the move, or **Cancel** to cancel the move. If you click **OK**, the message flow is moved to its new location.
4. Check the Tasks view for any errors (indicated by the error icon ⊗) or warnings (indicated by the warning icon ⚠) generated by the move. The errors in the Tasks view include those caused by broker references. When the move is done, all references to this message flow (for example, if this is a reusable error message flow that you have embedded in another message flow) are checked.

If you have moved the message flow within the same broker schema (in the same or another project), all references are still valid.

However, if you move the message flow from one broker schema to another (in the same or a different project), the references are broken.

This is because the resources are linked by a fully-qualified name of which the broker schema is a part. Information about any broken references is written to the Tasks view, for example, `Linked or nested flow mflow1 cannot be located`.

5. Double-click each error or warning to correct it. This opens the message flow that has the error in the editor view and highlights the node in error.

Deleting a message flow

You can delete a message flow that you have created in a message flow project if you no longer need it.

Deleting a message flow in the workbench deletes the project and its resources, and the Configuration Manager does not hold a copy. If you are using a shared repository, the repository might retain a copy of a deleted resource.

In previous releases you could remove resources from the Control Center, which removed the reference in your workspace, but retained the resource in the Configuration Manager repository.

Before you start

To complete this task, you must have completed the following task:
- “Creating a message flow” on page 44

To delete a message flow:
1. Switch to the Broker Application Development perspective.
2. Select the message flow in the Navigator view (<message_flow_name>.msgflow) and press the Delete key. A confirmation dialog is displayed.

   You can also right-click the message flow in the Navigator view and click **Delete**, or click **Edit** → **Delete**. The same dialog is displayed.
3. Click **Yes** to delete the message flow definition file or **No** to cancel the delete request. When you click **Yes**, the requested objects are deleted.

   If you maintain resources in a shared repository, a copy is retained in that repository. You can follow the instructions provided by the repository supplier to retrieve the resource if required.

   If you are using the local file system or a shared file system to store your resources, no copy of the resource is retained. Be very careful to select the correct resource when you complete this task.
4. Check the Tasks view for any errors that are caused by the deletion. Errors are generated if you delete a message flow that is embedded within another flow because the reference is no longer valid.
   a. Click the error in the Tasks view This opens the message flow that now has a non-valid reference.
   b. Either remove the node that represents the deleted message flow from the parent message flow, or create a new message flow with the same name to provide whatever processing is required.

Deleting a broker schema

You can delete a broker schema that you have created in a message flow project if you no longer need it.

Before you start:

- Create a broker schema
- Read the concept topic about broker schemas

To delete a broker schema:

1. Switch to the Broker Application Development perspective.
2. Select the broker schema in the Resource Navigator view and press the Delete key. A confirmation dialog box is displayed.

   You can also right-click the broker schema in the Resource Navigator view and click **Delete**, or click **Edit** → **Delete**. The same dialog box is displayed.

   If the broker schema contains resources, the **Delete** menu option is disabled, and the Delete key has no effect. You must delete all resources within the schema before you can delete the schema.
3. Click **Yes** to delete the broker schema directory or **No** to cancel the delete request. When you click **Yes**, the requested objects are deleted.

   If you maintain resources in a shared repository, a copy is retained in that repository. You can follow the instructions provided by the repository supplier to retrieve the resource, if required.

   If you are using the local file system or a shared file system to store your resources, no copy of the resource is retained. Be very careful to select the correct resource when you complete this task.

Viewing version and keyword information for deployable objects

This topic contains information about how to view the version and keyword information of deployable objects.

- "Displaying object version in the bar file editor"
- "Displaying version, deploy time, and keywords of deployed objects"

Displaying object version in the bar file editor

A column in the bar editor called *Version* displays the version tag for all objects that have a defined version. These are:

- `.cmf` files

You cannot edit the *Version* column.

You can use the mqsireadbar command to list the keywords defined for each deployable file within a deployable archive file.

Displaying version, deploy time, and keywords of deployed objects

The Eclipse *Properties View* displays, for any deployed object:

- Version
- Deploy Time
- All defined keywords

For example, if you deploy a message flow with these literal strings:
- `$MQSI_VERSION=v1.0 MQSI$`
- `$MQSI Author=fred MQSI$`
- `$MQSI Subflow 1 Version=v1.3.2 MQSI$`

then the Properties View displays:

Deployment Time	Date and time of deployment
Modification Time	Date and time of modification
Version	`v1.0`
Author	`fred`
Subflow 1 Version	`v1.3.2`

You are given a reason if the keyword information is not available. For example, if keyword resolution has not been enabled at deploy time, the Properties View displays the message `Deployed with keyword search disabled`. Also, if you deploy to a Configuration Manager that is an earlier version than Version 6.0, the properties view displays `Keywords not available on this Configuration Manager`.

Saving a message flow

You might want to save your message flow when you want to:
- Close the workbench.
- Work with another resource.
- Validate the contents of the message flow.

Before you start:

To complete this task, you must have completed the following task:
- "Creating a message flow" on page 44

To save a message flow:
1. Switch to the Broker Application Development perspective.
2. Select the editor view that contains the open message flow that you want to save.
3. If you want to save the message flow without closing it in the editor view, press Ctrl+S or click **File** → **Save name** on the taskbar menu (where *name* is the name of this message flow). You can also choose to save everything by clicking **File** → **Save All**.

   The message flow is saved and the message flow validator is invoked to validate its contents. The validator provides a report of any errors that it finds in the Tasks view. The message flow remains open in the editor view.

   For example, if you save a message flow and have not set a mandatory property, an error message appears in the Tasks view and the editor marks the node with the error icon ⊗ . The message flow in the Navigator view is also marked with the error icon. This can occur if you have not edited the properties of an MQInput node to define the queue from which the input node retrieves its input messages.

(If you edit the properties of a node, you cannot click **OK** unless you have set all mandatory properties. Therefore this situation can arise only if you have never set any properties.)

You might also get warnings when you save a message flow. These are indicated by the warning icon ⚠. This informs you that, although there is not an explicit error in the configuration of the message flow, there is a situation that might result in unexpected results when the message flow completes. For example, if you have included an input node in your message flow that you have not connected to any other node, you get a warning. In this situation, the editor marks the node with the warning icon. The message flow in the Navigator view is also marked with a warning icon.

4. If you save a message flow that includes a subflow, and the subflow is no longer available, three error messages are added to the Tasks view that indicate that the input and output terminals and the subflow itself cannot be located. This can occur if the subflow has been moved or renamed.

   To resolve this situation, right-click the subflow node in error and click **Locate Subflow**. The Locate Subflow dialog is displayed, listing the available message flow projects. Expand the list and explore the resources available to locate the required subflow. Select the correct subflow and click **OK**. All references in the current message flow are updated for you and the errors removed from the Tasks view.
5. If you want to save the message flow when you close it, click the close view icon ✕ on the editor view tab for this message flow or click **File** → **Close** on the taskbar menu. The editor view is closed and the file saved. The same validation occurs and any errors and warnings are written to the Tasks view.

For information about using the **File** → **Save As** option to take a copy of the current message flow, see "Copying a message flow using save."

See "Correcting errors from saving a message flow" on page 52 for information about handling errors from the save action.

Copying a message flow using save

You can copy a message flow by using the **File** → **Save As** option.

1. Click **File** → **Save name As**.
2. Specify the message flow project in which you want to save a copy of the message flow. The project name defaults to the current project. You can accept this name, or choose another name from the valid options that are displayed in the File Save dialog.
3. Specify the name for the new copy of the message flow. If you want to save this message flow in the same project, you must either give it another name, or confirm that you want to overwrite the current copy (that is, copy the flow to itself).

   If you want to save this message flow in another project, the project must already exist (you can only select from the list of existing projects). You can save the flow with the same or another name in another project.
4. Click **OK**. The message flow is saved and the message flow editor validates its contents. The editor provides a report of any errors that it finds in the Tasks view. See "Correcting errors from saving a message flow" on page 52 for information about handling errors from the save action.

Correcting errors from saving a message flow

Correct the errors that are reported when you save a message flow.

To correct errors from the save or save as action:

1. Examine the list of errors and warnings that the validator has generated in the Tasks view.
2. Double-click each entry in turn. The message flow is displayed in the editor view (if it is not already there), and the editor selects the node in which the error was detected. If the error has been generated because you have not set a mandatory property, the editor also opens the Properties view or dialog box for that node.

   If you have included a user-defined node in your message flow, and have one or more of its properties have been defined as configurable, you might get a warning about a custom property editor. If a property has been defined as configurable, and you have specified that it uses a custom property editor, the bar editor cannot handle the custom property editor; the bar editor handles the property as if it is type String, which restricts your ability to make changes to this property at deploy time.
3. Correct the error that is indicated by the message. For example, provide a value for the mandatory property.
4. When you have corrected all the errors, you can save again. The editor validates all the resources that you have changed, removes any corrected errors from the Tasks view, and removes the corresponding graphical indication from the nodes that you have modified successfully.

You do not have to correct every error to save your work. The editor saves your resources even if it detects errors or warnings, so that you can continue to work with them at a later date. However, you cannot deploy any resource that has a validation error. You must correct every error before you deploy a resource. Warnings do not prevent successful deployment.

Defining message flow content

When you create a new message flow, the editor view is initially empty. You must create the contents of the message flow by:

- “Adding a message flow node” on page 55
- “Adding a subflow” on page 56
- “Renaming a message flow node” on page 57
- “Configuring a message flow node” on page 58
- “Connecting message flow nodes” on page 59
- “Adding a bend point” on page 62
- “Aligning and arranging nodes” on page 64

When you finalize the content of the message flow, you might also need to perform the following tasks:

- “Removing a message flow node” on page 59
- “Removing a node connection” on page 62
- “Removing a bend point” on page 63

To learn more about message flow content, try importing the Airline Reservations sample or the Error Handler sample, and follow the supplied instructions to build the sample yourself. Also, try adding and deleting nodes, adding subflows, and connecting nodes together.

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

Using the node palette

Before you start:

Read the concept topic about the node palette.

The node palette contains all of the built-in nodes, which are organized into categories. You can add the nodes that you use most often to the Favorites category by following the instructions in "Adding nodes to the Favorites category on the palette" on page 54.

You can change the palette preferences in the Message Brokers Toolkit. The changes that you can make are described in the following topics.

- "Changing the palette layout"
- "Changing the palette settings"
- "Customizing the palette" on page 54

Changing the palette layout

You can change the layout of the palette in the Message Flow editor and the Broker Topology editor.

1. Switch to the Broker Application Development perspective
2. Right-click the palette to display the pop-up menu.
3. Click **Layout**.
4. Click one of the available views:

   Columns
   Displays named icons in one or more columns. Change the number of columns by clicking on the right edge of the palette and dragging.

   List Displays named icons in a single-column list. The list view is the default layout.

   Icons Only
   Displays a list of icons only.

   Details
   Displays descriptions of the icons.

Changing the palette settings

Change the palette settings in the Message Flow editor and the Broker Topology editor using the **Palette Settings** dialog box.

1. Switch to the Broker Application Development perspective.
2. Right-click the palette to display the pop-up menu.
3. Click **Settings**. The **Palette Settings** dialog box opens.
4. Use the dialog to change the appropriate setting:

- Click **Change** to change the font on the palette.
- Click **Restore Default** to restore the default palette settings.
- In the **Layout** list, click the appropriate radio button to change the palette layout. (See “Changing the palette layout” on page 53 for more information.)
- Select **User large icons** to toggle between large and small icons in the palette.
- In the **Drawer options** list, click the appropriate radio button to change the way that drawers are handled in the palette. A drawer is a container for a list of icons, such as the **Favorites** drawer on the Message Flow editor’s palette, or the **Entity** drawer on the Broker Topology editor’s palette.

Customizing the palette

Customize the palette for the Message Flow editor using the **Customize Palette** dialog box.

1. Switch to the Broker Application Development perspective.
2. Right-click the palette to display the pop-up menu.
3. Click **Customize**. The **Customize Palette** dialog box opens.
4. To change the order of entries and drawers in the palette, click the appropriate item in the list to highlight it, then click **Move Down** or **Move Up**. You cannot move any category above the Favorites category.
5. To hide an entry or drawer, click the appropriate item in the list to highlight it, then select **Hide**.
6. To create a new separator, click **New** → **Separator**.
7. To create a new drawer:
   a. Click **New** → **Drawer**.
   b. Type a name and description for the drawer.
   c. If required, select **Open drawer at start-up**.
   d. If required, select **Pin drawer open at start-up**.

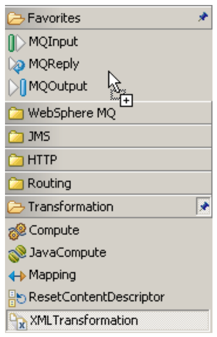
Adding nodes to the Favorites category on the palette

Before you start:

Read the concept topic about the message flow node palette.

The nodes on the palette are organized in categories. The first category is Favorites, which is usually empty. You can drag the nodes that you use most often to the Favorites category.

1. Switch to the Broker Application Development perspective.
2. On the palette, open the Favorites category.
3. On the palette, open the category that contains the node that you want to add to the Favorites category.
4. Use the mouse to drag the node into the Favorites category, as shown in the following example:



Alternatively, right-click the palette and choose the appropriate option to add or remove nodes from the Favorites category.

Adding a message flow node

When you have created a new message flow, add nodes to define its function.

Before you start:
- Create a message flow or open an existing message flow
- Read the concept topic about message flow nodes

To add a node to a message flow:
1. Switch to the Broker Application Development perspective.
2. Open the message flow with which you want to work.
3. Click **Selection** above the palette of nodes.
4. Decide which node you want to add, a built-in node or a user-defined node. You can select any of the nodes that appear in the node palette, but you can add only one node at a time.

   Nodes are grouped in categories according to the function that they provide. To see descriptions of the nodes in the palette, either hover the mouse over a node in the palette, or switch to the Details view by following the instructions in "Changing the palette layout" on page 53.
5. Drag the node from the node palette onto the canvas.

   When you add a node to the canvas, the editor automatically assigns a name to the node, but the name is highlighted and you can change it by entering a name of your choice. If you do not change the default name at this time, you can change it later. The default name is set to the type of node for the first instance. For example, if you add an MQInput node to the canvas, it is given the name MQInput; if you add a second MQInput node, the default name is MQInput1; the third is MQInput2, and so on.
6. Repeat steps 4 and 5 to add further nodes.
7. You can also add nodes from other flows into this flow:
   a. Open the other message flow.
   b. Select the node or nodes that you want to copy from the editor or outline views, and press Ctrl+C or click **Edit** → **Copy**.
   c. Return to the flow with which you are currently working.
   d. Press Ctrl+V or click **Edit** → **Paste**. This action copies the node or nodes into your current flow. The node names and properties are preserved in the new copy.

When you have added the nodes that you want in this message flow, you can connect them to specify the flow of control through the message flow, and you can configure their properties.

Next: configure the nodes.

Adding a node using the keyboard

Before you start:
- Create a message flow or open an existing message flow
- Read the concept topic about message flow nodes

You can use the keyboard to perform tasks in the Message Flow editor, such as adding a node to the canvas.

1. Switch to the Broker Application Development perspective.
2. Open the message flow to which you want to add a node.
3. Open the Palette view or the Palette bar.
4. Select a node in the Palette view or Palette bar using the up and down arrows to highlight the node that you want to add to the canvas.
5. Add the Node to the canvas using one of the following methods:
   - Press **Alt + L**, then press **N**.
   - Press **Shift + F10** to open the context-sensitive menu for the Palette, and press **N**.

   The node that you selected in the Palette bar or Palette view is placed on the canvas in the Editor view.

   When you add a node to the canvas, the editor automatically assigns a name to the node, but the name is highlighted and you can change it by entering a name of your choice. If you do not change the default name at this time, you can change it later. The default name is set to the type of node for the first instance. For example, if you add an MQInput node to the canvas, it is given the name MQInput; if you add a second MQInput node, the default name is MQInput1; the third is MQInput2, and so on.

You can move the node that you have placed on the canvas using the keyboard controls described in Message Brokers Toolkit keyboard shortcuts.

Adding a subflow

Within a message flow, you might want to include an embedded message flow, also known as a subflow.

Before you start

To complete this task, you must have completed one of the following tasks:
- "Creating a message flow" on page 44
- "Opening an existing message flow" on page 45

When you add a subflow, it appears in the editor view as a single node.

You can embed subflows into your message flow if either of the following statements is true:
- The flow that you want to embed is defined in the same message flow project.

- The flow is defined in a different message flow project, and you have specified the dependency of the current message flow project on that other project.

To add a subflow to a message flow:

1. Switch to the Broker Application Development perspective.
2. Open the message flow that you want to work with.
3. Drag and drop the message flow from the Navigator view into the editor view. Alternatively, highlight the embedding message flow and click **Edit** → **Add subflow**, which displays a list of valid flows that you can add to the current flow.
4. Select the flow that you want to add from the list. The subflow icon is displayed with the terminals that represent the Input and Output nodes that you have included in the subflow.
5. Click **OK**.
6. Repeat steps 3, 4, and 5 to add further subflow nodes.
7. Select and open (double-click) the flow by name in the Navigator view, or right-click the embedded flow icon and select **Open Subflow** to work with the contents of the embedded flow

When you have added the nodes that you want in this message flow, you can connect them to specify the flow of control through the message flow, and you can modify their properties.

Renaming a message flow node

Before you start:

- Create a message flow
- Read the concept topic about message flow nodes

You can change the name of any type of node (a built-in node, user-defined node, or subflow node) to reflect its purpose. When you first add a node to the canvas, the editor automatically assigns a name to the node, but the name is highlighted and you can change it by entering a name of your choice. If you do not change the default name at this time, you can change it later, as described in this topic.

When you rename a node, use only the supported characters for this entity. The editor prevents you from entering unsupported characters.

To rename a node:

1. Switch to the Broker Application Development perspective.
2. Open the message flow with which you want to work.
3. You can rename a node in three ways:
   - Right-click the node and click **Rename**. The name is highlighted; enter a name of your choice and press Enter.
   - Click the node to select it, then click the node's name so that it is highlighted; enter a name of your choice and press Enter.
   - Click the node to select it, then on the Description tab of the Properties view, enter a name of your choice in the Node name field.

   The name that you enter must be unique within the message flow.

Configuring a message flow node

Before you start:
- Read the concept topic about message flow nodes
- Add a node

When you have included an instance of a node in your message flow, you can configure its properties to customize how it works. The node can be a built-in node, a user-defined node, or a subflow node. You can choose whether the properties appear in the Properties view below the Message Flow editor, or in the Properties dialog box. By default, properties are shown in the Properties view. To switch to the Properties dialog box, take the following steps:

1. Click **Window → Preferences** to open the Preferences dialog box.
2. Expand **Broker Development** and click **Message Flow Editor**.
3. Select **Show node properties in a Properties Dialog**.

Viewing a node's properties

To view a node's properties:

1. Switch to the Broker Application Development perspective.
2. Open the message flow with which you want to work.
3. Click **Selection** above the node palette.
4. Right-click a node and click **Properties** to open the Properties view or Properties dialog box. Alternatively, you can double-click the node to display its properties.

Editing a node's properties

Properties are organized into related groups and displayed on tabs. Each tab is listed on the left of the Properties view or Properties dialog box. Click each tab to view the properties that you can edit.

- Every node has at least one tab, Description, where you can change the name of the node and enter short and long descriptions. The description fields are optional because they are used only for documentation purposes.
- If a property is mandatory, that is, one for which you must enter a value, the property name is marked with an asterisk, as shown in the following example:

  ```
  Queue Name* _______________________________
  ```

For details of how to configure each individual built-in node, see the node description. You can find a list of the nodes, with links to the individual topics, in "Built-in nodes" on page 126.

Promoted properties

You can promote node properties to their containing message flow. Use this technique to set some values at the message flow level, without having to change individual nodes. This can be useful, for example, when you embed a message flow in another flow, and want to override some property such as output queue with a value that is correct in this context.

Overriding properties at deploy time

A small number of node property values can be overridden when a message flow is deployed. These are known as configurable properties, and you can use these to modify some characteristics of a deployed message flow without changing the message flow definitions. For example, you can update queue manager information.

Even though you can set values for configurable properties at deploy time, you must set values for these properties within the message flow if they are mandatory.

Next: connect the nodes.

Removing a message flow node

When you have created and populated a message flow, you might need to remove a node to change the function of the flow, or to replace it with another more appropriate node. The node can be a built-in node, a user-defined node, or a subflow node.

Before you start:
- Add a node
- Add a subflow
- Read the concept topic about message flow nodes

To remove a node:
1. Switch to the Broker Application Development perspective.
2. Open the message flow that you want to work with.
3. Select the node in the editor view and press the Delete key.
4. Highlight the node and click **Edit** → **Delete**

   You can also right-click the node in the editor view and click **Delete**, or right-click the node in the Outline view and click **Delete**. The editor removes the node. If you have created any connections between that node and any other node, those connections are also deleted when you delete the node.
5. If you delete a node in error, you can restore it by right-clicking in the editor view and clicking **Undo Delete**. The node and its connections, if any, are restored.
6.

   You can also click **Edit** → **Undo Delete** or press Ctrl+Z.
7. If you undo the delete, but decide it is the correct delete action, you can right-click in the editor view and click **Redo Delete**.

   You can also click **Edit** → **Redo Delete**.

Connecting message flow nodes

When you include more than one node in your message flow, you must connect the nodes to indicate how the flow of control passes from input to output. The nodes can be built-in nodes, user-defined nodes, or subflow nodes.

Before you start:
- Add a node
- Add a subflow
- Read the concept topic about connections

Your message flow might contain just one MQInput node, and one MQOutput node. Or it might involve a large number of nodes, and perhaps embedded message flows, that provide a number of paths that a message can travel through depending on its content.

When you have completed a connection, it is displayed as a black line, and is drawn as close as possible to a straight line between the connected terminals. This might result in the connection passing across other nodes. To avoid this, you can add bend points to the connection.

In the Message Flow editor, you can display node and connection metadata by hovering the mouse over a node or subflow in a message flow. To view metadata information for a node, subflow, or connection:

1. Switch to the Broker Application Development perspective.
2. Open a message flow.
3. In the Message Flow editor, hover the mouse over a node, a subflow, or a node connection in the open message flow by placing the mouse over the element.

A custom tooltip is displayed below the element.

- To turn the pop-up window into a scrollable window, press **F2**.
- To hide the pop-up window, either press Esc or move the mouse away from the node.

If you define a complex message flow, you might have to create a large number of connections. The principle is the same for every connection. You create connections either by using the mouse, or by using the Terminal Selection dialog. See “Creating node connections with the mouse” and “Creating node connections with the Terminal Selection dialog box” on page 61 for more information.

Next: add a bend point.

Creating node connections with the mouse

Before you start:

Read the concept topic about connections.

To connect one node to another using the mouse:

1. Switch to the Broker Application Development perspective.
2. Open the message flow with which you want to work.
3. Click the terminal from which the connection is to be made; that is, the terminal from which the message is propagated from the current node.

   For example, you can click the failure, out, or catch terminal of the MQInput node. Hover the mouse over each terminal to see the name of the terminal. You do not need to keep the mouse button pressed.

   Alternatively, click **Connection** on the palette, then click the node from which the connection is to be made. The Terminal Selection dialog box opens for you to choose the terminal from which to make a connection. Click **OK**.
4. Click the in terminal of the next node in the message flow (to which the message passes for further processing). The connection is made when you click a valid in terminal. The connection appears as a black line between the two terminals.

In the Message Flow editor, you can display node and connection metadata by hovering the mouse over a node or subflow in a message flow. To view metadata information for a node, subflow, or connection:

1. Switch to the Broker Application Development perspective.
2. Open a message flow.
3. In the Message Flow editor, hover the mouse over a node, a subflow, or a node connection in the open message flow by placing the mouse over the element.

A custom tooltip is displayed below the element.

- To turn the pop-up window into a scrollable window, press **F2**.
- To hide the pop-up window, either press Esc or move the mouse away from the node.

Creating node connections with the Terminal Selection dialog box

Before you start:

Read the concept topic about connections.

Use the Terminal Selection dialog box to connect one node to another:

1. Switch to the Broker Application Development perspective.
2. Open the message flow with which you want to work.
3. Click **Connection** above the node palette.
4. Click the node from which you want the connection to be made. The Terminal Selection dialog box is displayed.
5. Select the terminal from the list of valid terminals on this node. Click **OK**. The dialog box closes.
6. Click the node to which to make the connection. If this node has only one in terminal, the connection is made immediately. If this node has more than one in terminal, the Terminal Selection dialog is displayed again, listing the in terminals of the selected node. Select the correct terminal by clicking it, and click **OK**.

You can also make a connection as follows:

1. Click **Selection** above the node palette.
2. Right-click the node from which you want to make the connection and click **Create Connection**. The Terminal Selection dialog box is displayed.
3. Select the terminal from the list of valid terminals on this node. Click **OK**. The dialog box closes.
4. Click the node to which to make the connection. If this node has only one in terminal, the connection is made immediately. If this node has more than one in terminal, the Terminal Selection dialog box is displayed again, listing the in terminals of the selected node. Highlight the correct terminal and click **OK**.

In the Message Flow editor, you can display node and connection metadata by hovering the mouse over a node or subflow in a message flow. To view metadata information for a node, subflow, or connection:

1. Switch to the Broker Application Development perspective.
2. Open a message flow.
3. In the Message Flow editor, hover the mouse over a node, a subflow, or a node connection in the open message flow by placing the mouse over the element.

A custom tooltip is displayed below the element.

- To turn the pop-up window into a scrollable window, press **F2**.
- To hide the pop-up window, either press Esc or move the mouse away from the node.

Removing a node connection

The message flow editor displays the nodes and connections in the editor view. You can remove connections to change the way in which the message flow processes messages.

Before you start:
- Connect the nodes
- Read the concept topic about connections

If you want to remove a connection that you have created between two nodes:

1. Switch to the Broker Application Development perspective.
2. Open the message flow that you want to work with.
3. Click **Selection** above the node palette.
4. Select the connection that you want to delete. When you hover your mouse pointer over the connection, the editor highlights the connection that you have selected by thickening its line, adding an arrowhead at the target terminal end, and annotating the connection with the name of the two terminals connected, for example Out->In.

   When you select the connection, the editor appends a small black square at each end and at every bend point of the connection, and a small arrowhead at the target terminal end. The annotation disappears when you select the connection.
5. Check that the selected connection is the one that you want to delete.
6. Right-click the connection and click **Delete**, press the Delete key, or click **Edit** → **Delete**. If you want to delete further connections, repeat these actions from step 4.
7. If you delete a connection in error, you can restore it by right-clicking in the editor view and clicking **Undo Delete**. The connection is restored.
8. If you undo the delete, but decide that it is the correct delete action, you can right-click in the editor view and click **Redo Delete**. You can also delete a connection by selecting it in the Outline view and pressing the Delete key.

If you delete a node, its connections are automatically removed; you do not have to do this as a separate task.

Adding a bend point

When you are working with a message flow, and connecting your chosen nodes together to determine the flow of control, you might find that a connection that you have made crosses over an intervening node and makes the flow of control difficult to follow.

To help you to display the message flow nodes and their connections in a clear way, you can add bend points to the connections that you have made to improve the organization of the display. The addition of bend points has no effect on the execution of the nodes or the operation of the message flow.

Before you start:

- Connect the nodes
- Read the concept topic about bend points

To add a bend point:

1. Switch to the Broker Application Development perspective.
2. Open the message flow that you want to work with.
3. Click **Selection** above the node palette.
4. Select the connection to which you want to add a bend point. The editor appends a small black square to each end of the connection to highlight it.
   a. Check that this is the correct connection. The editor also adds a small point (a handle) in the connection halfway between the in and out terminals that are joined by this connection.
5. Hover your mouse pointer over this point until the editor displays a black cross to indicate that you now have control of this bend point.
   a. Hold down the left mouse button and move your mouse to move the black cross and bend point across the editor view.
6. As you drag your mouse, the connection is updated, retaining its start and end points with a bend point at the drag point. You can move this anywhere within the editor view to improve the layout of your message flow.
7. Release the mouse button when the connection is in the correct place. The editor now displays the bend point that you have created with a small square (like those at the ends of the connection), and displays another two small points within the connection, one between your newly-created bend point and the out terminal, the other between the new bend point and the in terminal.

If you want to add more than one bend point to the same connection, repeat these actions from step 4 using the additional small points inserted into the connection.

Next: align and arrange the nodes.

Removing a bend point

When you are working with a message flow in the editor view, you might want to simplify the display of the message flow by removing a bend point that you previously added to a connection between two nodes.

Before you start:

- Add a bend point
- Read the concept topic about bend points

To remove a bend point:

1. Switch to the Broker Application Development perspective.
2. Open the message flow that you want to work with.
3. Click **Selection** above the node palette.
4. Select the connection from which you want to remove the bend point. The editor highlights the connection and its current bend points by thickening its line and appending a small black square to each end of the connection, and by indicating each bend point with a small black square. Check that this is the correct connection.
5. Right-click over the selected connection, if you added this bend point in the current edit session.

a. Click **Undo Create Bend Point**.

   The editor removes the selected bend point.

   If you right-click in the editor view without a connection being selected, you can also click **Undo Create Bend Point** from the menu. However, this removes the last bend point that you created in any connection, which might not be the one that you want to remove.

6. Move the bend point to straighten the line if you added this bend point in a previous edit session, because you cannot use the undo action. When the line is straight, the bend point is removed automatically.

   When the bend point has been removed, the connection remains highlighted. Both ends of the connection, and any remaining bend points, remain displayed as small black squares. The editor also inserts small points (handles) into the connection between each bend point and between each terminal and its adjacent bend point, which you can use to add more bend points if you choose.
7. If you want to remove another bend point from the same connection, repeat these actions from step 4 on page 63.

Aligning and arranging nodes

When you are working in the Message Flow editor, you can decide how your nodes are aligned within the editor view.

This option is closely linked to the way in which your nodes are arranged. Again, the default for this is left to right, which means that the in terminal of a node appears on its left edge, and its out terminals appear on its right edge. You can also change this characteristic of a node by rotating the icon display to right to left, top to bottom, and bottom to top.

Before you start

To complete this task, you must have completed the following task:
- "Adding a message flow node" on page 55

To modify the way in which nodes and connections are displayed in the editor:
1. Switch to the Broker Application Development perspective.
2. Open the message flow that you want to work with.
3. Click **Selection** above the node palette.
4. Right-click in the editor window and select **Manhattan Layout** if you want the connections between the nodes to be displayed in Manhattan style; that is with horizontal and vertical lines joined at right angles.
5. If you want to change the layout of the complete message flow:
   a. Right-click in the editor view and click **Layout**. The default for the alignment is left to right, such that your message flow starts (with an input node) on the left and control passes to the right.
   b. From the four further options displayed, **Left to Right**, **Right to Left**, **Top to Bottom**, and **Bottom to Top**, click the option that you want for this message flow. The message flow display is updated to reflect your choice. As a result of the change in alignment, all the nodes within the message flow are also realigned.

For example, if you have changed from a left to right display (the default) to a right to left display, each node in the flow has now also changed to right to left (that is, the in terminal now appears on the right edge, the out terminals appear on the left edge).

6. You might want to arrange an individual node in a different direction from that in which the remaining nodes are arranged within the message flow, To do this:
   a. Right-click the node that you want to change and click **Rotate**. This gives you four further options: **Left to Right**, **Right to Left**, **Top to Bottom**, and **Bottom to Top**.
   b. Click the option that you want for this node. The option that represents the current arrangement of the node is not available for selection.

If you change the alignment of the message flow, or the arrangement of an individual node, or both, these settings are saved when you save the message flow. They are applied when another user accesses this same message flow, either through a shared repository or through shared files or import and export. When you reopen the message flow, you see these changed characteristics. The alignment and arrangement that you have selected for this message flow have no impact on the alignment and arrangement of any other message flow.

In the Message Brokers Toolkit Version 5.1 you can adjust the zoom by right-clicking in the editor view and clicking **Zoom in** or **Zoom out**. Alternatively, you can use the drop-down list on the editor toolbar to specify a zoom percentage.

You can also access the editor toolbar to select other options related to the display and arrangement of nodes, for example, snap to grid. These are defined in Message Flow editor.

Defining a promoted property

Before you start:

Read the concept topic about promoted properties.

When you create a message flow, you can promote properties from individual nodes within that message flow to the message flow level. Properties promoted in this way override the property values that you have set for the individual nodes. You can perform the following tasks related to promoting properties:
- “Promoting a property”
- “Renaming a promoted property” on page 69
- “Removing a promoted property” on page 70
- “Converging multiple properties” on page 71

Some of the properties that you can promote to the message flow are also configurable; you can modify them when you deploy the broker archive file in which you have stored the message flow to each broker. If you set values for configurable properties when you deploy a broker archive file, the values that you set override values set in the individual nodes, and those that you have promoted.

Promoting a property

You can promote a node property to the message flow level to simplify the maintenance of the message flow and its nodes, and to provide common values for multiple nodes within the flow by converging promoted properties.

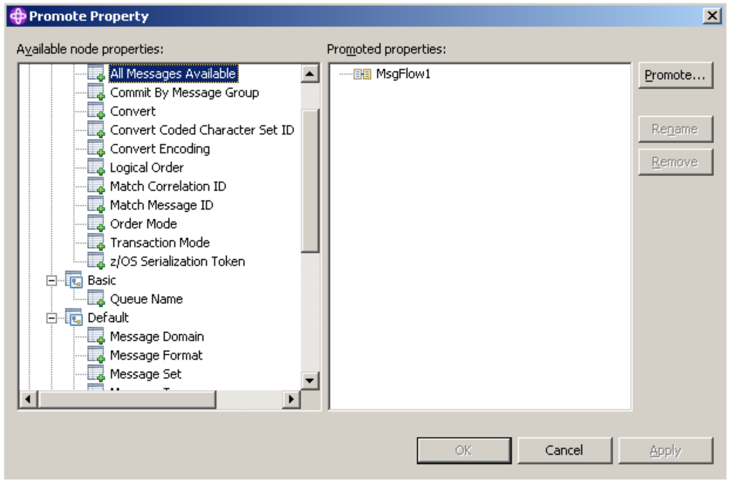
Before you start:
- Create a message flow
- Read the concept topic about promoted properties

To promote message flow node properties to the message flow level:

1. Switch to the Broker Application Development perspective.
2. Open the message flow for which you want to promote properties by double-clicking the message flow in the Navigator view. You can also open the message flow by right-clicking it in the Navigator view and clicking **Open** The message flow contents are displayed in the editor view.

   If this is the first message flow that you have opened, the message flow control window and the list of available built-in message flow nodes are also displayed, to the left of the editor view.
3. In the editor view, right-click the symbol of the message flow node whose properties you want to promote.
4. Select **Promote Property**.

   The Promote Property dialog is displayed.



   The left side of the dialog lists all available properties for all the nodes within the message flow. The properties for the node that you highlighted are expanded. You can access the properties for all the nodes in the open message flow from this dialog, regardless of the node that you selected when you first opened the dialog, by expanding the properties for all the other nodes in the flow (these are initially collapsed).

   The right side of the dialog lists the name of the open message flow and all the properties that are currently promoted to the message flow. If you have not yet promoted any properties, only the message flow name as the root of the promoted property tree is displayed on the right. In the image shown the message flow contains no promoted properties so only the name of the message flow is displayed.

   The majority of message flow node properties are available for promotion, but you cannot promote the following properties:
   - A property group, but you can promote an individual property.
   - A property that you cannot edit (for example, the *Fix* property in the Validate group of properties for the MQInput node).
   - The description properties (Short Description and Long Description).

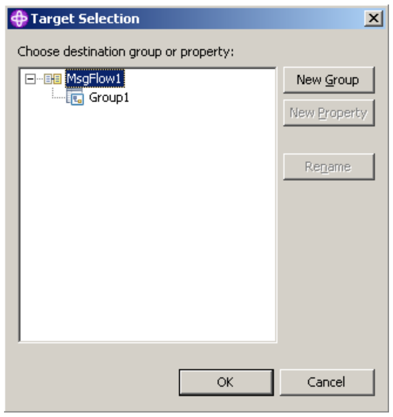
5. Select the property that you want to promote to the message flow. The list on the left initially shows the expanded list of all available properties for the selected node. If you have already promoted properties from this node, they do not appear on the left, but on the right.

   The list on the left also includes the other nodes in the open message flow. You can expand the properties listed under each node and work with all these properties at the same time. You do not have to close the dialog and select another node from the editor view to continue promoting properties.

   You can select multiple properties to promote by selecting a property, holding down Ctrl, and selecting one or more other properties.

   If you have you selected multiple properties to promote, all the properties you have selected must be available for promotion. If one or more of the selected properties is not available for promotion, the entire selection becomes unavailable for promotion, and the **Promote** button in the right-hand pane is grayed out.
6. Click the **Promote** button to promote the property or properties

   Clicking the Promote button invokes the Target Selection dialog:

   The Target Selection dialog displays only the valid targets for the promotion of the previously selected property or properties and allows you to create a new target for the promotion, such as to a new group or to a new property.
7. In the Target Selection dialog, select the destination group or property for the property or properties you want to promote. You can group together related properties from the same or different nodes in the message flow by dropping the selected property or properties onto a group or property that already exists. Alternatively, you can click **New Group** or **New Property** to create a new target for the promotion. You can rename groups and properties by selecting them and clicking **Rename**, or by double-clicking on the group or property.
8. Click **OK** to confirm your selections.

   Note: If you create a new group or property using the Target Selection dialog, the changes persist even if you select **Cancel** in the dialog. When the dialog closes, groups or properties that you have created using the Target Selection dialog will appear in the Promote properties dialog.
9. When you have selected the properties that you want to promote to the message flow, click **OK** Your updates are committed, and the Promoted Property dialog is closed. If you click **Apply**, this commits the changes but leaves the dialog open.

   When you have promoted a property, you can no longer make any changes to that property through the node properties dialog. You can only update its value at the message flow level.

Note: You can also promote properties from the Promote property dialog by dragging the selected property or properties from the left-hand pane of the Promote Property dialog to the right-hand pane:

1. Select the property you want to promote. You can select multiple properties to promote by selecting a property, holding down Ctrl, and selecting one or more other properties.
2. You can drop the selected property or properties into the right-hand pane using the following methods:
   a. Drop the selected property or properties in an empty space.

      A new group is automatically created for the message flow, and the property is placed within it, with the original name of the property and the name of the message flow node from which it came displayed beneath the property entry.

      The name of the first group created defaults to Group1. If a group called Group1 already exists, the group is given the name Group2, and so on. You can rename the group by double-clicking it and entering new text or by selecting the group in the Promoted properties pane and clicking **Rename**.

      Note:

      When you create a new promoted property, the name that you enter is the name by which the property is known within the system, and must meet certain Java and XML naming restrictions. These are enforced by the dialog, and a message is displayed if you enter a name that includes a non-valid character. For example, you cannot include a space or the double quote symbol.

   b. Drop the selected property or properties onto a group that already exists, to group together related properties from the same or different nodes in the message flow.

      For example, you might want to group all promoted properties that relate to database interactions. You can change the groups that promoted properties belong to at any time, by selecting a property in the Promoted properties pane and dragging it onto a different group.

   c. Drop the selected property or properties onto a property that already exists, to converge related properties from the same or different nodes in the message flow.

      For example, you might want to create a single promoted property that overrides the property on each node that defines a data source.

      For more information on converging properties, see “Converging multiple properties” on page 71.

The message flow node properties are now promoted to the message flow. To confirm this, right-click the message flow in the Navigator view, or right-click the editor view, and select **Properties**.

The Properties dialog of the message flow is displayed, showing the message flow node properties that you have promoted, organized in the groups that you have created. If you now set a value for one of these properties, that value appears as the default value for the property whenever the message flow is itself included in other message flows.

When you have promoted a property, you can no longer make any changes to that property through the node properties dialog. You can only update its value at the message flow level.

When you select an embedded message flow within another message flow (a subflow) and view its properties, you see the promoted property values. If you look inside the embedded flow (that is, if you select **Open Subflow**), you see the original values for the properties. The value of a promoted property does not replace the original property, but it takes precedence when you deploy the message flow.

Promoting mandatory properties

If you promote a property that is mandatory (that is, an asterisk appears beside the name in the properties dialog of the message flow node), the mandatory characteristic of the property is preserved. When a mandatory property is promoted, its value does not need to be set at the node-level. If the flow containing the mandatory promoted property is included as a subflow within another flow, then the property has to be filled in for the subflow node.

Promoting properties through a hierarchy of message flows

You can repeat the process of promoting message flow node properties through several levels of message flow. You can promote properties from any level in the hierarchy to the next level above, and so on through the hierarchy to the top level. The value of a property is propagated from the highest point in the hierarchy at which it is set down to the original message flow node when the message flow is deployed to a broker. The value of that property on the original message flow node is overridden.

Renaming a promoted property

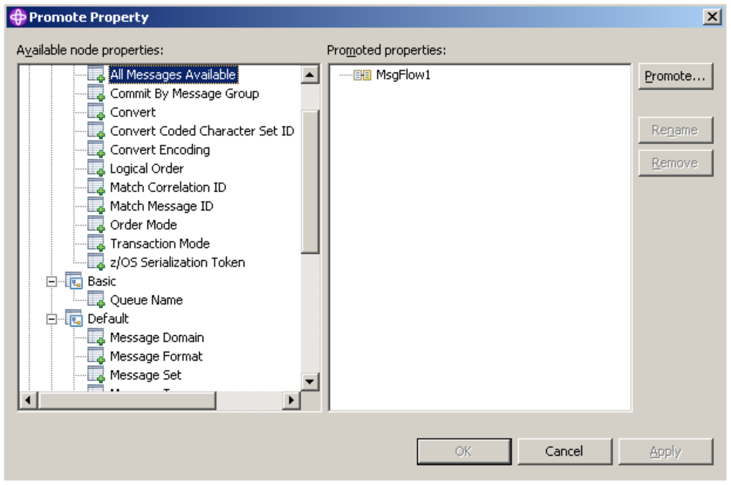
If you have promoted a property from the node to the message flow level, it is initially assigned the same name that it has at the node level. You can rename the property to have a more meaningful name in the context of the message flow.

Before you start:
- Promote a property
- Read the concept topic about promoted properties

To rename a promoted property :
1. Switch to the Broker Application Development perspective.
2. Open the message flow for which you want to promote properties by double-clicking the message flow in the Navigator view. You can also open the message flow by right-clicking it in the Navigator view and clicking **Open** The message flow contents are displayed in the editor view.

   If this is the first message flow that you have opened, the message flow control window and the list of available built-in message flow nodes are also displayed, to the left of the editor view.
3. In the editor view, right-click the symbol of the message flow node whose properties you want to promote.
4. Select **Promote Property**.

The Promote Property dialog is displayed.



5. Promoted properties are shown in the Promoted properties pane on the right of the Promote property dialog. Double-click the promoted property in the list of properties that are currently promoted to the message flow level, or select the property you want to rename and click **Rename**. The name is highlighted, and you can edit it. Modify the existing text or enter new text to give the property a new name, and press Enter.
6. Click **Apply** to commit this change without closing the Property Promotion dialog. Click **OK** to complete your updates and close the dialog.

Removing a promoted property

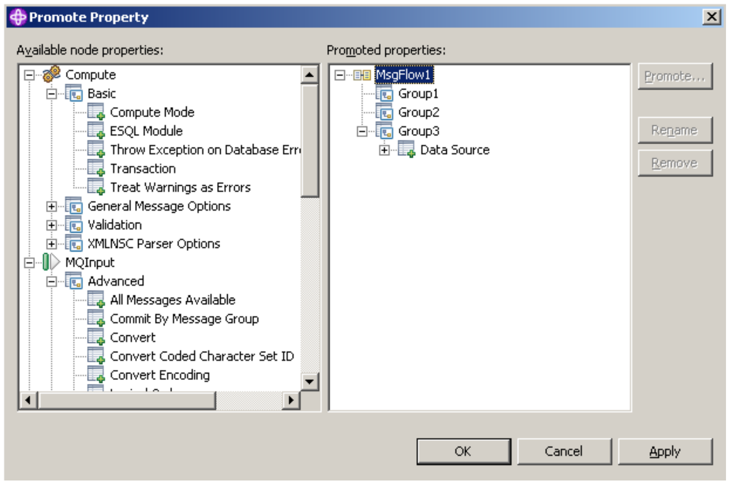
If you have promoted a property from the node to the message flow level, you can remove (delete) it if you no longer want to specify its value at the message flow level. The property reverts to the value that you specified at the node level. If you remove a promoted property that is a mandatory property, ensure that you have set a value at the node level. If you have not, you cannot successfully deploy a broker archive file that includes this message flow.

Before you start:
- Promote a property
- Read the concept topic about promoted properties

If you have promoted one or more message flow node properties, and want to delete them:
1. Switch to the Broker Application Development perspective.
2. Open the message flow for which you want to promote properties by double-clicking the message flow in the Navigator view. You can also open the message flow by right-clicking it in the Navigator view and clicking **Open** The message flow contents are displayed in the editor view.

   If this is the first message flow that you have opened, the message flow control window and the list of available built-in message flow nodes are also displayed, to the left of the editor view.
3. In the Editor view, right-click the symbol of the message flow node whose properties you want to promote.
4. Select **Promote Property**.

The Promote Property dialog is displayed.



5. Select the promoted property that you want to remove in the list of properties on the right of the dialog, and click **Remove**. The property is removed from the list on the right. It is restored to the list on the left, in its appropriate place in the tree of properties for the node from which you promoted it. You can promote this property again if you choose.
6. If you want to delete all the promoted properties within a single group, select the group in the list on the right and click **Remove**. The group and all the properties it contains are deleted from this list: the individual properties that you promoted are restored to the nodes from which you promoted them.
7. Click **Apply** to commit this change without closing the Property Promotion dialog. Click **OK** to complete your updates and close the dialog.

If you have included this message flow in a higher-level message flow, and have set a value for a promoted property that you have now deleted, the embedding flow is not automatically updated to reflect the deletion. However, when you deploy that embedding message flow in the broker domain, the deleted property is ignored.

Converging multiple properties

You can promote properties from several nodes in a message flow to define a single promoted property that provides a single value to be for that property in all those nodes.

Before you start:

- Create a message flow
- Read the concept topic about promoted properties

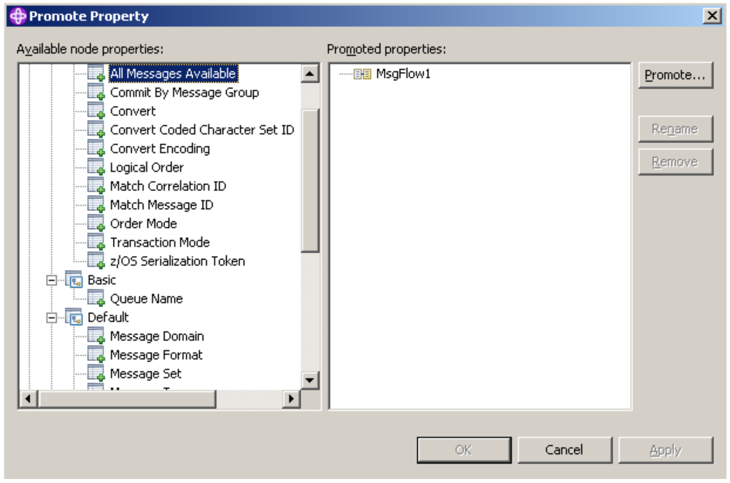
To converge multiple node properties to a single promoted property:

1. Switch to the Broker Application Development perspective.
2. Open the message flow for which you want to promote properties by double-clicking the message flow in the Navigator view. You can also open the message flow by right-clicking it in the Navigator view and clicking **Open** The message flow contents are displayed in the editor view.

   If this is the first message flow that you have opened, the message flow control window and the list of available built-in message flow nodes are also displayed, to the left of the editor view.

3. In the editor view, right-click the symbol of the message flow node whose properties you want to promote.
4. Select **Promote Property**.

   The Promote Property dialog box is displayed.



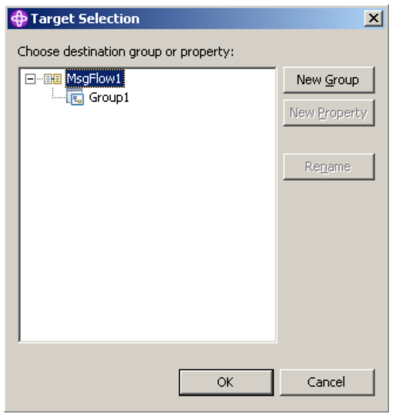
5. Select the property that you want to converge. The list on the left initially shows the expanded list of all available properties for the selected node. If you have already promoted properties from this node, they do not appear on the left, but on the right.

   The list on the left also includes the other nodes in the open message flow. You can expand the properties listed under each node and work with all these properties at the same time. You do not have to close the dialog box and select another node from the editor view to continue promoting properties.

   You can select multiple properties to promote by selecting a property, holding down Ctrl, and selecting one or more other properties.

   If you have you selected multiple properties to converge, all the properties you have selected must be available for promotion. If one or more of the selected properties is not available for promotion, the entire selection becomes unavailable for promotion, and the **Promote** button in the right-hand pane is grayed out.
6. Click the **Promote** button to promote the property or properties

   Clicking the Promote button invokes the Target Selection dialog box:



   The Target Selection dialog box displays only the valid targets for the promotion of the previously selected property or properties and allows you to create a new target for the promotion, such as to a new group or to a new property.

7. To converge properties from the same or different nodes in the message flow, expand the tree and click on a property that already exists. You can rename the properties by selecting them and clicking **Rename**, or by double-clicking on the group or property.
8. Click **OK** to confirm your selections.

   Note: If you create a new group or property using the Target Selection dialog box, the changes persist even if you select **Cancel** in the dialog box. When the dialog box closes, groups or properties that you have created using the Target Selection dialog box appear in the Promote properties dialog box.
9. Expand the property trees for all the nodes for which you want to promote properties.
   a. Drag the first instance of the property that you want to converge from the list on the left, and drop it on the appropriate group in the list on the right. If the group already contains one or more promoted properties, the new property is added at the end of the group. Rename the new property if you want to by double-clicking the property, or by selecting the property and clicking **Rename**.

      If you want the promoted property to appear in a new group, you can drag and drop the property into an empty space below the existing groups, which forces a new group to be created. You can also place the promoted property in a new group by selecting the property you want to promote, and clicking **Promote**, which opens the Target Selection dialog box. Click **New Group**, and enter the name of the new group. Click **OK** to confirm your changes.

      If you drag the property onto an existing promoted property of a different type, a no-entry icon is displayed and you cannot drop the property. You must create this as a new promoted property, or drop it onto a compatible existing promoted property. Properties must be associated with the same property editor to be compatible. For example, if you are using built-in nodes, you can only converge like properties (string with string, boolean with boolean).
10. Drag all remaining instances of the property from each of the nodes in the list on the left onto the existing promoted property. The new property is added under the existing promoted property, and is not created as a new promoted property.
11. Click **Apply** to commit this change without closing the Property Promotion dialog box. Click **OK** to complete your updates and close the dialog box.

Note: You can also converge properties from the Promote property dialog box by dragging the selected property or properties from the left-hand pane of the Promote Property dialog box to the right-hand pane:

1. Select the property you want to promote. You can select multiple properties to promote by selecting a property, holding down Ctrl, and selecting one or more other properties.
2. Drop the selected property or properties onto a property in the right-hand pane to converge related properties from the same or different nodes in the message flow.

   For example, you might want to create a single promoted property that overrides the property on each node that defines a data source.

   For more information on converging properties, see “Converging multiple properties” on page 71.

Collecting message flow accounting and statistics data

Before you start:

Read the concept topic about message flow accounting and statistics data.

You can collect statistics on message flow behavior.

The following topics describe the tasks that you can complete to control collection of message flow accounting and statistics data:
- "Starting to collect message flow accounting and statistics data"
- "Stopping message flow accounting and statistics data collection" on page 76
- "Viewing message flow accounting and statistics data collection parameters" on page 77
- "Modifying message flow accounting and statistics data collection parameters" on page 77
- "Resetting message flow accounting and statistics archive data" on page 78

The topics listed here show examples of how to issue the accounting and statistics commands. The examples for z/OS are shown for SDSF; if you are using another interface, you must modify the example shown according to the requirements of that interface. For details of other z/OS options, see Issuing commands to the z/OS console.

Starting to collect message flow accounting and statistics data

Before you start:
- Create a message flow
- Deploy a broker archive file
- Read the concept topic about message flow accounting and statistics collection options

You can start collecting message flow accounting and statistics data for an active broker at any time.

Select the granularity of the data that you want to be collected by specifying the appropriate parameters on the mqsichangeflowstats command. You must request statistics collection on a broker basis. If you want to collect information for more than one broker, you must issue the corresponding number of commands.

To start collecting message flow accounting and statistics data:
1. Identify the broker for which you want to collect statistics .
2. Decide the resource for which you want to collect statistics. You can collect statistics for a specific execution group, or for all execution groups for the specified broker.
   - If you indicate a specific execution group, you can request that data is recorded for a specific message flow or all message flows in that group.
   - If you specify all execution groups, you must specify all message flows.
3. Decide if you want to collect thread related statistics.
4. Decide if you want to collect node related statistics. If you do, you can also collect information about terminals for the nodes.

5. Decide if you want to associate data collection with a particular accounting origin. This option is valid for snapshot and archive data, and for message flows and execution groups. However, when active, you must set its origin value in each message flow to which it refers. If you do not configure the participating message flows to set the appropriate origin identifier, the data collected for that message flow is collected with the origin set to `Anonymous`.

   See "Setting message flow accounting and statistics accounting origin" on page 76 for further details.
6. Decide the target destination:
   - User trace log. This is the default setting. The output data can be processed using mqsireadlog and mqsiformatlog.
   - XML format publication message. If you chose this as your target destination, register the following topic for the subscriber:

```
$SYS/Broker/brokerName/StatisticsAccounting/recordType/executionGroupLabel/messageFlowLabel
```

     Where, *brokerName*, *executionGroupLabel*, and *messageFlowLabel* are the broker, execution group and message flow on which you want to receive data. *recordType* is the type of data collection on which you want to receive publications (snapshot, archive, or # to receive both snapshot and archive).
   - z/OS SMF (on z/OS only)
7. Decide the type of data collection that you want:
   - Snapshot
   - Archive

   You can collect snapshot and archive data at the same time, but you have to configure them separately.
8. Issue the mqsichangeflowstats command with the appropriate parameters to reflect the decisions that you have made.

   For example, to turn on snapshot data for all message flows in the default execution group for BrokerA, and include node data with the basic message flow statistics, enter:

   ```
   mqsichangeflowstats BrokerA -s -e default -j -c active -n basic
   ```

   z/OS Using SDSF on z/OS, enter:

   ```
   /F BrokerA,cs s=yes,e=default,j=yes,c=active,n=basic
   ```

   Refer to the mqsichangeflowstats command for further examples.

When the command completes successfully, data collection for the specified resources is started:

- If you have requested snapshot data, information is collected for approximately 20 seconds, and the results are written to the specified output.
- If you have requested archive data, information is collected for the interval defined for the broker (on the mqsicreatebroker or mqsichangebroker command, or by the external timer facility ENF). The results are written to the specified output, the interval is reset, and data collection starts again.

Next:

You can now perform the following tasks:

- "Setting message flow accounting and statistics accounting origin" on page 76
- "Stopping message flow accounting and statistics data collection" on page 76
- "Viewing message flow accounting and statistics data collection parameters" on page 77

- “Modifying message flow accounting and statistics data collection parameters” on page 77
- “Resetting message flow accounting and statistics archive data” on page 78

Setting message flow accounting and statistics accounting origin

Accounting and statistics data is associated with an accounting origin.

All message flow accounting and statistics data is collected with an accounting origin set to `Anonymous`. You cannot change this value.

Stopping message flow accounting and statistics data collection

You can stop collecting data for message flow accounting and statistics at any time. You do not have to stop the message flow, execution group, or broker to make this change, nor do you have to redeploy the message flow.

Before you start:
- Start to collect message flow accounting and statistics data
- Read the concept topic about message flow accounting and statistics data

You can stop collecting data for message flow accounting and statistics at any time. You do not have to stop the message flow, execution group, or broker to make this change, nor do you have to redeploy the message flow.

You can modify the parameters that are currently in force for collecting message flow accounting and statistics data without stopping data collection. See “Modifying message flow accounting and statistics data collection parameters” on page 77 for further details.

To stop collecting data:

1. Check the resources for which you want to stop collecting data.

   You do not have to stop all active data collection. If you choose, you can stop a subset of data collection. For example, if you started collecting statistics for all message flows in a particular execution group, you can stop doing so for a specific message flow in that execution group. Data collection for all other message flows in that execution group continues.
2. Issue the mqsichangeflowstats command with the appropriate parameters to stop collecting data for some or all resources.

   For example, to switch off snapshot data for all message flows in all execution groups for BrokerA, enter:

   ```
   mqsichangeflowstats BrokerA -s -g -j -c inactive
   ```

   z/OS Using SDSF on z/OS, enter:

   ```
   /F BrokerA,cs s=yes g=yes j=yes c=inactive
   ```

   Refer to the mqsichangeflowstats command for further examples.

When the command completes successfully, data collection for the specified resources is stopped. Any outstanding data that has been collected is written to the output destination when you issue this command, to ensure the integrity of data collection.

Viewing message flow accounting and statistics data collection parameters

You can review and check the parameters that are currently in effect for message flow accounting and statistics data collection.

Before you start:
- Start to collect message flow accounting and statistics data
- Read the concept topic about message flow accounting and statistics data

To view message flow accounting and statistics data collection parameters:

Issue the mqsireportflowstats command with the appropriate parameters to view the parameters that are currently being used by the broker to control archive data collection or snapshot data collection.

You can view the parameters in force for a broker, an execution group, or an individual message flow.

For example, to view parameters for snapshot data for all message flows in all execution groups for BrokerA, enter:

```
mqsireportflowstats BrokerA -s -g -j
```

z/OS Using SDSF on z/OS, enter:

```
/F BrokerA,rs s=yes,g=yes,j=yes
```

Refer to the mqsireportflowstats command for further examples.

The command displays the current status, for example:

```
BIP8187I: Statistics Snapshot settings for flow MyFlow1 in execution
group default - On?: inactive,
ThreadDataLevel: basic, NodeDataLevel:  basic,
OutputFormat: usertrace, AccountingOrigin: basic
```

Next:

You can now modify the data collection parameters.

Modifying message flow accounting and statistics data collection parameters

You can modify the parameters that you have set for message flow accounting and statistics data collection. For example, you can start collecting data for a new message flow that you have deployed to an execution group for which you are already collecting data.

You can modify parameters while data collection is active; you do not have to stop data collection and restart it.

Before you start:
- Start to collect message flow accounting and statistics data
- Read the concept topic about message flow accounting and statistics data

To modify message flow accounting and statistics parameters:

1. Decide which data collection parameters you want to change. You can modify the parameters that are in force for a broker, an execution group, or an individual message flow.
2. Issue the mqsichangeflowstats command with the appropriate parameters to modify the parameters that are currently being used by the broker to control archive data collection or snapshot data collection.

   For example, to modify parameters to extend snapshot data collection to a new message flow MFlow2 in execution group EG2 for BrokerA, enter:

   ```
   mqsichangeflowstats BrokerA -s -e EG2 -f MFlow2 -c active
   ```

   z/OS Using SDSF on z/OS, enter:

   ```
   /F BrokerA,cs s=yes,e=EG2,f=MFlow2,c=active
   ```

   If you want to specify an accounting origin for archive data for a particular message flow in an execution group, enter:

   ```
   mqsichangeflowstats BrokerA -a -e EG4 -f MFlowX -b basic
   ```

   z/OS Using SDSF on z/OS, enter:

   ```
   /F BrokerA,cs a=yes,e=EG4,f=MFlowX,b=basic
   ```

   Refer to the mqsichangeflowstats command for further examples.

When the command completes successfully, the new parameters that you have specified for data collection are in force. These parameters remain in force until you stop data collection or make further modifications.

Resetting message flow accounting and statistics archive data

You can reset message flow accounting and statistics archive data to purge any accounting and statistics data not yet reported for that collecting interval. This removes unwanted data. You can request this at any time; you do not have to stop data collection and restart it to perform reset. You cannot reset snapshot data.

Before you start:
- Start to collect message flow accounting and statistics data
- Read the concept topic about message flow accounting and statistics data

To reset message flow accounting and statistics archive data:
1. Identify the broker, and optionally the execution group, for which you want to reset archive data. You cannot reset archive data on a message flow basis.
2. Issue the mqsichangeflowstats command with the appropriate parameters to reset archive data.

   For example, to reset archive data for BrokerA, enter:

   ```
   mqsichangeflowstats BrokerA -a -g -j -r
   ```

   z/OS Using SDSF on z/OS, enter:

   ```
   /F BrokerA,cs a=yes,g=yes,j=yes,r=yes
   ```

When this command completes, all accounting and statistics data accumulated so far for this interval are purged and will not be included in the reports. Data collection is restarted from this point. All archive data for all flows (indicated by `-j` or `j=yes`) in all execution groups (indicated by `-g` or `g=yes`) is reset.

This command has a minimal effect on snapshot data because the accumulation interval is much shorter than for archive data. It does not effect the settings for archive or snapshot data collection that are currently in force. When the command has completed, data collection resumes according to the current settings.

You can change any other options that are currently in effect when you reset archive data, for example accounting origin settings or output type.

Part 2. Deploying

Deploying

The topics in this section provide information about deploying resources to execution groups on brokers. Read the overview section for information about the different mechanisms that you can use to deploy resources and about the different types of deployment:

- **Deployment overview**
  - Message flow application deployment
    - Broker archive file
    - Configurable properties
  - Broker configuration deployment
  - Publish/subscribe topology deployment
  - Publish/subscribe topics hierarchy deployment
  - Cancel deployment

The following topics describe the basic tasks associated with deployment:

- **Deploying a message flow application**
  - Creating a server project
  - Creating a broker archive
  - Adding files to a broker archive
    - Editing a broker archive file manually
    - Editing configurable properties
  - Refreshing the contents of a broker archive
  - Deploying a broker archive

Learn how to perform other types of deployment:

- Deploying a broker configuration
- Deploying a publish/subscribe topology
- Deploying a publish/subscribe topics hierarchy

Further topics describe other deployment-related tasks:

- Checking the results of deployment
- Canceling a deployment
- Renaming a deployed object
- Removing a deployed object

Deployment overview

Deployment is the process of transferring data to an execution group on a broker so that it can take effect in the broker domain. For deploying message flows and associated resources, the data is packaged in a broker archive (bar) file before being sent to the Configuration Manager, from where it is unpackaged and distributed appropriately.

This topic describes the environments from which you can perform a deployment, and then introduces a number of different types of deployment that you might need to use:

- “Deployment environments”
- “Types of deployment” on page 85

Most types of deployment can typically be configured to perform in one of two ways:

- Complete deployment; in which *everything* is deployed (or re-deployed) to the whole domain
- Delta or incremental deployment; made either to just update information or to deploy to selected brokers within the domain, depending on the type of deployment

After reading this conceptual overview, find detailed instructions for particular tasks in the subsequent topics.

Read the WebSphere Message Broker Basics Redbook for further information about deployment.

Deployment environments

Depending on the environment in which you are working, you can choose one of the following options to initiate a deployment:

Message Brokers Toolkit

In the Broker Administration perspective of the workbench, the Domain Navigator view displays all the objects associated with a specific domain. For example, if you expand the Topology view, all the brokers in the domain are displayed; if you expand a Broker view, all the execution groups within that broker are displayed. From the Domain Navigator view you can deploy a topology to all the brokers in the domain, or you can deploy all the execution groups to a particular broker. You can also drag a broker archive (bar) file from the Resource Navigator view onto an execution group within the Domain Navigator view to deploy the contents of the bar.

You might typically use the workbench if you are working in a development environment or if you are new to WebSphere Event Broker.

mqsideploy command

You can deploy from the command line using the mqsideploy command. On the command line, you must typically specify the connection details as well as parameters specific to the type of deployment you want to perform. Details are given in each topic describing the types of deployment.

You might typically use the mqsideploy command in a script when you are more familiar with WebSphere Event Broker.

WebSphere Event Broker provides two files to help you when writing your own scripts for managing broker deployment outside the workbench. These are:

- Initialization file mqsicfgutil.ini. This is a plain text file in the mqsideploy command’s working directory that contains configurable variables that are required to connect to the Configuration Manager. For example:

```
hostname = localhost
queueManager = QMNAME
port = 1414
securityExit = test.myExit
```

If you do not explicitly specify any of this information as parameters on the mqsideploy command (as has been done in the examples in subsequent topics), the information is taken from the mqsicfgutil.ini file.

Alternatively, use the -n parameter on the command to specify an XML-format .configmgr file that describes the connection parameters to the Configuration Manager.

- Windows Batch file mqsideploy.bat. The parameters used with the mqsideploy command in WebSphere Event Broker Version 6.0 are not the same as those used in earlier versions of the command. On Windows platforms, use mqsideploy.bat if you want to use the same parameters as in previous versions.

Configuration Manager Proxy API

You can control deployment from any Java program using the Configuration Manager Proxy API. You can also interrogate the responses from the broker and take appropriate action.

The Configuration Manager Proxy API also allows Java applications to control other objects in the domain, such as brokers, execution groups, publish/subscribe topologies, topics, subscriptions and the Configuration Manager and its event log. Because of this, you can use the Configuration Manager Proxy API to create and manipulate an entire domain programmatically.

Types of deployment

The other topics within this section describe what each type of deployment does, the situation in which each type should and should not be used.

- Broker configuration deployment

To deploy message flows, message sets and other deployable objects to an execution group, use:

- Message flow application deployment

  This uses a broker archive file for deployment. You can set configurable properties for objects in the message flow.

In publish/subscribe scenarios, you can deploy topics and topologies:

- Topics hierarchy deployment
- Topology configuration deployment

You can also cancel a deployment.

This table lists appropriate ways of deploying for a number of common scenarios.

Scenario	**Suggested deployment**
Adding a broker to the domain (when not using publish/subscribe)	None required.
Connecting publish/subscribe brokers using connections or a collective	Delta topology deployment
Modifying the publish/subscribe topic hierarchy	Delta deployment of the topics hierarchy (The changed elements in the topic hierarchy are deployed to all brokers in the domain.
Modifying the publish/subscribe topic hierarchy, after adding a new broker to the domain	Complete topics deployment (The entire topic hierarchy is deployed to all brokers in the domain. The new broker also receives the complete topic hierarchy.)

Scenario	Suggested deployment
Tidying up a broker's resources after removing it from the topology	If the broker is part of a publish/subscribe network, or if you are using the Message Brokers Toolkit, initiate a delta publish/subscribe topology deployment. Otherwise, no deployment is required.
Creating an execution group	Message flow application deployment using an incremental bar file deployment.
Deleting an execution group	None required.
If a broker is not responding to a deploy request	Ensure that the broker is running. If the broker is not running, cancel the broker deployment. You should only cancel a broker deployment if you are sure that the broker will never respond to the deploy request.

Message flow application deployment

You do not deploy a message flow application directly to an execution group. Instead, you package all the relevant resources into a broker archive (bar), which you then deploy. When you add files to the broker archive, they are automatically compiled as part of the process.

The broker archive itself is a compressed file which is sent to the Configuration Manager where its contents are extracted and distributed to execution groups. If an execution group has not been initialized on the broker (that is, if the broker has only just been created), then the execution group is created as part of the deployment.

There are two ways of deploying a bar file:

- Incremental deployment, in which deployed files are added to the execution group. Files which already exist in the execution group are replaced with the new version.
- Complete deployment, in which files already deployed to the execution group are removed before the complete contents of the bar file is deployed. Thus, nothing is left in the execution group from any previous deployment.

Incremental bar file deployment

Incrementally deploying a bar file tells the Configuration Manager to extract the contents of the bar file and send it to an execution group.

- If a file in the bar file has the same name as an object that is already deployed to the execution group, the version that is already deployed is replaced with the version in the bar file.
- If a file in the bar file is of zero length and a file of that name has already been deployed to the execution group, then the deployed file is removed from the execution group.

When to use it

- If you want to incrementally deploy message flows, message sets or other deployable objects to an execution group.

When *not* to use it

- If you want to completely clear the contents of the execution group before the bar file is deployed. In this case, use a complete bar file deployment instead.

Complete bar file deployment

Completely deploying a bar file tells the Configuration Manager to extract the deployable content of the bar file and send it to an execution group, first removing any existing deployed contents of the execution group.

When to use it

- If you want to deploy message flows, message sets or other deployable objects to an execution group.

When *not* to use it

- If you want to merge the existing contents of the execution group with the contents of the bar file. In this case, use an incremental bar file deployment instead.

Broker archive

The unit of deployment to the broker is the *broker archive* or *bar* file.

The bar file is a zipped (compressed) file which can contain a number of different files:

- A .cmf file for each message flow. This is a compiled version of the message flow. You can have any number of these files within your bar file.
- A broker.xml file. This file is called the *broker deployment descriptor*. You can have only one of these files within your bar file. This file, in XML format, resides in the META-INF folder of the zipped file and can be modified using a text editor or shell script.
- As a zipped file archive, the broker archive file can also contain any additional files you need. For example, you might want to include Java source files for future reference.

To deploy XML, XSL, and JAR files inside a broker archive, the connected Configuration Manager and target broker must be Version 6.0 or later.

Configurable properties of a broker archive

System objects defined in message flows can have configurable properties that you can update within the broker archive (bar) file before deployment. Configurable properties allow an administrator to update target-dependent properties, such as queue names, queue manager names, and database connections.

By changing configurable properties, you can customize a bar file for a new domain, for example a test system, without needing to edit and rebuild the message flows. Any properties that you define are contained within the deployment descriptor, META-INF/broker.xml. The deployment descriptor is parsed when the bar file is deployed.

| Edit the configurable properties either using the Broker Archive editor or by using
| the mqsiapplybaroverride command from a command prompt.

| Although the two methods indicated above are preferable, you can also edit the
| XML-format deployment descriptor manually using an external text editor or shell
| script.

Viewing version and keyword information for deployable objects

This topic contains information about how to view the version and keyword information of deployable objects.

- "Displaying object version in the bar file editor"
- "Displaying version, deploy time, and keywords of deployed objects"

Displaying object version in the bar file editor

A column in the bar editor called *Version* displays the version tag for all objects that have a defined version. These are:

- `.cmf` files

You cannot edit the *Version* column.

You can use the mqsireadbar command to list the keywords defined for each deployable file within a deployable archive file.

Displaying version, deploy time, and keywords of deployed objects

The Eclipse *Properties View* displays, for any deployed object:

- Version
- Deploy Time
- All defined keywords

For example, if you deploy a message flow with these literal strings:

- `$MQSI_VERSION=v1.0 MQSI$`
- `$MQSI Author=fred MQSI$`
- `$MQSI Subflow 1 Version=v1.3.2 MQSI$`

then the Properties View displays:

Deployment Time	Date and time of deployment
Modification Time	Date and time of modification
Version	`v1.0`
Author	`fred`
Subflow 1 Version	`v1.3.2`

You are given a reason if the keyword information is not available. For example, if keyword resolution has not been enabled at deploy time, the Properties View displays the message `Deployed with keyword search disabled`. Also, if you deploy to a Configuration Manager that is an earlier version than Version 6.0, the properties view displays `Keywords not available on this Configuration Manager`.

Broker configuration deployment

A broker configuration deployment informs a broker of various configuration settings, including the list of execution groups, multicast and inter-broker settings.

When to use it

- If you have modified multicast or inter-broker settings in the Message Brokers Toolkit or in a Configuration Manager Proxy application.

When *not* to use it

- If you have used the mqsichangeproperties command to change publish/subscribe settings directly on the broker component. In this case, a broker configuration deployment overwrites any changes you have made to the settings.
- If you are adding execution groups. In this case, the first time you deploy a broker archive (bar) file, the execution group is automatically initialized.

Publish/subscribe topology deployment

Topology deployment is primarily only required when using publish/subscribe. It informs each broker in the domain of the brokers with which it can share publications and subscriptions.

There are two ways of deploying a topology configuration:

- Complete topology deployment, in which all brokers are told of their neighboring publish/subscribe brokers.
- Delta topology deployment, in which only changes to the publish/subscribe topology are deployed. Such changes are deployed only to those brokers whose neighbor lists have changed since the last successful topology deployment.

Complete topology deployment

Deploying a complete topology:

- Tells each broker in the domain the set of brokers with which it can share publish/subscribe information.
- Forces the Configuration Manager to re-subscribe to the broker's status topics, such as start and stop messages.

Note: Whatever type of deployment you perform, the Configuration Manager attempts to subscribe to the broker's status messages if it is the first deployment to the broker. But only deploying the complete topology initiates a re-subscription.

When to use it

- If the Configuration Manager is not correctly reporting whether it is in a stopped or started state.
- If you have moved a Configuration Manager from one queue manager to another.
- If a broker's publish/subscribe function has become inconsistent. An example of inconsistency would be, for example, if one broker is able to share publications with a second broker, but not the other way round.

When *not* to use it

- If you are adding brokers to the domain and you are not using publish/subscribe. That is, if you are not connecting brokers together so that they can share publications and subscriptions.
- If you are adding execution groups to a broker.
- If you have changed the publish/subscribe network. In this case, deploy a delta topology, if possible, so that you only deploy to those brokers affected by the changes you have made.
- If you have removed a broker from the domain.

Delta topology deployment

Deploying a delta topology sends updated publish/subscribe network information to any broker with a publish/subscribe configuration that the Configuration Manager determines not to be current.

When to use it

- If you have modified a publish/subscribe network.
- If you are using the workbench to remove a broker from the domain, the Configuration Manager automatically requests the broker component to stop message flows that are running and to tidy up any resources being used. If this operation fails for any reason, you can again request the broker to tidy up. Deploying a delta topology is the most convenient way.

When *not* to use it

- If you are adding brokers to the domain and you are not using publish/subscribe. That is, if you are not connecting brokers together so that they can share publications and subscriptions.
- If you are adding or removing execution groups.

Publish/subscribe topics hierarchy deployment

If you are using publish/subscribe, deploy the topics hierarchy in these situations:

- If you have modified the hierarchy of topics. The deployment communicates the new hierarchy to each broker.
- If you have added a broker to the domain and you want it to use the existing topics hierarchy. The deployment communicates the hierarchy to the new broker.

There are two ways of deploying a publish/subscribe topics hierarchy:

- Complete deployment, in which the complete topics hierarchy is sent to all the brokers in a domain.
- Delta deployment, in which changes to the topics hierarchy (made since the last topics deployment) are sent to all the brokers in a domain.

Complete topics deployment

A complete topics deployment sends the publish/subscribe topics hierarchy to all the brokers in a domain.

When to use it

- If you have made changes to the topics hierarchy and one of the brokers has an inconsistent view of the expected topics hierarchy.
- If you have added a new broker to the domain that uses the topics hierarchy.

When *not* to use it

- If you have changed the topics hierarchy. In this case, a delta topics deployment is usually sufficient.

Delta topics deployment

A delta topics deployment sends the publish/subscribe topics hierarchy to all the brokers in a domain.

When to use it

- If you have made changes to the topics hierarchy.

When *not* to use it

- If the topics hierarchy has not changed.

Cancel deployment

The Configuration Manager allows only one deployment to be in progress to each broker at any one time. If for some reason a broker does not respond to a deployment request, subsequent requests cannot reach the broker, because, to the Configuration Manager, a deployment is still in progress.

Canceling deployment tells the Configuration Manager to assume that a broker will never respond to an outstanding deploy. In most cases, the action does *not* remove any deployment messages that have been sent to the broker, nor does it alter the running configuration of the broker. (Thus, for any brokers that *have* successfully deployed a configuration, the deployed information remains on the broker.)

If a broker subsequently *does* provide a response to an outstanding deployment that has been canceled, the response is ignored by the Configuration Manager and there is then an inconsistency between what is running on the broker and the information that is provided by the Configuration Manager.

Because of this risk of inconsistency, only cancel a deployment as a last resort, and only if you are sure that a broker will never be able to process a previous deployment request. However, before canceling deployment, you can manually remove any outstanding deployment messages to ensure that they are not processed.

When canceling deployment across the domain, the locks for all outstanding deploys in the domain are removed. When canceling deployment for a specific broker, only the lock for that broker is removed.

Canceling deployment is the equivalent of the 'force deploy' action in previous versions, except that cancel does not redeploy domain information.

You can cancel a deployment:
- To all the brokers in a domain
- To a single broker

Cancel deployment to a domain

Canceling the deployment to a domain tells the Configuration Manager to assume that all brokers in the domain that have outstanding deployments will not respond. If a broker then does provide a response to an outstanding deploy that has been canceled, it will be ignored and there will be an inconsistency between what is running on the broker and the information that is provided by the Configuration Manager.

When applied to a domain, canceling deployment does not remove deployment messages that have been sent to the brokers, and does not change the brokers' running configuration.

When to use it

Cancel a domain deployment only if these two conditions are *both* met:
- You attempt a deployment and you receive error message BIP1510.
- Any of the brokers that have outstanding deployments are not responding.

When *not* to use it

- If a broker is simply taking a long time to respond to a deployment request. The broker might have been temporarily stopped, for example.
- If other users might be deploying to the domain at the same time.
- If only one broker is not responding, or a small number of brokers are not responding. In this case, cancel the broker deployment instead.

Cancel deployment to a broker

Canceling the deployment to a single broker tells the Configuration Manager to assume that a specific broker in the domain that has an outstanding deployment will not respond. If the broker then does provide a response to an outstanding deploy that has been canceled, it will be ignored and there will be an inconsistency between what is running on the broker and the information that is provided by the Configuration Manager.

When applied to an individual broker, canceling deployment causes the Configuration Manager to attempt to remove from the broker, deployment messages that have not yet been processed. This only succeeds if the broker and the Configuration Manager share the same queue manager and if the message has not already been processed by the broker.

When to use it

Cancel a domain deployment only if these two conditions are *both* met:
- You attempt a deployment and you receive error message BIP1510.
- The broker is not responding.

When *not* to use it

- If the broker is simply taking a long time to respond to a deployment request. The broker might have been temporarily stopped, for example.
- If the version of the connected Configuration Manager is earlier than Version 6.0. In this case, canceling deployment to a specific broker has no effect; you must cancel the entire domain deployment instead.

Deploying a message flow application

Before you start:

Before you can deploy a message flow application, you must have created, and started, a Configuration Manager. The broker added to the domain is actually a reference, so you must also create and start the physical broker on the target system. A WebSphere MQ listener must be running and you must also have created a domain, added a broker to that domain, and created an execution group within the broker. See the links to related tasks below for help with these.

Message flow applications are deployed to execution groups by adding required resources, optionally with their source files, to a broker archive (bar) file. The bar file is then sent to the appropriate Configuration Manager where it is unpacked and the individual files deployed to execution groups on individual brokers. The Message flow application deployment topic gives more details.

The tasks in this section describe the process:
1. Creating a server project
2. Creating a broker archive
3. Adding files to a broker archive
   - Editing a broker archive file manually

- Editing configurable properties
4. Refreshing the contents of a broker archive
5. Deploying a broker archive
6. Checking the results of deployment

"Viewing version and keyword information for deployable objects" on page 88 describes how to view version and keyword information about deployed objects.

Creating a server project

Before you can deploy a message flow application, you must create a server project for it.

Before you start:

Save your message flow projects.

Follow these steps to create a server project using the Message Brokers Toolkit.
1. Switch to the Broker Administration perspective.
2. Click **File** → **New** → **Other**.
3. Select **Show all wizards**, then in the list of wizards, expand **Server** and click **Server Project**.
4. Click **Next**.
5. If asked, click **OK** to enable ″Base J2EE Support″.
6. Enter the name of your new server project.
7. Click **Finish**.

The folder that is created appears twice in the Navigator view (if **Show empty projects in Navigators** has been selected in the **Broker Administration Preferences** page):
- in the Domain Connections folder
- in the Broker Archives folder

Next:

Continue by creating a broker archive (bar) and adding files to it.

Creating a broker archive

Create a separate broker archive (bar) file for each configuration that you want to deploy.

There are two ways of creating a bar file:
- Using the Message Brokers Toolkit
- Using the mqsicreatebar command

Using the Message Brokers Toolkit

Before you start:

Either create a server project, or ensure that one already exixts.

Follow these steps to create a bar file using the workbench:

1. From the Broker Administration perspective, click **File** → **New** → **Message Broker Archive**.
2. Enter the name of your server project, or select one from the displayed list.
3. Enter a name for the bar file that you are creating.
4. Click **Finish**.

A file with a .bar extension is created and is displayed in the Broker Administration Navigator view, under the Broker Archives folder. The Content editor for the bar file opens.

Next:

Continue by adding files to your broker archive and then deploying it.

Using the mqsicreatebar command

Follow these steps to create a bar file using the mqsicreatebar command:
1. Open a command window that is configured for your environment.
2. Enter the command, typed on a single line, using this as an example:
   ```
   mqsicreatebar -b barName -p projectNames -o filePath
   ```
   A file with a .bar extension is created.

   The -b (bar file name), and -o (path for included files) parameters must be specified. The -p (project names) parameter is optional. The mqsicreatebar topic gives more details.

Next:

Continue by adding files to your broker archive and then deploying it.

Adding files to a broker archive

To deploy files to an execution group, you first include them in a broker archive (bar). The bar file is deployed by sending it to the Configuration Manager and from there, its contents are sent to the execution group on a broker.

You can only add message flows at the project level. However, after you have added the project to the bar file, you can use the **Remove** icon to remove individual message flows or message definitions, if required. Likewise, if you check the **Include message flow source** box, the source for all the message flows in the project are included but you can manually remove the source files for the message flows that you do not want.

To deploy XML, XSL, and JAR files inside a broker archive, the connected Configuration Manager and target broker must be Version 6.0 or later.

If there is a parent flow and subflow displayed in the **Add** dialogue, subflows are added automatically, so you only have to add the parent flow.

It is not possible to read deployed files back from broker execution groups. Therefore, keep a copy of the deployed bar file, or of the individual files within it.

Follow these steps to add files to a broker archive (bar) file using the Message Brokers Toolkit:
1. Switch to the Broker Administration perspective.

2. Double-click your bar file in the Broker Administration Navigator view to open it. The contents of the bar file are shown in the Content editor. (If the bar file is new, this view is empty.)
3. Click the **Add** icon.
4. Check the boxes for the message flows and other files that you want to include. (Duplicates within a bar file are automatically removed.)
5. Optional: If you want to include your message flow source files, check the **Include message flow source** box.
6. Optional: If you want to compile ESQL so that it is compatible with Version 2.1 brokers, check the **Compile ESQL for brokers version 2.1** box.
7. Click **OK**.

A list of the files that are now in your bar file is displayed in the Content editor. You can choose not to display your message flow source files by clearing the **Show source files** box at the bottom of the Content editor pane.

Next:

The next step is to deploy your broker archive (bar) file, but you might first want to edit configurable properties. You can also edit the contents of your bar file manually.

Editing a broker archive file manually

Before you start:

This task explains how to manually edit a broker archive (bar) file that already exists. If you have not already created a bar file, create it now, before continuing.

Follow these steps to edit a bar file manually using the Message Brokers Toolkit:

1. Export the bar file.
   a. From the workbench, click **File** → **Export**. The Export window appears.
   b. Select the export destination, such as a compressed file with .zip extension, and click **Next**.
   c. Select the resources that you want to export and click **Next**.
   d. Complete the destination information and click **Finish**. The file appears at the destination you specified as a compressed file.
2. Extract files from the bar file.
3. Edit the properties that you want to change in an editor of your choice.
4. Save the file.
5. Import the bar file back into the workbench to deploy it.
   a. From the workbench, click **File** → **Import**. The Import window appears.
   b. Select **Zip file** from the list.
   c. Click **Next**.
   d. Specify the name and location of your bar file.
   e. Select the server project that you want to contain the bar file.
   f. Click **Finish**.

Next:

Continue by deploying your broker archive (bar) file.

Editing configurable properties

You can edit the configurable properties of your broker archive (bar) file deployment descriptor.

Before you start:

If you have not already created a bar file, create it now, before continuing.

There are two ways of editing configurable properties:
- Using the Message Brokers Toolkit
- Using the mqsiapplybaroverride command

Using the Message Brokers Toolkit:

Follow these steps to edit properties using the Message Brokers Toolkit:
1. Switch to the Broker Administration perspective.
2. Select the **Configure** tab at the bottom of the Content editor pane. The properties that you can configure are listed.
3. Click the property for which you want to edit the value. The values that can be edited are displayed.
4. Replace the current value with the new value.
5. Save your bar file.

Next:

Continue by deploying your broker archive (bar) file.

Using the mqsiapplybaroverride command:

Follow these steps to edit properties using the mqsiapplybaroverride command:
1. Open a command window that is configured for your environment.
2. Create a text file (with a .properties file extension) following the format in the example below.
3. Enter the command, typed on a single line, specifying the location of your bar deployment descriptor (typically broker.xml) and the file containing the properties to be changed:
   ```
   mqsiapplybaroverride -b barName -p propertiesNames
   ```
   A file with a .bar extension is created.

Example

In this example, the bar deployment descriptor file contains the following information (elements to be changed are highlighted):

```
<?xml version="1.0" encoding="UTF-8"?>
<Broker>
   <CompiledMessageFlow name="sampleFlow">
      <ConfigurableProperty uri="sampleFlow#additionalInstances" />
      <ConfigurableProperty uri="sampleFlow#commitCount" />
      <ConfigurableProperty uri="sampleFlow#commitInterval" />
      <ConfigurableProperty uri="sampleFlow#coordinatedTransaction" />
      <ConfigurableProperty override="LOCAL_QUEUE_MGR"
         uri="sampleFlow#MQOutput.queueManagerName" />
      <ConfigurableProperty override="INPUT_QUEUE"
         uri="sampleFlow#MQInput.queueName" />
```

```
      <ConfigurableProperty override="REPLY_TO_QUEUE1"
         uri="sampleFlow#MQOutput.replyToQ" />
      <ConfigurableProperty uri="sampleFlow#MQInput.topicProperty" />
      <ConfigurableProperty uri="sampleFlow#MQOutput.replyToQMgr" />
      <ConfigurableProperty uri="sampleFlow#MQInput.validateMaster" />
      <ConfigurableProperty
         uri="sampleFlow#MQInput.serializationToken" />
      <ConfigurableProperty uri="sampleFlow#MQOutput.validateMaster" />
      <ConfigurableProperty override="OUTPUT_QUEUE"
         uri="sampleFlow#MQOutput.queueName" />
   </CompiledMessageFlow>
</Broker>
```

The mqsiapplybaroverride command is run, specifying a property file containing this information (the first line is a comment):

```
#Old Value in broker.xml = New Value in broker.xml
LOCAL_QUEUE_MGR = UnitTestQmgr
INPUT_QUEUE = UnitTestInQueue1
REPLY_TO_QUEUE =
OUTPUT_QUEUE = unitTestOutQueue1
```

As a consequence, the bar deployment descriptor is modified to:

```
<?xml version="1.0" encoding="UTF-8"?>
<Broker>
   <CompiledMessageFlow name="sampleFlow">
      <ConfigurableProperty uri="sampleFlow#additionalInstances" />
      <ConfigurableProperty uri="sampleFlow#commitCount" />
      <ConfigurableProperty uri="sampleFlow#commitInterval" />
      <ConfigurableProperty uri="sampleFlow#coordinatedTransaction" />
      <ConfigurableProperty override="UnitTestQmgr"
         uri="sampleFlow#MQOutput.queueManagerName" />
      <ConfigurableProperty override="UnitTestInQueue1"
         uri="sampleFlow#MQInput.queueName" />
      <ConfigurableProperty uri="sampleFlow#MQOutput.replyToQ" />
      <ConfigurableProperty uri="sampleFlow#MQInput.topicProperty" />
      <ConfigurableProperty uri="sampleFlow#MQOutput.replyToQMgr" />
      <ConfigurableProperty uri="sampleFlow#MQInput.validateMaster" />
      <ConfigurableProperty
         uri="sampleFlow#MQInput.serializationToken" />
      <ConfigurableProperty uri="sampleFlow#MQOutput.validateMaster" />
      <ConfigurableProperty override="UnitTestOutQueue1"
         uri="sampleFlow#MQOutput.queueName" />
   </CompiledMessageFlow>
</Broker>
```

Note that in this example, REPLY_TO_QUEUE is unset as no replacement value was set in the properties file.

Next:

Continue by deploying your broker archive (bar) file.

Refreshing the contents of a broker archive

Before you start:

You should already have created a broker archive and have added resources to it. You are likely to have subsequently made changes to those resources, that you want reflected in the archive before you deploy it.

It is possible to refresh the contents of a broker archive by removing resources from it and, having made required changes, add them back again. However, you can alternatively use the Refresh option in the Broker Archive editor.

Follow these steps to refresh the contents of a broker achive:

1. Switch to the Broker Administration perspective.

   Broker archive (bar) files that need refreshing are shown with an 'out-of-synch' icon ( ) in the Navigator view. (The bar file is considered to be inconsistent when any of its deployable workspace files has a time stamp later than that of the bar file.)
2. Double-click your bar file in the Navigator view to open it.

   The contents of the bar file are shown in the Content editor. Icons indicate resources that are consistent ( ), and those that need to be refreshed ( ).
3. To refresh all the resources in the broker archive, click **Refresh** ( ).

   A dialog box opens, showing progress. When the operation is complete, click **Details** to see information about what was refreshed, what was not, and why. If the refresh process was successful, you see the same information that is placed in the user log by each of the resource compilers.

   Alternatively, you can refresh the achive contents by right clicking a bar file in the Navigator view and selecting **Refresh Archive Contents**. The broker archive is rebuilt in the background.

   You can view, and clear ( ), the user and service logs by clicking the appropriate tabs in the Broker Archive editor.
4. (Optional) To view the properties of an individual resource in the Content editor, right click the resource and click **Show in Properties**.

   The Deployable properties view opens (if it is not already in the perspective) and shows details of the resource selected. The view has two fields:
   - Workspace Resource, with references to the linked workspace resources (.mset, .msgflow, .xml, and .xslt files, for example.
   - Last Compile Status, which shows the user log entry for the last compilation. The text can be copied, but cannot be modified.

Next:

Continue to deploy your bar file.

Deploying a broker archive file

Before you start:

This task explains how to deploy your broker archive (bar). If you have not already created a bar file, create it now, before continuing.

You can deploying a broker archive (bar) file in three ways:
- Using the Message Brokers Toolkit
- Using the mqsideploy command
- Using the Configuration Manager Proxy API

You need to have access rights if the execution group to which you want to deploy is restricted by an ACL.

Using the Message Brokers Toolkit

Follow these steps to deploy a bar file using the workbench:

1. Switch to the Broker Administration perspective.
2. Optional. Normally, an incremental bar file deployment is performed. If you want to perform a complete bar file deployment: right-click the target execution group in the Domains view and select **Remove Deployed Children**. Wait for the operation to complete before continuing.

   It is not necessary to remove deployed children if you only want to refresh one or more of them with the contents of the bar file. The difference between a complete and an incremental bar file deployment is explained in the Message flow application deployment topic.
3. Click the bar file shown in the Navigator view to highlight it.
4. Drag the file onto your target execution group shown in the Domains view.

   Alternatively, right-click the bar file and click **Deploy file**. A dialog box shows all the domains, as well as execution groups within those domains to which the workbench is connected. A dialog box shows the execution groups (within their domains) to which you can deploy the bar file. Select an execution group and click **OK** to deploy the bar file. (Note: If you select a broker topology that is not connected to a domain, an attempt is made to connect it. If you click **Cancel**, the broker topology remains unconnected to a domain.)

   Whichever method you use, you cannot select (and deploy to) more than one execution group at a time.
5. If the bar file has not been saved since it was last edited, you are asked whether you want to save it before deploying. If you click **Cancel**, the bar file is not saved and deployment does not take place.

The bar file is transferred to the Configuration Manager from where its contents (message flows, for example) are deployed to the execution group. In the Domains view, the assigned message flows are added to the appropriate execution group.

Next:

Continue by checking the results of the deployment.

Using the mqsideploy command

Follow these steps to deploy a bar file using the mqsideploy command:

1. Open a command window that is configured for your environment.
2. Using these as examples, enter the appropriate command, typed on a single line:

   z/OS On z/OS:

   ```
   /f MQ01CMGR,dp b=broker e=exngp a=barfile
   ```

   This performs an incremental deployment. Add the `m=yes` parameter to perform a complete bar file deployment.

   On other platforms:

   ```
   mqsideploy -i ipAddress -p port -q qmgr -b broker -e exngp -a barfile
   ```

   This performs an incremental deployment. Add the –m parameter to perform a complete bar file deployment.

   The -i (IP address), -p (port), and -q (queue manager) parameters represent the connection details of the queue manager workstation, and on the z/OS console, MQ01CMGR is the name of the Configuration Manager component.

The -b (broker name), -e (execution group name), and -a (bar file name) parameters (or z/OS equivalent) must also be specified.

Next:

Continue by checking the results of the deployment.

Using the Configuration Manager Proxy API

Use the deploy method of the ExecutionGroupProxy class. By default, the deploy method performs an incremental deployment. To perform a complete deployment, use a variant of the method that includes the boolean isIncremental parameter; setting this to false indicates a complete deployment. (Setting it to true indicates an incremental deployment.)

For example:

```
import com.ibm.broker.config.proxy.*;
import java.io.IOException;

public class DeployTopology {
  public static void main(String[] args) {
    ConfigManagerConnectionParameters cmcp = new MQConfigManagerConnectionParameters
   ("localhost", 1414, "QM1");
    try {
      ConfigManagerProxy cmp = ConfigManagerProxy.getInstance(cmcp);
      TopologyProxy t = cmp.getTopology();
      BrokerProxy b = t.getBrokerByName("BROKER1");
      ExecutionGroupProxy e = b.getExecutionGroupByName("default");
      e.deploy("deploy.bar");
    }
    catch (ConfigManagerProxyException cmpe) {
      cmpe.printStackTrace();
    }
    catch (IOException ioe) {
      ioe.printStackTrace();
    }
  }
}
```

Next:

Continue by checking the results of the deployment.

Deploying a broker configuration

The broker configuration deployment overview explains when you might want to deploy a broker configuration.

There are three ways to deploy a broker configuration:
- Using the Message Brokers Toolkit
- Using the mqsideploy command
- Using the Configuration Manager Proxy API

Using the Message Brokers Toolkit

If you modify any multicast or interbroker settings with the workbench, a broker configuration deployment is automatically initiated when the changes are applied.

Using the mqsideploy command

Follow these steps to deploy a broker configuration using the mqsideploy command:

1. Open a command window that is configured for your environment.
2. Using these as examples, enter the appropriate command, typed on a single line, specifying the broker to which you want to deploy:

   z/OS On z/OS:

   ```
   /f MQ01CMGR,dp b=broker
   ```

   On other platforms:

   ```
   mqsideploy -i ipAddress -p port -q qmgr -b broker
   ```

   The -i (IP address), -p (port), and -q (queue manager) parameters represent the connection details of the queue manager workstation, and on the z/OS console, MQ01CMGR is the name of the Configuration Manager component.

   By specifying the broker to which you want to deploy (b= or -b), without indicating a bar file (-a), the broker configuration is deployed rather than a message flow application.

Next:

Continue by checking the results of the deployment.

Using the Configuration Manager Proxy API

Use the deploy method of the BrokerProxy class.

For example:

```
import com.ibm.broker.config.proxy.*;

public class DeployBrokerConfig {
  public static void main(String[] args) {
    ConfigManagerConnectionParameters cmcp =
            new MQConfigManagerConnectionParameters
                ("localhost", 1414, "QM1");
    try {
      ConfigManagerProxy cmp = ConfigManagerProxy.getInstance(cmcp);
      TopologyProxy t = cmp.getTopology();
      BrokerProxy b = t.getBrokerByName("BROKER1");
      if (b != null) {
        b.deploy();
      }
    }
    catch (ConfigManagerProxyException e) {
      e.printStackTrace();
    }
  }
}
```

Next:

Continue by checking the results of the deployment.

Deploying a publish/subscribe topology

Before you start:

Make sure that you have configured your broker domain.

The publish/subscribe topology deployment overview explains when you might want to deploy a topology and the difference between a complete and delta deployment.

There are three ways to deploy topology information:
- Using the Message Brokers Toolkit
- Using the mqsideploy command
- Using the Configuration Manager Proxy API

You can configure the workbench preferences so that topology information is automatically deployed after a change.

After you have deployed a publish/subscribe topology, you might see an extra execution group process called $SYS_mqsi in a process listing, or in the output from the mqsilist command. When you deploy a publish/subscribe topology for the first time, a new execution group process is started on your broker to handle the publish/subscribe messages. This execution group is only used internally: it does not appear in the workbench and you cannot deploy message flows to it. After you have deployed one or more of your own flows to another execution group, $SYS_mqsi is removed when the broker is subsequently restarted.

Using the Message Brokers Toolkit

Follow these steps to deploy a topology configuration using the workbench:
1. Switch to the Broker Administration perspective.
2. In the Domains view, expand the Domains from where you want to perform the deploy.
3. Right-click **Broker Topology** hierarchy.
4. Click **Deploy Topology Configuration**.
5. Click **Delta** to deploy only the changed items, or click **Complete** to deploy the entire configuration.

   Alternatively, you can make a change to the Topology document in the Broker Administration perspective, save the changes and then select **Delta**. This behavior can be modified in the workbench preferences dialog.

The topology is deployed, and the Configuration Manager distributes it to the brokers in the domain.

Next:

Continue by checking the results of the deployment.

Using the mqsideploy command

Follow these steps to deploy a topology configuration using the mqsideploy command:

1. Open a command window that is configured for your environment.
2. Using these as examples, enter the appropriate command, typed on a single line:

   z/OS On z/OS:

   `/f MQ01CMGR,dp l=yes`

   This performs a delta deployment. Add the `m=yes` parameter to deploy the entire configuration.

   On other platforms:

   mqsideploy –i *ipAddress* –p *port* –q *qmgr* –l

   This performs a delta deployment. Add the –m parameter to deploy the entire configuration. The -i (IP address), -p (port), and -q (queue manager) parameters represent the connection details of the queue manager workstation, and on the z/OS console, MQ01CMGR is the name of the Configuration Manager component.

Next:

Continue by checking the results of the deployment.

Using the Configuration Manager Proxy API

Use the deploy method of the TopologyProxy class. By default, the deploy method performs a delta deployment. To deploy the complete hierarchy, use a variant of the method that includes the boolean isDelta parameter; setting this to false indicates a complete deployment. (Setting it to true indicates a delta deployment.)

For example:

```
import com.ibm.broker.config.proxy.*;

public class DeployTopology {
  public static void main(String[] args) {
    ConfigManagerConnectionParameters cmcp =
            new MQConfigManagerConnectionParameters
                ("localhost", 1414, "QM1");
    try {
      ConfigManagerProxy cmp =
            ConfigManagerProxy.getInstance(cmcp);
      TopologyProxy t = cmp.getTopology();
      t.deploy(false);
    }
    catch (ConfigManagerProxyException e) {
      e.printStackTrace();
    }
  }
}
```

Next:

Continue by checking the results of the deployment.

Deploying a publish/subscribe topics hierarchy

Before you start:

Make sure that you have configured your broker domain.

The topic deployment overview explains when you might want to deploy a topic hierarchy and the difference between a complete and delta deployment.

There are three ways to deploy a topics hierarchy:
- Using the Message Brokers Toolkit
- Using the mqsideploy command
- Using the Configuration Manager Proxy API

You can configure the workbench preferences so that a topics hierarchy is automatically deployed after a change.

Using the Message Brokers Toolkit

Follow these steps to deploy a topics hierarchy using the workbench:
1. Switch to the Broker Administration perspective.
2. In the Domains view, expand the Domains from where you want to perform the deploy.
3. Right-click **Topics** hierarchy.
4. Click **Deploy Topics Configuration**.
5. Click **Delta** to deploy only the changed items, or click **Complete** to deploy the entire configuration.

The topics hierarchy is deployed, and the Configuration Manager distributes the topics to brokers in the domain.

Next:

Continue by checking the results of the deployment.

Using the mqsideploy command

Follow these steps to deploy a topics hierarchy using the mqsideploy command:
1. Open a command window that is configured for your environment.
2. Using these as examples, enter the appropriate command, typed on a single line:

   z/OS On z/OS:
   ```
   /f MQ01CMGR,dp t=yes
   ```
   This performs a delta deployment. Add the `m=yes` parameter to deploy the entire configuration.

   On other platforms:
   ```
   mqsideploy -i ipAddress -p port -q qmgr -t
   ```
   This performs a delta deployment. Add the –m parameter to deploy the entire configuration. The -i (IP address), -p (port), and -q (queue manager) parameters represent the connection details of the queue manager workstation.

Next:

Continue by checking the results of the deployment.

Using the Configuration Manager Proxy API

Use the deploy method of the TopicRootProxy class. By default, the deploy method performs a delta deployment. To deploy the complete hierarchy, use a variant of the method that includes the boolean isDelta parameter; setting this to false indicates a complete deployment. (Setting it to true indicates a delta deployment.)

For example:

```
import com.ibm.broker.config.proxy.*;

public class DeployTopics {
  public static void main(String[] args) {
    ConfigManagerConnectionParameters cmcp =
            new MQConfigManagerConnectionParameters
                ("localhost", 1414, "QM1");
    try {
      ConfigManagerProxy cmp =
            ConfigManagerProxy.getInstance(cmcp);
      TopicRootProxy t = cmp.getTopicRoot();
      t.deploy(false);
    }
    catch (ConfigManagerProxyException e) {
      e.printStackTrace();
    }
  }
}
```

Next:

Continue by checking the results of the deployment.

Checking the results of deployment

After you have made a deployment, check that the operation has completed successfully. There are three ways of checking the results of a deployment:

- Using the Message Brokers Toolkit
- Using the mqsideploy command
- Using the Configuration Manager Proxy API

Also, check the system log on the target system where the broker was deployed to make sure that the broker has not reported any errors.

Using the Message Brokers Toolkit

Follow these steps to check a deployment using the workbench:

1. Switch to the Broker Administration perspective.
2. Expand the Domains view.
3. Double-click the **Event Log**.

When the deployment is initiated, an information message is displayed, confirming that the request was received by the Configuration Manager:

- BIP0892I

If the deployment completes successfully, you might also see one or more of these additional messages:

- BIP4040I

- BIP4045I
- BIP2056I

Using the mqsideploy command

The command returns numerical values from the Configuration Manager and any brokers affected by the deployment to indicate the outcome of the deployment. If it completes successfully, it returns 0. Refer to the mqsideploy topic for details of other values that you might see.

Using the Configuration Manager Proxy API

If you are using a Configuration Manager Proxy application, you can find out the result of a publish/subscribe topology deployment operation, for example, by using code similar to this:

```
TopologyProxy t = cmp.getTopology();

boolean isDelta = true;
long timeToWaitMs = 10000;
DeployResult dr = topology.deploy(isDelta, timeToWaitMs);

System.out.println("Overall result = "+dr.getCompletionCode());

// Display overall log messages
Enumeration logEntries = dr.getLogEntries();
while (logEntries.hasMoreElements()) {
  LogEntry le = (LogEntry)logEntries.nextElement();
  System.out.println("General message: " + le.getDetail());
}

// Display broker specific information
Enumeration e = dr.getDeployedBrokers();
while (e.hasMoreElements()) {

  // Discover the broker
  BrokerProxy b = (BrokerProxy)e.nextElement();

  // Completion code for broker
  System.out.println("Result for broker "+b+" = " +
    dr.getCompletionCodeForBroker(b));

  // Log entries for broker
  Enumeration e2 = dr.getLotEntriesForBroker(b);
  while (e2.hasMoreElements()) {
    LogEntry le = (LogEntry)e2.nextElement();
    System.out.println("Log message for broker " + b +
      le.getDetail()));
  }
}
```

The `deploy()` method blocks until all affected brokers have responded to the deployment request.

When the method returns, the DeployResult represents the outcome of the deployment at the time when the method returned; the object is not updated by the Configuration Manager Proxy.

If the deployment message could not be sent to the Configuration Manager, a `ConfigManagerProxyLoggedException` is thrown at the time of deployment. If the Configuration Manager receives the deployment message, then log messages for the overall deployment are displayed, followed by completion codes specific to each broker affected by the deployment. The completion code is one of the following static instances from the `com.ibm.broker.config.proxy.CompletionCodeType` class:

Completion code	**Description**
`pending`	The deploy is held in a batch and will not be sent until you issue *ConfigManagerProxy.sendUpdates()*.
`submitted`	The deploy message was sent to the Configuration Manager but no response was received before the timeout occurred.
`initiated`	The Configuration Manager replied stating that deployment has started, but no broker responses were received before the timeout occurred.
`successSoFar`	The Configuration Manager issued the deployment request and some, but not all, brokers responded with a success message before the timeout period expired. No brokers responded negatively.
`success`	The Configuration Manager issued the deployment request and all relevant brokers responded successfully before the timeout period expired.
`failure`	The Configuration Manager issued the deployment request and at least one broker responded negatively. You can use *getLogEntriesForBroker* for more information on why the deployment failed.
`notRequired`	A deployment request was submitted to the Configuration Manager involved with the supplied broker, but the request was not sent to the broker because its configuration is already up to date.

Canceling a deployment that is in progress

Before you start:

Canceling a deployment should only be a last resort if you are sure that a broker, or several brokers in a domain, will never be able to process a previous deployment request. For this reason, make sure that you understand the implications of this action, described in the Cancel deployment overview topic.

It is possible to cancel all outstanding deployments in the domain, or just those to a particular broker.

- When canceling deployment across the domain, you must have full access on the Configuration Manager.
- When canceling deployment to a specific broker, you must have full access on that broker.

If you want to ensure that previous deployment messages are not processed when an affected broker is restarted, first remove any deployment messages:

1. Stop the broker
2. Check the broker's SYSTEM.BROKER.ADMIN.QUEUE and SYSTEM.BROKER.EXECUTIONGROUP.QUEUE, and manually remove any deployment messages.
3. Proceed to cancel the deployment.

There are three ways to cancel a deployment:

- Using the Message Brokers Toolkit

- Using the mqsideploy command
- Using the Configuration Manager Proxy API

Using the Message Brokers Toolkit

Follow these steps to cancel the deployment to a particular broker or all outstanding deployments in a domain, using the workbench:

1. Switch to the Broker Administration perspective.
2. In the Domains view, right-click either a particular broker or a connected domain.
3. Click **Cancel Deployment**.

Deployments to the broker or domain are canceled.

Next:

Continue by checking the results. (A BIP0892I information message is displayed to show that the request was received by the Configuration Manager.)

Using the mqsideploy command

Follow these steps to cancel a deployment using the mqsideploy command:

1. Open a command window that is configured for your environment.
2. Using these as examples, enter the appropriate command, typed on a single line:

   z/OS On z/OS:

   ```
   /f MQ01CMGR,dp t=yes b=B1
   ```

   This cancels deployment to the broker called B1. Omit the b argument to cancel all outstanding deployments in the domain.

   On other platforms:

   ```
   mqsideploy -i ipAddress -p port -q qmgr –c –b B1
   ```

   This cancels deployment to the broker called B1. Omit the -b parameter to cancel all outstanding deployments in the domain. The -i (IP address), -p (port), and -q (queue manager) parameters represent the connection details of the queue manager workstation, and on the z/OS console, MQ01CMGR is the name of the Configuration Manager component.

Next:

Continue by checking the results. (A BIP0892I information message is displayed to show that the request was received by the Configuration Manager.)

Using the Configuration Manager Proxy API

To cancel all outstanding deployments in a domain, use the cancelDeployment method of the ConfigManagerProxy class. For example:

```
public class CancelAllDeploys {
  public static void main(String[] args) {
    ConfigManagerConnectionParameters cmcp =
            new MQConfigManagerConnectionParameters
                ("localhost", 1414, "QM1");
    try {
      ConfigManagerProxy cmp =
```

```
            ConfigManagerProxy.getInstance(cmcp);
      cmp.cancelDeployment();
    }
    catch (ConfigManagerProxyException e) {
      e.printStackTrace();
    }
  }
}
```

To cancel deployment to a specific broker in a domain, use the cancelDeployment method of the BrokerProxy class. For example, to cancel deployment to a broker called *B1*:

```
import com.ibm.broker.config.proxy.*;

public class CancelDeploy {
  public static void main(String[] args) {
    ConfigManagerConnectionParameters cmcp =
            new MQConfigManagerConnectionParameters
                ("localhost", 1414, "QM1");
    try {
      ConfigManagerProxy cmp =
            ConfigManagerProxy.getInstance(cmcp);
      TopologyProxy t = cmp.getTopology();
      BrokerProxy b = t.getBrokerByName("B1");
      b.cancelDeployment();
    }
    catch (ConfigManagerProxyException e) {
      e.printStackTrace();
    }
  }
}
```

Next:

Continue by checking the results. (A BIP0892I information message is displayed to show that the request was received by the Configuration Manager.)

Renaming objects that are deployed to execution groups

You cannot rename an object while it is still deployed to an execution group. Instead, first remove the deployed object from the execution group. Then, having renamed it, deploy it again.

Removing a deployed object from an execution group

There are three ways of removing deployed objects from an execution group:
- Using the Message Brokers Toolkit
- Using the mqsideploy command
- Using the Configuration Manager Proxy API

Using the Message Brokers Toolkit

Follow these steps to remove an object from an execution group using the workbench:
1. Switch to the Broker Administration perspective.
2. From the Domains view, right-click the object that you want to remove.
3. Click **Remove** from the pop-up menu, and **OK** to confirm.

An automatic deployment is performed for the updated broker and a BIP08921 information message is produced, confirming that the request was received by the Configuration Manager.

| A JAR file that is being used by a message flow cannot be deleted before the
| message flow that is using it has first been deleted. If more than one message flow
| is using the same JAR file, the all of these message flows must first be removed
| before the JAR file can be deleted.

Using the mqsideploy command

Follow these steps to remove an object from an execution group using the mqsideploy command:

1. Open a command window that is configured for your environment.
2. Using these as examples, enter the appropriate command, typed on a single line:

   z/OS On z/OS:

   ```
   /f MQ01CMGR,dp t=yes b=broker e=execgp d=file1.cmf:file2.dictionary:file3.xml
   ```

   On other platforms:

   ```
   mqsideploy -i ipAddress -p port -q qmgr –b broker –e execgp
                         –d file1.cmf:file2.dictionary:file3.xml
   ```

   Optionally, specify the -m option to clear the contents of the execution group. This tells the execution group to completely clear any existing data before the new bar file is deployed. The -i (IP address), -p (port), and -q (queue manager) parameters represent the connection details of the queue manager workstation, and on the z/OS console, MQ01CMGR is the name of the Configuration Manager component.

The -d argument (or d= argument on z/OS) is a colon separated list of files to be removed from the named execution group. Invoking the command above causes the deployed objects (file1.cmf, file2.dictionary and file3.xml) to be removed from the specified execution group and broker.

The command displays feedback as responses are received from the Configuration Manager and any brokers affected by the deployment. If the command completes successfully, it returns 0.

Using the Configuration Manager Proxy API

One way of removing deployed objects using the Configuration Manager Proxy API is to get a handle to the relevant `ExecutionGroupProxy` object and then invoke its `deleteDeployedObjectsByName()` method. For example:

```
import com.ibm.broker.config.proxy.*;

public class DeleteDeployedObjects {
  public static void main(String[] args) {
    ConfigManagerConnectionParameters cmcp =
            new MQConfigManagerConnectionParameters
                      ("localhost", 1414, "QM1");
    try {
      ConfigManagerProxy cmp =
            ConfigManagerProxy.getInstance(cmcp);
      TopologyProxy t = cmp.getTopology();
      BrokerProxy b = t.getBrokerByName("broker1");
      ExecutionGroupProxy e =
```

```
            b.getExecutionGroupByName("default");
      e.deleteDeployedObjectsByName(
            new String[] { "file1.cmf",
                           "file2.dictionary",
                           "file3.xml" }, 0);
    }
    catch (ConfigManagerProxyException e) {
      e.printStackTrace();
    }
  }
}
```

Part 3. Exploiting user-defined extensions

User-defined nodes

A user-defined node is a component that has been designed and implemented by WebSphere Message Broker users or by third-party vendors to add to the function of your implementation of WebSphere Event Broker.

With WebSphere Event Broker, you can deploy the following types of user-defined extensions:
- User-defined input nodes
- User-defined message processing nodes
- User-defined output nodes

User-defined nodes can be used in conjunction with the nodes that are supplied with the product, and with third-party supplied nodes. They can interact with the other nodes in the message flow, and can have characteristics such as rollback, commit, accessing external databases, and accessing WebSphere MQ queues.

You can configure the terminals and properties on your user-defined nodes, according to your system setup. However, you cannot change any of the internals of a user-defined node.

User-defined nodes can be written in the C or Java programming language. User-defined nodes written in C are compiled into a loadable implementation library, that is, a shared library on Linux and UNIX, or a Windows DLL. User-defined nodes written in Java are packaged as a jar file.

For information on deploying or deleting user-defined nodes, see the relevant topics in this section of the help.

Installing a user-defined node on a broker domain

Before you start

You must have a compiled user-defined extension, which has been supplied either by a third party vendor, or by a WebSphere Message Broker Version 6.0 user.

1. Put a copy of your compiled or packaged user-defined extension file on every broker system from which you intend to use it.

   Specify the directory in which to put the file, by using either the mqsichangebroker command or the mqsicreatebroker command.

   Note: Do not save the .lil or .jar file in the WebSphere Event Broker install directory.

   For C user-defined extensions, it is recommended that the .pdb file, which corresponds to the .lil file, is also stored in the chosen directory. The .pdb file provides symbolic information that is used by WebSphere Event Broker when displaying stack diagnostic information in the event of access violations or other software malfunctions.
2. Stop and start each broker. This is to ensure that the existence of a new file is detected.

   There are two situations where a broker restart is not necessary:

   - If you have created an execution group in the Toolkit, and there is nothing yet deployed to it, you can add the .lil, .pdb, and .jar file to your chosen directory.
   - If something has already been deployed to the execution group you that want to use, add the .lil, .pdb, and .jar file to your chosen directory and then use the mqsireload command to restart the group. It is not possible to overwrite an existing file on the Windows platform when the broker is running because of the file lock that is put in place by the operating system.

   These two situations should be used with caution because any execution group that is connected to the same broker will also detect the new .lil, .pdb, and .jar files when that execution group is restarted, or when something is first deployed to that execution group. By using the more conventional way of restarting the broker, you ensure that anyone with an interest in a particular execution group is made aware that recent changes have been made to the broker.

   These two situations assume that you have already completed the previous step, and have therefore used either the mqsichangebroker command or the mqsicreatebroker command to notify the broker of the directory in which the user-defined extension files have been placed.

   When you have installed a user-defined node, it is referred to by its schema and name, just like a message flow.

Deleting a user-defined node

Before you start

You must have at least one user-defined extension installed on your broker system.

On all types of system, you can remove a user-defined node file from the broker by completing the following steps:

1. Stop the broker, using the mqsistop command.
2. Delete the .lil or .jar file from the appropriate directory.
3. Restart the broker using the mqsistart command.

Part 4. Reference

Message flows

Message flow reference information is available for:

- "Message flow preferences"
- "Description properties for a message flow"
- "Built-in nodes" on page 126
- "User-defined nodes" on page 183
- "Supported code pages" on page 183
- "Data integrity within message flows" on page 211
- "Configurable message flow properties" on page 211
- "Message flow porting considerations" on page 212
- "Message flow accounting and statistics data" on page 213

Message flow preferences

You can set Message flow preferences from **Window** → **Preferences** then click **Message Flow** in the left pane.

Property	Type	Meaning
Default version tag	String	Provide the default version information you would like to be set in the message flow Version property when you create a new message flow.

Description properties for a message flow

Property	Type	Meaning
Version	String	You can enter a version for the message flow in this field. This allows the version of the message flow to be displayed using the Eclipse properties view. A default for this field can be set in the messages flow preferences.
Short Description	String	You can enter a short description of the message flow in this field.
Long Description	String	You can add information to enhance the understanding of the message flow's function in this field. It is a string field and any standard alphanumeric characters can be used. You can also use this field to define a keyword and its value that will display for the deployed message flow in the properties view of Eclipse. An example is: `$MQSI Author=Fred MQSI$` When the properties of the deployed message flow are displayed, this will add a row to the display showing "Author" as the property name and "Fred" as its value. For information on keywords see "Guidance for defining keywords" on page 124.

To view and edit the properties of a message flow click **Flow** → **Properties**.

Guidance for defining keywords

This topic contains the rules to follow when defining keywords. Keywords and their values are displayed in the properties view of a deployed object.

A number of objects in WebSphere Event Broker can have additional information added to the object. This information can display information about an object after the object has been deployed. The default information that is displayed is the time the object was deployed and the last time the object was modified.

You can define custom keywords, and their values that the Configuration Manager will interpret as additional information to be displayed, in the properties view. For example, you can define keywords for “Author” and “Subflow 1 Version”:

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

The information the Configuration Manager shows is:

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

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

The syntax for defining a keyword and its associated value is:

```
$MQSI KeywordName = KeywordValue MQSI$
```

Where:

$MQSI
: $MQSI opens the definition. It can be followed by an optional underscore or white space character that is ignored.

KeywordName
: The name of the keyword for which you are setting the value. It can be made up of any sequence of alphanumeric characters apart from the equals (=) sign. It can contain white space characters, but any leading or trailing white space characters are omitted.

= The equals (=) sign is the delimiter between the keyword and the value that you are setting it to.

KeywordValue
: The value to which the keyword is set. It can be made up of any sequence of alphanumeric characters. It can contain white space characters, but any leading or trailing white space characters are omitted.

MQSI$
: MQSI$ closes the keyword definition.

Examples

Example definitions	Interpreted keyword and value	Comments

$MQSIAuthor=JohnMQSI$ or
$MQSI Author=John MQSI$ or
$MQSI Author = John MQSI$
	Keyword = ″Author″
	Value = ″John″
		Each of these is a basic example of what can be set and shows that the leading and trailing white space characters for the name and value parameters are ignored.

$MQSI_Author = John MQSI$	Keyword = ″Author″
	Value = ″John″
		The first character after $MQSI can be an underscore character. The underscore character is omitted in the interpreted keyword. If a second underscore character appears, this forms part of the keyword name.

$MQSI Flow designer = John Smith MQSI$	Keyword = ″Flow designer″
	Value = ″John Smith″
		White space characters are accepted for each parameter value.

$MQSI bar = MQSI$	Keyword = ″bar″
	Value = ″″
		The keyword value can be set to an empty (″″) string.

$MQSI_mqsitag=$MQSI$MQSI$	Keyword = ″mqsitag″
	Value = ″$″
		This is a poorly formatted definition. After defining the keyword name, the parser is looking to find the delimiters that form the boundary of the value to be set. In this case, the only character prior to the MQSI$ that closes the definition is a ’$’, and that is set as the keyword value.

$MQSI=barMQSI$		This pattern is ignored because the keyword name cannot be an empty string.
$MQSItagbarMQSI$		This pattern is ignored because there is not a separator (=) between the keyword name and the keyword value.

Do not use the following keywords as described below:

VERSION
When you use the Message Brokers Toolkit to edit message flows and dictionaries, it is possible to set the **Version** property in the Properties pane, which you can then view in the Broker Archive file editor. If you set this property, a keyword called VERSION is added to the resulting cmf or dictionary file. For this reason, do not add `$MQSI_VERSION=...MQSI$` to these files.

BAR The BAR keyword is associated with each object automatically when it is deployed and it contains the full path name of the broker archive file that deployed the object.

The values of both keywords are defined programmatically in the class `com.ibm.broker.config.proxy.DeployedObject`.

Restrictions within keywords

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

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

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

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

Built-in nodes

WebSphere Event Broker supplies built-in nodes that you can use to define your message flows. For information about each of these nodes, follow the appropriate link below. The nodes are listed in the categories under which they are grouped in the node palette.

WebSphere MQ
- "MQInput node" on page 154
- "MQOutput node" on page 164
- "MQOptimizedFlow node" on page 163
- "MQeInput node" on page 145
- "MQeOutput node" on page 151

JMS
- "JMSInput node" on page 128
- "JMSOutput node" on page 137
- "JMSMQTransform node" on page 135
- "MQJMSTransform node" on page 161

Routing
- "Publication node" on page 171

Construction
- "Input node"
- "Output node" on page 169

Additional protocols
- "SCADAInput node" on page 177
- "SCADAOutput node" on page 181
- "Real-timeInput node" on page 173
- "Real-timeOptimizedFlow node" on page 175

Input node

This topic contains the following sections:
- "Purpose"
- "Configuring the Input node" on page 127
- "Terminals and properties" on page 127

Purpose

The Input node provides an In terminal for an embedded message flow (a subflow). You can use a subflow for a common task that can be represented by a sequence of message flow nodes. For example, you can create a subflow to increment or decrement a loop counter, or to provide error processing that is common to a number of message flows.

You must use an Input node to provide the In terminal to a subflow; you cannot use a standard input node (a built-in input node such as MQInput , or a user-defined input node).

When you have started your subflow with an Input node, you can connect it to any In terminal on any message flow node, including an Output node.

You can include one or more Input nodes in a subflow. Each Input node that you include provides a terminal through which to introduce messages to the subflow. If you include more than one Input node, you cannot predict the order in which the messages are processed through the subflow.

The Input node is represented in the workbench by the following icon:

When you select and include a subflow in a message flow, it is represented by the icon:

When you include the subflow in a message flow, this icon shows a terminal for each Input node that you include in the subflow, and the name of the terminal (which you can see when you hover over it) matches the name of that instance of the Input node. Give your Input nodes meaningful names that you can recognize easily when you use their corresponding terminal on the subflow node in your message flow.

Configuring the Input node

When you have put an instance of the Input node into a message flow, you can configure it. To display its properties, either double-click the node, or right-click the node and click **Properties**.

Optional: On the Description tab, enter a short description, a long description, or both. You can also rename the node on this tab.

Terminals and properties

The Input node terminals are described in the following table.

Terminal	Description
Out	The input terminal that delivers a message to the subflow.

The following table describes the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the bar file to deploy it).

The Input node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, e.g. Input.	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

JMSInput node

This topic contains the following sections:
- "Purpose"
- "Using the JMSInput node in a message flow"
- "Making the JMS Provider client available to the JMS nodes" on page 129
- "Configuring the JMSInput node" on page 129
- "Terminals and properties" on page 134

Purpose

Use the JMSInput node to receive messages from JMS destinations. JMS destinations are accessed through a connection to a JMS provider. The JMSInput node acts as a JMS message consumer and can receive all six message types that are defined in the Java Message Service Specification, version 1.1. Messages are received by using method calls, which are described in the JMS specification.

The JMSInput node is represented in the workbench by the following icon:

Using the JMSInput node in a message flow

The JMS Nodes sample contains a message flow in which the JMSInput node is used; refer to this sample for an example of how to use the JMSInput node.

Message flows that handle messages that are received from connections to JMS providers, must always start with a JMSInput node. If you include an output node in a message flow that starts with a JMSInput node, it can be any of the supported output nodes (including user-defined output nodes); you do not need to include a JMSOutput node. However, if you do not include a JMSOutput node, you must include the JMSMQTransform node to transform the message to the format that is expected by the output node.

If you are propagating JMS messages and creating a message flow to use as a subflow, you cannot use a standard input node; you must use an instance of the JMSInput node as the first node in order to create an In terminal for the subflow.

Restriction: When the JMSInput node receives publication topics, it internally restricts the message flow property *Additional Instances* to zero to prevent the receipt of duplicate publications.

Making the JMS Provider client available to the JMS nodes

For distributed systems, copy the Java .jar files and any native libraries for the JMS provider client into the broker shared-classes directory; for example, `C:\Documents and Settings\All Users\Application Data\IBM\MQSI\shared-classes` on Windows. Copying the files into the shared-classes directory ensures that the Java class path for the JMS nodes is set correctly.

For z/OS, there is no shared-classes directory; instead, perform the following steps:

1. Specify each JMS provider Java .jar file in the class path in the BIPPROF member of the broker's Partitioned Data Set (PDS).
2. Update the LIBPATH with any native libraries.
3. Submit the BIPGEN JCL job to update the broker ENVFILE.

Configuring the JMSInput node

When you have put an instance of the JMSInput node into a message flow, you can configure it. To display its properties, either double-click the node, or right-click the node and click **Properties**.

All mandatory properties that do not have a default value defined are marked with an asterisk.

Configure the JMSInput node as follows:

1. Optional: On the Description tab, enter a short description, a long description, or both. You can also rename the node on this tab.
2. On the JMS Connection tab, set the following properties:
   - Enter an *Initial Context Factory* value. A JMS application uses the initial context to obtain and look up the JNDI administered objects for the JMS provider. The default value is

     `com.sun.jndi.fscontext.RefFSContextFactory`, which defines the file-based initial context factory for the WebSphere MQ JMS provider.

     To identify the name of the Initial Context Factory for the JMS provider, refer to the JMS provider documentation.
   - Enter a value for the *Location JNDI Bindings*. This value specifies either the file system path or the LDAP location for the bindings file. The bindings file contains definitions for the JNDI administered objects that are used by the JMSInput node.

     When you enter a value for *Location JNDI Bindings*, ensure that it complies with the following instructions:
     – Construct the bindings file before you deploy a message flow that contains a JMSInput node.
     – Do not include the filename of the bindings file in this field.
     – If you have specified an LDAP location that requires authentication, configure the LDAP principal (userid) and LDAP credentials (password) separately. These values are configured at broker level. For information about configuring these values, refer to the mqsicreatebroker and mqsichangebroker commands.
     – The string value must include the leading keyword, which is one of:
       - `file:/`
       - `iiop:/`
       - `ldap:/`

For information about constructing the JNDI administered objects bindings file, refer to the documentation that is supplied with the JMS provider.

- Enter a *Connection Factory Name*. The connection factory name is used by the JMSInput node to create a connection to the JMS provider. This name must already exist in the bindings file.
- Enter a *Backout Destination* name. Input messages are sent to this destination when errors prevent the message flow from processing the message, and the message must be removed from the input destination. The backout destination name must exist in the bindings file.
- Enter a value for the *Backout Threshold*. This value determines when an input message is put to the *Backout Destination*. For example, if the value is `3`, the JMS provider attempts to deliver the message to the input destination three times. After the third attempted delivery, the message is removed from the input destination and is sent to the backout destination. The default value is `0`.

3. The JMSInput node has two modes, which are mutually exclusive. On the Basic tab, select the mode for the JMSInput node:
   - If the node is to read from a queue (point-to-point), select *Source Queue* and enter the name of the source queue, which is the JMS queue that is listed in the bindings file. This property is mutually exclusive with *Subscription Topic* on the Basic tab.
   - If the node is to read from a subscription topic (publish/subscribe), select *Subscription Topic* and enter the name of the subscription topic.
     - If you select *Subscription Topic*, the node operates in the publish/subscribe message domain only.
     - This property is mutually exclusive with *Source Queue* on the Basic tab.
     - The *Subscription Topic* name must conform to the standards of the JMS provider that is being used by the node.
   - If the node is to receive publications from a durable subscription topic, enter a *Durable Subscription ID*.
     - Removing a durable subscription is a separate administration task. For information about removing a durable subscription, refer to the JMS provider documentation.
     - This property is valid only when a *Subscription Topic* string has been specified.
4. On the Input Message Parsing tab, set values for the properties that describe the message domain, message set, message type, and message format:
   - If you are using the MRM or IDoc parser, select the correct message set from the drop-down list in *Message Set*. This list is populated with available message sets when you select MRM or IDoc as the domain.

     Leave *Message Set* blank for XML, XMLNS, JMS, , and BLOB parsers.
   - If you are using the MRM parser, select the correct message from the drop-down list in *Message Type*. This list is populated with messages that are defined in the message set that you have selected.

     Leave *Message Type* blank for XML, XMLNS, JMS, , BLOB, and parsers.
   - If you are using the MRM or IDoc parser, select the format of the message from the drop-down list in *Message Format*. This list includes all the physical formats that you have defined for this message set.

     Leave *Message Format* blank for XML, XMLNS, JMSMap, JMSStream, , and BLOB parsers.
5. If you need to filter messages, set the properties on the Message Selectors tab:

- If the JMS provider is required to filter messages, based on message properties that are set by the originating JMS client application, enter a value for *Application Property*, specifying both the property name and the selection conditions; for example, `OrderValue > 200`.

  Leave this property blank if you do not want the input node to make a selection based upon application property. Refer to JMS message selectors for a description of how to construct the message selector.
- If the JMS provider is required to filter messages that have been generated at specific times, enter a value for *Timestamp*, where the value is an unqualified Java millisecond time; for example, `105757642321`. Qualify the selector with operators, such as `BETWEEN` or `AND`.

  Leave this property blank if you do not want the input node to make a selection based on JMSTimeStamp.
- If the JMS provider is required to filter messages based on the JMSDeliveryMode header value in the JMS messages, select an option for *Delivery Mode* from the drop-down list. Choose from:
  - *Non Persistent* (the default) to receive messages that are marked as non persistent by the originating JMS client application.
  - *Persistent* to receive messages that are marked as persistent by the originating JMS client application.
- If the JMS provider is required to filter messages based on the JMSPriority header value in the JMS message, enter a value for *Priority*.

  Valid values for message priority are from 0 (lowest) to 9 (highest); for example, you can enter `5` to receive messages of priority 5. You can also qualify the selector; for example, `> 4` to receive messages with a priority greater than 4, or `BETWEEN 4 AND 8` to receive messages with a priority in the range 4 to 8.

  Leave this property blank if you do not want the input node to make a selection based on JMSPriority.
- If the JMS provider is required to filter messages based on the JMSMessageID header, enter a value for *Message ID*.

  Enter a specific Message ID, or enter a conditional selector; for example, enter `> WMBRK123456` to return messages where the Message ID is greater than `WMBRK123456`.

  Leave this property blank if you do not want the input node to make a selection based on JMSMessageID.
- If the JMS provider is required to filter messages based upon the JMSRedelivered header, enter a value for *Redelivered*:
  - Enter `FALSE` if the input node accepts only messages that have not been redelivered by the JMS Provider.
  - Enter `TRUE` if the input node accepts only messages that have been redelivered by the JMS Provider.
  - Leave this property blank if you do not want the input node to make a selection based on JMSRedelivered.
- If the JMS provider is required to filter messages based upon the JMSCorrelationID header, enter a value for *Correlation ID*.

  Enter a specific Correlation ID or enter a conditional string; for example, `WMBRKABCDEFG` returns messages whose Correlation ID matches this value.

  Leave this property blank if you do not want the input node to make a selection based on JMSCorrelationID.

6. On the Advanced tab, define the transactional characteristics of how the message is handled by selecting an option from the *Transaction Mode* drop-down list. Choose one of the following options:
   - Select *none* if the incoming message is to be treated as non persistent. If you select this value, the message is received using a non-transacted JMS session that is created using the `Session.AUTO_ACKNOWLEDGE` flag.
   - Select *local* if the JMSInput node should coordinate the commit or roll back of JMS messages that are received by the node, along with any other resources such as DB2 or WebSphere MQ that perform work within the message flow. If you select this value, the node uses a transacted JMS session.
   - Select *global* if the JMSInput node should participate in a global message flow transaction that will be managed by the broker's external syncpoint coordinator. The syncpoint coordinator is the broker's queue manager on distributed systems and RRS (Resource Recovery Services) on z/OS. If you select this value, any messages that are received by the node are globally coordinated using an XA JMS session.

Connecting the terminals:

For each message that is received successfully, the JMSInput node routes the message to the Out terminal. If this fails, the message is retried. If the retry threshold is reached, where the threshold is defined by the BackoutThreshold property of the node, the message is routed to the Failure terminal. You can connect nodes to the Failure terminal to handle this condition. If you have not connected nodes to the Failure terminal, the message is written to the backout destination. If a backout destination has not been provided, the node issues a BIP4669 error message and stops processing further input.

If the message is caught by the JMSInput node after an exception has been thrown elsewhere in the message flow, the message is routed to the Catch terminal. If you have not connected nodes to the Catch terminal, the node backs out messages for re-delivery until the problem is resolved, or the backout threshold is reached.

You must define a backout destination. If you do not define a backout destination, the node issues a BIP4669 error message and stops processing further input.

Configuring for coordinated transactions:

When you include a JMSInput node in a message flow, the value that you set for *Transaction Mode* defines whether messages are received under syncpoint.

- If you set this property to `Global`, the message is received under external syncpoint coordination; that is, within a WebSphere MQ unit of work. Any messages that are sent subsequently by an output node in the same instance of the message flow are put under syncpoint, unless the output node overrides this setting explicitly.
- If you set this property to `Local`, the message is received under the local syncpoint control of the JMSInput node. Any messages that are sent subsequently by an output node in the flow are not put under local syncpoint, unless an individual output node specifies that the message must be put under local syncpoint.
- If you set this property to `none`, the message is not received under syncpoint. Any messages that are sent subsequently by an output node in the flow are not put under syncpoint, unless an individual output node specifies that the message must be put under syncpoint.

The JMS provider can supply additional .jar files that are required for transactional support. Refer to the JMS provider documentation. For example, on distributed systems, the WebSphere MQ JMS provider supplies an extra .jar file: `com.ibm,mqetclient.jar`. You must add this .jar file to the broker shared-classes directory. Refer to "Making the JMS Provider client available to the JMS nodes" on page 129 in this topic.

When you want to receive messages under external syncpoint, you must take additional configuration steps, which need be applied only the first time that a JMSOutput or JMSInput node is deployed to the broker for a particular JMS provider:

- On distributed systems, the external syncpoint coordinator for the broker is WebSphere MQ. Before you deploy a message flow in which the *Transaction Coordination* property is set to `Global`, modify the queue manager .ini file to include extra definitions for each JMS provider Resource Manager that participates in globally-coordinated transactions.
  - On Windows, if you have WebSphere MQ Version 5 installed:
    1. Start WebSphere MQ Services.
    2. Right-click the queue manager name and click **Properties**.
    3. Click the **Resource properties** tab.
    4. Set the *SwitchFile* property to the following value:
       ```
       install_dir/bin/ JMSSwitch.dll
       XAOpenString=Initial Context,location JNDI,Optional_parms
       ThreadOfControl=THREAD
       ```
  - On Windows, if you have WebSphere MQ Version 6.0 installed:
    1. Start WebSphere MQ Explorer.
    2. Right-click the queue manager name in the left pane and click **Properties**.
    3. Click **XA resource managers** in the left pane.
    4. Set the *SwitchFile* property to the following value:
       ```
       install_dir/bin/ JMSSwitch.dll
       XAOpenString=Initial Context,location JNDI,Optional_parms
       ThreadOfControl=THREAD
       ```

    Refer to the *WebSphere MQ System Administration Guide* for more information.
  - On Linux and UNIX systems, add a stanza to the queue manager ini file for each JMS provider.

    For example:
    ```
    XAResourceManager:
    Name=Jms_Provider_Name
    SwitchFile=/install_dir/bin/ JMSSwitch.so
    XAOpenString=Initial Context,location JNDI,Optional_parms
    ThreadOfControl=THREAD
    ```

    Where:

    Name is an installation defined name that identifies a JMS provider Resource Manager.

    SwitchFile
    is the file system path to the JMSSwitch library that is supplied in the bin directory of the broker.

    The values for XAOpenString are as follows:
    - *Initial Context* is the value that is set in the JMSInput node basic property *Initial Context Factory*.

- *location JNDI* is the value that is set in the JMSInput node basic property *Location of JNDI*. This value should include the leading keyword, which is `file:/`, `iiop:/`, or `ldap:/`.

The following parameters are optional:

- *LDAP Principal* matches the value that is set for the broker by using the mqsicreatebroker or mqsichangebroker commands.
- *LDAP Credentials* matches the value that is set for the broker by using the mqsicreatebroker or mqsichangebroker commands.
- *Recovery Connection Factory Name* is the JNDI administered connection factory that is defined in the bindings file. If a value is not specified, you must add a default value for `recoverXAQCF` to the bindings file. In either case, the Recovery Connection Factory should be defined as an XA Queue Connection Factory for the JMS provider that is associated with the Initial Context Factory.

The optional parameters are comma separated and are positional. Therefore, any parameters that are missing must be represented by a comma.

1. Update the Java CLASSPATH environment variable for the broker's queue manager to include a reference to xarecovery.jar; for example:

   `install_dir/classes/xarecovery.jar`
2. Update the Java PATH environment variable for the broker's queue manager to point to the bin directory in which the SwitchFile is located; for example:

   `install_dir/bin`

Refer to the *WebSphere MQ System Administration Guide* for more information.

To use the same queue manager for both the broker and the JMS provider, ensure that your WebSphere MQ installation is at the minimum required level: Version 5.3 CSD12 or Version 6 FixPack 1.

– On z/OS, the external syncpoint manager is Resource Recovery Services (RRS). The only JMS provider that is supported on z/OS is WebSphere MQ JMS. The only transport option that is supported for WebSphere MQ JMS on z/OS is the bind option.

Syncpoint control for the JMS provider is managed with RRS syncpoint coordination of the queue manager of the broker. You do not need to modify the .ini file.

Terminals and properties

The terminals of the JMSInput node are described in the following table.

Terminal	**Description**
Failure	The output terminal to which the message is routed if an error occurs. Even if the Validation property is set, messages propagated to this terminal are not validated.
Out	The output terminal to which the message is routed if it is successfully retrieved from the WebSphere MQ queue.
Catch	The output terminal to which the message is routed if an exception is thrown downstream and caught by this node.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the bar file to deploy it).

The Description properties of the JMSInput node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The JMS Connection properties of the JMSInput node are described in the following table.

Property	M	C	Default	Description
Initial Context Factory	Yes		`com.sun.jndi.fscontext.RefFSContextFactory`	This property is the starting point for a JNDI name space. A JMS application uses the initial context to obtain and look up the connection factory and queue or topic objects for the JMS provider. The default value is that value used when WebSphere MQ Java is used as the JMS provider.
Location JNDI Bindings	Yes			The system path or the LDAP location for the bindings file.
Connection Factory Name	Yes			The name of the connection factory that is used by the JMSInput node to create a connection to the JMS provider.
Backout Destination	No			The destination that is used by the JMSInput node when a message cannot be processed by the message flow because of errors in the message.
Backout Threshold	No		0	The value that controls when a re-delivered message is put to the backout destination.

The Advanced properties of the JMSInput node are described in the following table.

Property	M	C	Default	Description
Transaction Mode	Yes	No	`none`	This property controls whether the incoming message is received under external syncpoint, local syncpoint, or out of syncpoint. Valid values are `none`, `local`, and `global`.

JMSMQTransform node

This topic contains the following sections:
- "Purpose" on page 136
- "Using the JMSMQTransform node in a message flow" on page 136
- "Configuring the JMSMQTransform node" on page 136
- "Terminals and properties" on page 136

Purpose

Use the JMSMQTransform node to transform a message with a JMS message tree into a message that has a message tree structure that is compatible with the format of messages that are produced by the WebSphere MQ JMS provider.

You can use the JMSMQTransform node to send messages to legacy message flows and to interoperate with WebSphere MQ JMS and WebSphere Event Broker publish/subscribe.

The JMSMQTransform node is represented in the workbench by the following icon:

Using the JMSMQTransform node in a message flow

The JMS Nodes sample contains a message flow in which the JMSMQTransform node is used. Refer to this sample for an example of how to use the JMSMQTransform node.

Configuring the JMSMQTransform node

When you have put an instance of the JMSMQTransform node into a message flow, you can configure it. To display its properties, either double-click the node, or right-click the node and click **Properties**.

Optional: On the Description tab, enter a short description, a long description, or both. You can also change the name of the node on this tab.

Terminals and properties

The terminals of the JMSMQTransform node are described in the following table:

Terminal	Description
Failure	The output terminal to which the message is routed if an error occurs. Even if the Validation property is set, messages propagated to this terminal are not validated.
Out	The output terminal to which the message is routed if it is successfully retrieved from the JMS destination.
In	The input terminal that accepts a message for processing by the node.

The following table describes the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the bar file to deploy it).

The JMSMQTransform node Description properties are described in the following table:

Property	M	C	Default	Description
Node name	No	No	The node type	The name of the node.
Short Description	No	No		A brief description of the node.

Property	M	C	Default	Description
Long Description	No	No		Text that describes the purpose of the node in the message flow.

JMSOutput node

This topic contains the following sections:

- "Purpose"
- "Using the JMSOutput node in a message flow"
- "Making the JMS provider client available to the JMS nodes"
- "Using the Message Destination Mode" on page 138
- "Configuring the JMSOutput node" on page 139
- "Terminals and properties" on page 143

Purpose

Use the JMSOutput node to send messages to JMS destinations. The JMSOutput node acts as a JMS message producer and can publish all six message types that are defined in the Java Message Service Specification, version 1.1. Messages are published by using method calls, which are described in the JMS specification.

The JMSOutput node is represented in the workbench by the following icon:

Using the JMSOutput node in a message flow

The JMS Nodes sample contains a message flow in which the JMSOutput node is used. Refer to this sample for an example of how to use the JMSOutput node.

Message flows that handle messages that are received from connections to JMS providers must always start with a JMSInput node. If you include the JMSOutput node in a message flow, you do not need to include a JMSInput node; but if you do not include a JMSInput node, you must include the MQJMSTransform node to transform the message to the format that is expected by the JMSOutput node.

If you are propagating JMS messages and creating a message flow to use as a subflow, use an instance of the JMSOutput node as the last node in order to create an out terminal for the subflow.

Making the JMS provider client available to the JMS nodes

For distributed systems, copy the Java .jar files and any native libraries for the JMS provider client into the broker shared-classes directory; for example, C:\Documents and Settings\All Users\Application Data\IBM\MQSI\shared-classes on Windows. Copying the files to the shared-classes directory ensures that the Java class path for the JMS nodes is set correctly.

For z/OS, there is no shared-classes directory; instead, perform the following steps:

1. Specify each JMS provider Java .jar file in the class path in the BIPPROF member of the broker's Partitioned Data Set (PDS).
2. Update the LIBPATH with any native libraries.
3. Submit the BIPGEN JCL job to update the broker ENVFILE.

Using the Message Destination Mode

The JMSOutput node acts as a message producer and supports the following message scenarios:

- "Sending a datagram message"
- "Sending a reply message"
- "Sending a request message" on page 139

Sending a datagram message

A *datagram* is a self-contained, independent entity of data that carries sufficient information to be routed from the source to the destination computer, without reliance on earlier exchanges between the source and destination computer and the transporting network. The following instructions describe how to send a datagram message:

1. On the Basic tab, set the message destination depending on the message model that is being used. Set one of the following properties to a valid JNDI administered object name:
   - Publication Topic
   - Destination Queue
2. Leave the Reply To Destination field blank.

The node resolves the name of the JNDI administered object, which is supplied in either Publication Topic or Destination Queue, and sends the message to that JMS Destination.

Sending a reply message

The sender of a message might want the recipient to reply to the message. In this case, the JMSOutput message can treat the outgoing message as a reply, and route it according to the value that is obtained from the JMSReplyTo property from the request message. You can modify the value of the JMSReplyTo property in the MbMessage for instances using a Compute node or a JavaCompute node. This action allows dynamic routing of messages from the JMSOutput node. The node sends the message to the JMS Destination name that is set in the JMSReplyTo field of the MbMessage Tree.

The JMSReplyTo value in the MbMessage Tree represents the name of the JMS Destination that is resolved from JNDI. For example:

```
queue://QM_mn2/myJMSQueue4
```

In this case, the value is the JMS-provider specific representation of a JMS Destination for the WMQSeries JMS Provider.

If you do not want to specify a resolved JMS destination name, the JMSOutput node can also accept a JNDI administered object name in the JMSReplyTo field. However, it is necessary to resolve an administered object name through JNDI before the node can route the message to the underlying JMS Destination. In this case, the value in the JMSReplyTo field should be prefixed with the string: `jndi:\\`. For example:

```
jndi:\\jmsQ4
```

where `jmsQ4` is the name of the JNDI-administered object.

Performance might be slightly impacted when you use this method because of the need to look up the administered object in JNDI.

Sending a request message

The JMSOutput node can send a message to a JMS Destination with the expectation of a response from the message consumer that processes the request. The following instructions describe how to send a request message:

1. On the Basic tab, set the message destination depending on the message model that is being used. Set one of the following properties to a valid JNDI-administered object name:
   - Publication Topic
   - Destination Queue
2. The JMSReplyTo destination in the outgoing message can be derived from the JMSReplyTo field of the MbMessage Tree that is passed to the node. Alternatively, this value can be overridden by a JNDI-administered object name that is set in the Reply To Destination node property.

   To allow the JMSOutput node to set the JMSReplyTo property dynamically in the outgoing message, leave the Reply To Destination field blank on the Basic tab, and set the JMSReplyTo value in the MbMessage using a Compute node or a JavaCompute node.

The node looks first for a value in the JMSReplyTo field of the MbMessage. If it finds it, it passes this value into the JMSReplyTo field of the outgoing message. However, if the Reply To Destination field of the Basic tab has been specified, this value overrides anything that is set previously in the JMSReplyTo property of the outgoing message, after first resolving the name of the JNDI-administered object.

The node resolves the name of the JNDI-administered object that is supplied in either Publication Topic or Destination Queue, and sends the message to that JMS Destination.

Configuring the JMSOutput node

When you have put an instance of the JMSOutput node into a message flow, you can configure it. To display its properties, either double-click the node, or right-click the node and click **Properties**.

All mandatory properties that do not have a default value defined are marked with an asterisk.

Configure the JMSOutput node as follows:

1. Optional: On the Description tab, enter a short description, a long description, or both. You can also rename the node on this tab.
2. On the JMS Connection tab, set the following properties:
   - Enter an Initial context factory value. A JMS application uses the initial context to obtain and look up the JNDI administered objects for the JMS provider. The default value is `com.sun.jndi.fscontext.RefFSContextFactory`, which defines the file-based initial context factory for the WebSphere MQ JMS provider.

     To identify the name of the Initial Context Factory for the JMS provider, refer to the JMS provider documentation.

- Enter a value for the Location JNDI bindings. This value specifies either the file system path or the LDAP location for the bindings file. The bindings file contains definitions for the JNDI-administered objects that are used by the JMSOutput node.

  When you enter a value for Location JNDI Bindings, ensure that it complies with the following instructions:
  - Construct the bindings file before you deploy a message flow that contains a JMSOutput node.
  - Do not include the file name of the bindings file in this field.
  - If you have specified an LDAP location that requires authentication, configure both the LDAP principal (userid) and LDAP credentials (password) separately. These values are configured at broker level. For information about configuring these values, refer to the **mqsicreatebroker** and **mqsichangebroker** commands.
  - The string value must include the leading keyword, which is one of:
    - `file:/`
    - `iiop:/`
    - `ldap:/`

  For information about constructing the JNDI-administered objects bindings file, refer to the documentation that is supplied with the JMS provider.
- Enter a Connection factory name. The connection factory name is used by the JMSOutput node to create a connection to the JMS provider. This name must already exist in the bindings file.

3. On the Basic tab, if the JMSOutput node is to be used to publish a topic, set the following properties:
   - If the message is to be treated as a reply, select Send reply to the JMS header ″JMSReplyTo″ destination. The JMS provider is supplied with the JMSReplyTo value from the JMSTransport_Header_values section of the message tree.
   - If the JMSOutput node is to be used to send point-to-point messages, enter the Destination queue name for the JMS queue name that is listed in the bindings file.
   - Enter the name of the Publication Topic.
     - If this property is configured, the node operates only in the publish/subscribe message domain.
     - This property is mutually exclusive with the Destination queue property.
     - The Publication Topic name must conform to the standards of the JMS provider that is being used by the node.
   - Enter a value for Reply To Destination. You can enter a JMS destination, which can be either a subscription queue or a destination topic. The Reply To Destination is the name of the JMS destination to which the receiving application should send a reply message. For a reply message to be returned to this JMS destination, the JMS destination name must be known to the domain of the JMS provider that is used by the receiving client.

     The default value is blank, in which case the JMS output message can be regarded as a datagram. If the field is blank, the JMSOutput node does not expect a reply from the receiving JMS client.
4. On the Advanced tab, set the following properties:
   - If the JMSOutput node is required to generate a new Correlation ID for the message, select the check box. The check box is cleared by default; if you

leave the check box cleared, the Correlation ID of the output message is taken from the JMSCorrelationID field in the JMSTransport_Header_Values section of the message tree.

- To define the transactional characteristics of how the message is handled, select the *Transaction Mode*:
  - Select *None* if the outgoing message is to be treated as non persistent. If you select this value, the message is sent using a non-transacted JMS session that is created using the `Session.AUTO_ACKNOWLEDGE` flag.
  - Select *Local* if the input node that received the message should coordinate the commit or roll-back of JMS messages that have been sent by the JMSOutput node, along with any other resources, such as DB2 or WebSphere MQ, that perform work within the message flow. If you select this value, the node uses a transacted JMS session.
  - Select *Global* if the JMSOutput node should participate in a global message flow transaction that is managed by the broker's external syncpoint coordinator. The syncpoint coordinator is the broker's queue manager on distributed systems, and RRS (Resource Recovery Services) on z/OS. If you select this value, any messages that are received by the node are globally coordinated using an XA JMS session.
- You can set the persistence of the outgoing JMS message by using the Delivery Mode property. Select an option from the drop-down list:
  - *Non Persistent* to indicate to the JMS provider that the message should be treated as non persistent
  - *Persistent* to mark messages as persistent to the JMS provider and to ensure that they are preserved until they are delivered successfully to a receiving JMS client application
- Enter a value, in milliseconds, for Message Expiration to request that the JMS provider keeps the output JMS message for a specified time. The default value `0` is used to indicate that the message should not expire.
- To assign a relative importance to the message, select an option from the Message Priority drop-down list. This value can be used for message selection by a receiving JMS client application or a JMSOutput node. Valid values for message priority are from `0` (lowest) to `9` (highest). The default value is `4`, which indicates medium priority. Priorities in the range `0` to `4` relate to normal delivery. Priorities in the range `5` to `9` relate to graduations of expedited delivery.

Connecting the terminals:

Connect the In terminal of the JMSOutput node to the node from which outbound messages are routed.

Connect the Out terminal of the JMSOutput node to another node in the message flow to process the message further, to process errors, or to send the message to an additional destination.

Configuring for coordinated transactions:

When you include a JMSOutput node in a message flow, the value that you set for Transaction Mode defines whether messages are sent under syncpoint.

- If you set the Transaction Mode to *Global*, the message is sent under external syncpoint coordination; that is, within a WebSphere MQ unit of work. Any messages that are sent subsequently by an output node in the same instance of the message flow are put under syncpoint, unless the output node overrides this setting explicitly.

- If you set the Transaction Mode to *Local*, the message is sent under the local syncpoint control of the JMSOutput node. Any messages that are sent subsequently by an output node in the flow are not put under local syncpoint, unless an individual output node specifies that the message must be put under local syncpoint.
- If you set the Transaction Mode to *None*, the message is not sent under syncpoint. Any messages that are sent subsequently by an output node in the flow are not put under syncpoint, unless an individual output node specifies that the message must be put under syncpoint.

The JMS provider can supply additional .jar files that are required for transactional support. Refer to the JMS provider documentation. For example, on distributed systems, the WebSphere MQ JMS provider supplies an extra .jar file, `com.ibm,mqetclient.jar`, which must be added to the broker shared-classes directory. Refer to "Making the JMS provider client available to the JMS nodes" on page 137 in this topic.

When you want to receive messages under external syncpoint, you must perform additional configuration steps, which need to be applied only the first time that a JMSOutput or JMSInput is deployed to the broker for a particular JMS provider:

- On distributed systems, the external syncpoint coordinator for the broker is WebSphere MQ. Before you deploy a message flow in which the Transaction Mode is set to *Global*, modify the queue manager .ini file to include extra definitions for each JMS provider Resource Manager that participates in globally coordinated transactions:
  - On Windows:
    1. If you have WebSphere MQ Version 5 installed, start WebSphere MQ Services and right-click the queue manager name. Click **Properties** and click the **Resource properties** tab.

       If you have WebSphere MQ Version 6.0 installed, start WebSphere MQ Explorer and right-click the queue manager name in the left pane. Click **Properties** and click **XA resource managers** in the left pane. Refer to the *WebSphere MQ System Administration Guide* for more information.
    2. Set the SwitchFile property to the following value:

       ```
       install_dir/bin/ JMSSwitch.dll
       XAOpenString=Initial Context,location JNDI,Optional_parms
       ThreadOfControl=THREAD
       ```
  - On Linux and UNIX systems, add a stanza to the queue manager's .ini file for each JMS provider.

    For example:

    ```
    XAResourceManager:
    Name=Jms_Provider_Name
    SwitchFile=/install_dir/bin/ JMSSwitch.so
    XAOpenString=Initial Context,location JNDI,Optional_parms
    ThreadOfControl=THREAD
    ```

    Where:

    Name is an installation-defined name that identifies a JMS provider Resource Manager.

SwitchFile
is the file system path to the JMSSwitch library that is supplied in the bin directory of the broker.

The values for *XAOpenString* are as follows:

- *Initial Context* is the value that is set in the JMSInput node basic property Initial context factory.
- *location JNDI* is the value that is set in the JMSInput node basic property Location of JNDI. This value should include the leading keyword, which is `file:/`, `iiop:/` or `ldap:/`

The following parameters are optional:

- *LDAP Principal* matches the value that is set for the broker by using the **mqsicreatebroker** or **mqsichangebroker** commands.
- *LDAP Credentials* matches the value that is set for the broker by using the **mqsicreatebroker** or **mqsichangebroker** commands.
- *Recovery Connection Factory Name* is the JNDI administered connection factory that is defined in the bindings file. If a value is not specified, a default value for `recoverXAQCF` must be added to the bindings file. In either case, the Recovery Connection Factory should be defined as an XA Queue Connection Factory for the JMS provider that is associated with the Initial Context Factory.

The optional parameters are comma-separated and are positional. Therefore, any parameters that are missing must be represented by a comma.

1. Update the Java CLASSPATH environment variable for the broker's queue manager to include a reference to xarecovery.jar; for example:
   ```
   <Broker Installation Path>/classes/xarecovery.jar
   ```
2. Update the Java PATH environment variable for the broker's queue manager to point to the bin directory, which is where the switch file is located; for example:
   ```
   <Broker Installation Path>/bin
   ```

Refer to the *WebSphere MQ System Administration Guide* for more information.

To use the same queue manager for both the broker and the JMS provider, ensure that your WebSphere MQ installation is at the minimum required level: Version 5.3 CSD12 or Version 6.0 Fix Pack 1.

– On z/OS, the external syncpoint manager is Resource Recovery Services (RRS). The only JMS provider that is supported on z/OS is WebSphere MQ JMS. The only transport option that is supported for WebSphere MQ JMS on z/OS is the bind option.

Syncpoint control for the JMS provider is managed with RRS syncpoint coordination of the queue manager of the broker. You do not need to modify the .ini file.

Terminals and properties

The terminals of the JMSOutput node are described in the following table.

Terminal	Description
Failure	The output terminal to which the message is routed if an error occurs. Even if the Validation property is set, messages that are propagated to this terminal are not validated.
Out	The output terminal to which the message is routed if it is successfully retrieved from the WebSphere MQ queue.

Terminal	Description
Catch	The output terminal to which the message is routed if an exception is thrown downstream and caught by this node.

The following tables describe the node properties. The column headed M indicates whether the property is mandatory (marked with an asterisk if you must enter a value when no default is defined), the column headed C indicates whether the property is configurable (you can change the value when you add the message flow to the bar file to deploy it).

The Description properties of the JMSOutput node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The JMS Connection properties of the JMSOutput node are described in the following table.

Property	M	C	Default	Description
Initial Context Factory	Yes		`com.sun.jndi.fscontext.RefFSContextFactory`	This property is the starting point for a JNDI name space. A JMS application uses the initial context to obtain and look up the connection factory and queue or topic objects for the JMS provider. The default value is that value used when WebSphere MQ Java is used as the JMS provider.
Location JNDI Bindings	No			The system path or the LDAP location for the bindings file.
Connection Factory Name	No			The name of the connection factory that is used by the JMSOutput node to create a connection to the JMS provider.

The Advanced properties of the JMSOutput node are described in the following table.

Property	M	C	Default	Description
New Correlation ID	No			This property is selected if a New Correlation ID is required.
Transaction Mode	Yes	No	`None`	This property controls whether the incoming message is received under syncpoint. Valid values are `None`, `Local`, and `Global`.

Property	M	C	Default	Description
Delivery Mode	No		`Non Persistent`	This property controls the persistence mode that a JMS provider uses for a message. Valid values are: • `Automatic`: the mode from the input message is inherited • `Persistent`: the message survives if the JMS provider has a system failure • `Non Persistent`: the message is lost if the JMS provider has a system failure
Message Expiration (ms)	No		`0`	This property controls the length of time for which the JMS provider keeps the output JMS message. Values are in milliseconds and the default value, `0`, is used to indicate that the message should not expire.
Message Priority	No		`4`	This property assigns relative importance to the message and it can be used for message selection by a receiving JMS client application or a JMSOutput node.

MQeInput node

Attention: The use of message flows that contain MQeInput and MQeOutput nodes in WebSphere Message Broker Version 6.0 is deprecated. The behavior that is described here is intended only for when you are deploying from Version 6.0 to a previous version, and to provide a route for migration. Redesign your flows to remove the MQe nodes and replace them with MQ nodes that are configured to your own specifications and coordinated with your MQe Gateway configuration. For more details, see Migrating a message flow that contains WebSphere MQ Everyplace nodes.

This topic contains the following sections:
- "Purpose"
- "Using the MQeInput node in a message flow" on page 146
- "WebSphere MQ Everyplace documentation" on page 146
- "Configuring the MQeInput node" on page 147
- "Terminals and properties" on page 150

Purpose

Use the MQeInput node to receive messages from clients that connect to the broker using the WebSphere MQ Mobile Transport protocol.

The MQeInput node receives messages that are put to a message flow from a specified bridge queue on the broker's WebSphere MQ Everyplace queue manager. The node also establishes the processing environment for the messages. You must create and configure the WebSphere MQ Everyplace queue manager before you deploy a message flow that contains this node.

Message flows that handle messages that are received across WebSphere MQ Everyplace connections must always start with an MQeInput node. You can set the MQeInput node's properties to control the way in which messages are received; for example, you can indicate that a message is to be processed under transaction control.

When you deploy message flows that contain WebSphere MQ Everyplace nodes to a broker, you must deploy them to a single execution group, regardless of the number of message flows. The WebSphere MQ Everyplace nodes in the message flows must all specify the same WebSphere MQ Everyplace queue manager name. If you do not meet this restriction, an error is raised when you deploy.

If you include an output node in a message flow that starts with an MQeInput node, it can be any of the supported output nodes, including user-defined output nodes; you do not need to include an MQeOutput node. You can create a message flow that receives messages from WebSphere MQ Everyplace clients and generates messages for clients that use any of the supported transports to connect to the broker, because you can configure the message flow to request the broker to provide any conversion that is necessary.

WebSphere MQ Everyplace Version 1.2.6 is used by WebSphere Event Broker. This is compatible with later versions of WebSphere MQ Everyplace. Clients that use later versions of WebSphere MQ Everyplace (for example, Version 2.0), work correctly when connected to this node, although additional functionality that is not supported in Version 1.2.6 (for example, JMS support) does not work.

Queue managers are *not* interchangeable between different versions of WebSphere MQ Everyplace. Nodes must use a queue manager that was created using Version 1.2.6. Similarly, the client must use its own level of the code when creating a queue manager.

z/OS You cannot use MQeInput nodes in message flows that you deploy to z/OS systems.

If you create a message flow to use as a subflow, you cannot use a standard input node; you must use an instance of the Input node as the first node to create an In terminal for the subflow.

If your message flow does not receive messages across WebSphere MQ connections, you can choose one of these other input nodes.

The MQeInput node is represented in the workbench by the following icon:

Using the MQeInput node in a message flow

For an example of how this node can be used, consider a farmer who checks his fields to see how well they are irrigated. He is carrying a PDA device with WebSphere MQ Everyplace installed. He sees an area of field that requires water, so he uses his PDA and a Global Satellite Navigation link to send a message to an MQeInput node. A message is published by a Publication node so that a remote SCADA device can pick up the message and trigger the irrigation sprinklers. The farmer can see the water delivered to the field, minutes after sending his message.

WebSphere MQ Everyplace documentation

Find further information about WebSphere MQ Everyplace, and the properties of the node, in the WebSphere MQ Everyplace documentation on the WebSphere MQ Web page.

Configuring the MQeInput node

When you have put an instance of the MQeInput node into a message flow, you can configure it. To display its properties, either double-click the node, or right-click the node and click **Properties**. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

Configure the MQeInput node as follows:

1. Optional: On the Description tab, enter a short description, a long description, or both. You can also rename the node on this tab.
2. On the General tab, set the following properties:
   a. Enter the *Queue Name* of the WebSphere MQ Everyplace bridge queue from which this input node retrieves messages. If the queue does not exist, it is created for you when the message flow is deployed to the broker.
   b. Set the level of *Trace* that you want for this node. If trace is active, the trace information is recorded into the file identified by *Trace Filename* (described below). Choose a level of trace:
      - `None` (the default). No trace output is produced, unless a fatal error occurs.
      - `Standard`. Minimal trace output is generated to reflect the overall operations of the node.
      - `Debug`. Trace information is recorded at a level that helps you to debug WebSphere MQ Everyplace programs.
      - `Full`. All available debug information is recorded to provide a full record of the node activities.

      If you set the trace level to `Debug` or `Full`, you will impact the performance of WebSphere MQ Everyplace, and significant trace files can be generated. Use these options for short periods only.
   c. In *Trace Filename*, specify the name of the file to which the trace information is written. The directory structure in which the file is specified must already exist; it cannot be created during operation.
   d. Select the *Transaction Mode* to define the transactional characteristics of how this message is handled:
      - If you select `Automatic`, the incoming message is received under syncpoint if it is marked persistent; otherwise it is not. The transactionality of any derived messages that are sent subsequently by an output node is determined by the incoming persistence property, unless the output node has overridden transactionality explicitly.
      - If you select `Yes`, the incoming message is received under syncpoint. Any derived messages that are sent subsequently by an output node in the same instance of the message flow are sent transactionally, unless the output node has overridden transactionality explicitly.
      - If you select `No`, the incoming message is not received under syncpoint. Any derived messages that are sent subsequently by an output node in the message flow are sent non-transactionally, unless the output node has specified that the message must be put under syncpoint.
   e. The *Use Config File* check box is not selected by default; values for all properties for the MQeInput node are taken from the Properties view.

      If you select the check box, the definition of all properties is extracted from the file that is identified by *Config Filename* (described below) with the exception of the following properties:

- The *Queue Name* and *Config Filename* General properties
- All Default properties

Use a configuration file only to specify additional properties for the node. If the properties in the Properties view are sufficient for your needs, do not select the *Use Config File* check box.

f. If you have selected the *Use Config File* check box, enter the full path and name of the configuration file for WebSphere MQ Everyplace in *Config Filename*. This file must be installed on the system that supports every broker to which this message flow is deployed. If the file does not exist, an error is detected when you deploy the message flow. The default file name is MQeConfig.ini.

g. In *Queue Manager Name*, specify the name of the WebSphere MQ Everyplace queue manager. This queue manager is not related to the queue manager of the broker to which you deploy the message flow that contains this node.

   Only one WebSphere MQ Everyplace queue manager can be supported. Only one execution group can contain MQeInput or MQeOutput nodes. This property must therefore be set to the same value in every MQeInput node that is included in every message flow that you deploy to the same broker.

3. On the Channel tab, set the maximum number of channels that are supported by WebSphere MQ Everyplace in *Max Channels*. The default value is zero, which means that there is no limit.
4. On the Registry tab, set the following properties:
   a. Select the type of registry from the *Registry Type* drop down list:
      - `File Registry`. Registry and security information is provided in the *Directory* specified below.
      - `Private Registry`. You create the queue manager manually within WebSphere MQ Everyplace, specifying the security parameters that you need.

   b. In *Directory*, specify the directory in which the registry file is located. This property is valid only if you have selected a *Registry Type* of `File Registry`.

   c. If you have selected a *Registry Type* of `Private Registry`, complete the following properties (for further details of these properties, refer to the WebSphere MQ Everyplace documentation):
      - Specify a *PIN* for the associated queue manager.
      - Specify a *Certificate Request PIN* for authentication requests.
      - Provide a *Keyring Password* to be used as a seed for the generation of crypto keys.
      - In *Certificate Host*, specify the name of the certificate server that WebSphere MQ Everyplace uses for authentication.
      - In *Certificate Port*, specify the number of the port for the certificate server that WebSphere MQ Everyplace uses for authentication.
5. On the Listener tab, set the following properties that define the connection type for WebSphere MQ Everyplace:
   a. In *Listener Type*, select the adapter type to use from the drop-down list. The default value is `Http`; you can also select `Length` or `History`. For further details, refer to the WebSphere MQ Everyplace documentation.

   b. In *Hostname*, specify the hostname of the server. Set this property to the special value `localhost` or to the TCP/IP address `127.0.0.1` (the default value), both of which resolve correctly to the hostname of the server to which the message flow is deployed. You can also use any valid hostname or TCP/IP address in your network, but you must use a different message flow for each broker to which you deploy it, or configure this property at deploy time.

c. In *Port*, specify the port number on which WebSphere MQ Everyplace is listening. If more than one MQeInput node is included in a message flow that is deployed to a single broker, each MQeInput node must specify a different number for this property. You must also ensure that the number that you specify does not conflict with other listeners on the broker system; for example, with WebSphere MQ. The default value is `8081`.

d. In *Time Interval*, specify the timeout value, in seconds, before idle channels are timed out. The default value is `300` seconds.

   Channels are persistent logical entities that last longer than a single queue manager request, and can survive network breakages, so it might be necessary to time out channels that have been inactive for a period of time.

Connecting the terminals:

The MQeInput node routes each message that it retrieves successfully to the Out terminal; if this fails, the message is retried. If the retry timeout expires (as defined by the BackoutThreshold attribute of the input queue), the message is routed to the Failure terminal; you can connect nodes to this terminal to handle this condition. If you have not connected the Failure terminal, the message is written to the backout queue.

If the message is caught by this node after an exception has been thrown further on in the message flow, the message is routed to the Catch terminal. If you have not connected the Catch terminal, the message loops continually through the node until the problem is resolved. You must define a backout queue or a dead-letter queue (DLQ) to prevent the message looping continuously through the node.

Configuring for coordinated transactions:

When you include an MQeInput node in a message flow, the value that you set for the *Transaction Mode* property defines whether messages are received under syncpoint:

- If you set the property to `Yes` (the default), the message is received under syncpoint (that is, within a WebSphere MQ unit of work). Any messages that are sent subsequently by an output node in the same instance of the message flow are put under syncpoint, unless the output node has overridden this explicitly.
- If you set the property to `Automatic`, the message is received under syncpoint if the incoming message is marked persistent. Otherwise, it is not. Any message that are sent subsequently by an output node is put under syncpoint, as determined by the incoming persistence property, unless the output node has overridden this explicitly.
- If you set the property to `No`, the message is not received under syncpoint. Any messages that are sent subsequently by an output node in the flow are not put under syncpoint, unless an individual output node has specified that the message should be put under syncpoint.

The MQOutput node is the only output node that you can configure to override this option.

Terminals and properties

The MQeInput node terminals are described in the following table.

Terminal	Description
Failure	The output terminal to which the message is routed if an error occurs.
Out	The output terminal to which the message is routed if it is successfully retrieved from the WebSphere MQ Everyplace queue.
Catch	The output terminal to which the message is routed if an exception is thrown downstream and caught by this node.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the bar file to deploy it).

The MQeInput node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, e.g. Input	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The MQeInput node General properties are described in the following table.

Property	M	C	Default	Description
Queue Name	Yes	Yes		The name of the WebSphere MQ Everyplace bridge queue from which this node retrieves messages for processing by this message flow.
Trace	Yes	No	`None`	The level of trace required for this node. Valid values are `None`, `Standard`, `Debug`, and `Full`.
Trace Filename	Yes	Yes	`\MQeTraceFile.trc`	The name of the file to which trace records are written.
Transaction Mode	Yes	No	`Yes`	This property controls whether the incoming message is received under syncpoint. Valid values are `Automatic`, `Yes`, and `No`.
Use Config File	Yes	No	Cleared	If you select the check box, a configuration file is used for this node.
Config Filename	Yes	Yes	`\MQeconfig.ini`	The name of the configuration file to be used if the *Use Config File* check box is selected.
Queue Manager Name	Yes	Yes	`ServerQM1`	The name of the WebSphere MQ Everyplace queue manager.

The MQeInput node Channel properties are described in the following table.

Property	M	C	Default	Description
Max Channels	Yes	No	`0`	The maximum number of channels that are supported by the WebSphere MQ Everyplace queue manager.

The MQeInput node Registry properties are described in the following table.

Property	M	C	Default	Description
Registry Type	Yes	Yes	`File Registry`	The type of registry information to be used. Valid values are `File Registry` and `Private Registry`.
Directory	Yes	Yes	`\ServerQM1\registry`	The directory in which the registry file exists (valid only if `File Registry` is selected).
PIN	Yes	Yes		The PIN that is associated with the WebSphere MQ Everyplace queue manager (valid only if `Private Registry` is selected).
Certificate Request PIN	Yes	Yes		The PIN that is used to request authentication (valid only if `Private Registry` is selected).
Keyring Password	Yes	Yes		The password that is used to see crypto keys (valid only if `Private Registry` is selected).
Certificate Host	Yes	Yes		The name of the certificate server (valid only if `Private Registry` is selected).
Certificate Port	Yes	Yes		The port of the certificate server (valid only if `Private Registry` is selected).

The MQeInput node Listener properties are described in the following table.

Property	M	C	Default	Description
Listener Type	Yes	Yes	`Http`	The adapter type for the listener. Valid values are `Http`, `Length`, and `History`.
Hostname	Yes	Yes	`127.0.0.1`	The hostname of the server.
Port	Yes	Yes	`8081`	The port on which WebSphere MQ Everyplace listens.
Time Interval (sec)	Yes	Yes	`300`	The WebSphere MQ Everyplace polling interval, specified in seconds.

MQeOutput node

Attention: The use of message flows that contain MQeInput and MQeOutput nodes in WebSphere Message Broker Version 6.0 is deprecated. The behavior that is described here is intended only for when you are deploying from Version 6.0 to a previous version, and to provide a route for migration. Redesign your flows to remove the MQe nodes and replace them with MQ nodes that are configured to your own specifications and coordinated with your MQe Gateway configuration. For more details see Migrating a message flow that contains WebSphere MQ Everyplace nodes.

This topic contains the following sections:
- “Purpose” on page 152
- “Using this node in a message flow” on page 152
- “WebSphere MQ Everyplace documentation” on page 152

- “Configuring the MQeOutput node”
- “Terminals and properties” on page 153

Purpose

Use the MQeOutput node to send messages to clients that connect to the broker using the WebSphere MQ Mobile Transport protocol.

The MQeOutput node forwards messages to WebSphere MQ Everyplace queue managers. If you specify a non-local destination queue manager, ensure that there is either a route to the queue manager, or store-and-forward queue servicing for the queue manager, if it exists.

You cannot use the MQeOutput node to change the transactional characteristics of the message flow. The transactional characteristics that are set by the message flow’s input node determine the transactional behavior of the flow.

z/OS You cannot use MQeOutput nodes in message flows that you deploy to z/OS systems.

If you create a message flow to use as a subflow, you cannot use a standard output node; you must use an instance of the Output node to create an out terminal for the subflow through which to propagate the message.

If you do not want your message flow to send messages to a WebSphere MQ Everyplace queue, choose another supported output node.

The MQeOutput node is represented in the workbench by the following icon:

Using this node in a message flow

For an example of how this node can be used, consider a farmer who checks his fields to see how well they are irrigated. He is carrying a PDA device with WebSphere MQ Everyplace installed. He sees that his fields are not being irrigated, so he uses his PDA and a Global Satellite Navigation link to check the water flow valve, and finds that it is faulty. This information is available because the remote SCADA device that is responsible for controlling the valve has published a diagnostic message, which was retrieved by the broker and forwarded to an MQeOutput node and on to the WebSphere MQ Everyplace client on his PDA.

WebSphere MQ Everyplace documentation

You can find further information about WebSphere MQ Everyplace, and the properties of the node, in the WebSphere MQ Everyplace documentation on the WebSphere MQ Web page.

Configuring the MQeOutput node

When you have put an instance of the MQeOutput node into a message flow, you can configure it. To display its properties, either double-click the node, or right-click the node and click **Properties**. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

Configure the MQeOutput node as follows:

1. Optional: On the Description tab, enter a short description, a long description, or both. You can also rename the node on this tab.
2. On the Basic tab, enter the *Queue Manager Name* and *Queue Name* that specify the destination for the output message if you select `Queue Name` in *Destination Mode* (described below). If you select another option for *Destination Mode*, you do not need to set these properties.
3. On the Advanced tab, select the *Destination Mode* from the drop-down list (this property identifies the queues to which to deliver the output message):
   - `Queue Name` (the default). The message is sent to the queue that is named in the *Queue Name* property. The properties *Queue Manager Name* and *Queue Name* (on the Basic tab) are mandatory if you select this option.
   - `Reply To Queue`. The message is sent to the queue that is named in the ReplyToQ field in the MQMD.
   - `Destination List`. The message is sent to the list of queues that are named in the LocalEnvironment (also known as DestinationList) that is associated with the message.
4. On the Request tab, set properties to define the characteristics of each output message generated.
   a. Select *Request* to indicate that each output message is marked in the MQMD as a request message (MQMD_REQUEST), and the message identifier field is cleared (set to MQMI_NONE) to ensure that WebSphere MQ generates a new identifier. Clear the check box to indicate that each output message is not marked as a request message. You cannot select this check box if you have selected a *Destination Mode* of `Reply To Queue`.
   b. Enter a WebSphere MQ Everyplace queue manager name in *Reply-to queue manager*. This name is inserted into the MQMD of each output message as the reply-to queue manager. This new value overrides the current value in the MQMD.
   c. Enter a WebSphere MQ Everyplace queue name in *Reply-to queue*. This name is inserted into the MQMD of each output message as the reply-to queue. This new value overrides the current value in the MQMD.

Connecting the terminals:

Connect the In terminal to the node from which outbound messages bound are routed.

Connect the Out or Failure terminal of this node to another node in this message flow if you want to send the message to an additional destination.

Terminals and properties

The MQeOutput node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the message is routed if a failure is detected when the message is put to the output queue.
Out	The output terminal to which the message is routed if it has been successfully put to the output queue, and if further processing is required within this message flow.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the bar file to deploy it).

The MQeOutput node Description properties are described in the following table.

Property	**M**	**C**	**Default**	**Description**
Node name	No	No	The node type, e.g. Input	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The MQeOutput node Basic properties are described in the following table.

Property	**M**	**C**	**Default**	**Description**
Queue Manager Name	No	Yes		The name of the WebSphere MQ Everyplace queue manager to which the output queue, which is specified in *Queue Name*, is defined.
Queue Name	No	Yes		The name of the WebSphere MQ Everyplace output queue to which this node puts messages.

The MQeOutput node Advanced property is described in the following table.

Property	**M**	**C**	**Default**	**Description**
Destination Mode	Yes	No	`Destination List`	The queues to which the output message is sent. Valid values are `Queue Name`, `Reply To Queue`, and `Destination List`.

The MQeOutput node Request properties are described in the following table.

Property	**M**	**C**	**Default**	**Description**
Request	Yes	No	Cleared	If you select the check box, each output message is generated as a request message.
Reply-to queue manager	No	Yes		The name of the queue manager to which the output queue, which is specified in *Reply-to queue*, is defined.
Reply-to queue	No	Yes		The name of the reply-to queue to which to put a reply to this request.

MQInput node

This topic contains the following sections:
- "Purpose" on page 155
- "Using the MQInput node in a message flow" on page 155
- "Configuring the MQInput node" on page 155
- "Terminals and properties" on page 160

Purpose

Use the MQInput node to receive messages from clients that connect to the broker using the WebSphere MQ Enterprise Transport, and that use the MQI and AMI application programming interfaces.

The MQInput node receives message input to a message flow from a WebSphere MQ message queue that is defined on the broker's queue manager. The node uses MQGET to read a message from a specified queue, and establishes the processing environment for the message. If appropriate, you can define the input queue as a WebSphere MQ clustered queue or shared queue.

Message flows that handle messages that are received across WebSphere MQ connections must always start with an MQInput node. You can set the properties of the MQInput node to control the way in which messages are received, by causing appropriate MQGET options to be set. For example, you can indicate that a message is to be processed under transaction control. You can also request that data conversion is performed on receipt of every input message.

If you include an output node in a message flow that starts with an MQInput node, it can be any of the supported output nodes, including user-defined output nodes; you do not need to include an MQOutput node. You can create a message flow that receives messages from WebSphere MQ clients and generates messages for clients that use any of the supported transports to connect to the broker, because you can configure the message flow to request that the broker provides any conversion that is necessary.

If you create a message flow to use as a subflow, you cannot use a standard input node; you must use an instance of the Input node as the first node to create an In terminal for the subflow.

If your message flow does not receive messages across WebSphere MQ connections, you can choose one of the supported input nodes.

The MQInput node is represented in the workbench by the following icon:

Using the MQInput node in a message flow

Look at the following sample to see how to use the MQInput node:
- Soccer Results sample

Configuring the MQInput node

When you have put an instance of the MQInput node into a message flow, you can configure it. To display its properties, either double-click the node, or right-click the node and click **Properties**. All mandatory properties that do not have a default value defined are marked with an asterisk.

Configure the MQInput node as follows:
1. Optional: On the Description tab, enter a short description, a long description, or both. You can also rename the node on this tab.

2. On the Basic tab, enter the name of the queue from which the message flow receives messages. You must predefine this WebSphere MQ queue to the queue manager that hosts the broker to which the message flow is deployed.
3. On the Advanced tab, set properties that determine how the message is processed, such as its transactional characteristics. Many of these properties map to options on the MQGET call.
   - Select *Transaction Mode* from the drop-down list to define the transactional characteristics of how this message is handled:
     - If you select `Automatic`, the incoming message is received under syncpoint if it is marked persistent, otherwise it is not. The transactionality of any derived messages that are sent subsequently by an output node is determined by the incoming persistence property, unless the output node has overridden transactionality explicitly.
     - If you select `Yes`, the incoming message is received under syncpoint. Any derived messages that are sent subsequently by an output node in the same instance of the message flow are sent transactionally, unless the output node has overridden transactionality explicitly.
     - If you select `No`, the incoming message is not received under syncpoint. Any derived messages that are sent subsequently by an output node in the message flow are sent non-transactionally, unless the output node has specified that the messages must be put under syncpoint.
   - Select *Order Mode* from the drop-down list to determine the order in which messages are retrieved from the input queue. This property has an effect only if the message flow property *Additional Instances* is set to greater than zero; that is, if multiple threads read the input queue. Valid values are:
     - `Default`. Messages are retrieved in the order that is defined by the queue attributes, but this order is not guaranteed as the messages are processed by the message flow.
     - `By User ID`. Messages that have the same UserIdentifier in the MQMD are retrieved and processed in the order that is defined by the queue attributes; this order is guaranteed to be preserved when the messages are processed. A message that is associated with a particular UserIdentifier that is being processed by one thread, is completely processed before the same thread, or another thread, can start to process another message with the same UserIdentifier. No other ordering is guaranteed to be preserved.
     - `By Queue Order`. Messages are retrieved and processed by this node in the order that is defined by the queue attributes; this order is guaranteed to be preserved when the messages are processed. This behavior is identical to the behavior that is exhibited if the message flow property *Additional Instances* is set to zero.

     See "Configuring the node to handle message groups" on page 159 for more details about this option.
   - Select *Logical Order* to ensure that messages that are part of a message group are received in the order that has been assigned by the sending application. This option maps to the MQGMO_LOGICAL_ORDER option of the MQGMO of the MQI.

     If you clear the check box, messages that are sent as part of a group are not received in a predetermined order. If a broker expects to receive messages in groups, and this check box is not selected, either the order of the input messages is not significant, or the message flow must be designed to process them appropriately.

You must also select the *Commit by Message Group* check box if you want message processing to be committed only after the final message of a group has been received and processed.

More information about the options to which this property maps is available in the *WebSphere MQ Application Programming Reference*.

See “Configuring the node to handle message groups” on page 159 for more details about this option.

- Select *All Messages Available* if you want message retrieval and processing to be done only when all messages in a single group are available. This property maps to the MQGMO_ALL_MSGS_AVAILABLE option of the MQGMO of the MQI. Clear this check box if message retrieval does not depend on all messages in a group being available before processing starts.

  More information about the options to which this property maps is available in the *WebSphere MQ Application Programming Reference*.
- Enter a message identifier in *Match Message ID* if you want the input node to receive only messages that contain a matching message identifier value in the MsgId field of the MQMD.

  Enter an even number of hexadecimal digits (characters `0 to 9`, `A to F`, and `a to f` are valid) up to a maximum of 48 digits. If the ID that you enter is shorter than the size of the MsgId field, it is padded on the right with `X'00'` characters. This property maps to the MQMO_MATCH_MSG_ID option of the MQGMO of the MQI.

  Leave this property blank if you do not want the input node to check that the message ID matches.

  More information about the options to which this property maps is available in the *WebSphere MQ Application Programming Reference*.
- Enter a message identifier in *Match Correlation ID* if you want the input node to receive only messages that contain a matching correlation identifier value in the CorrelId field of the MQMD.

  Enter an even number of hexadecimal digits (characters `0 to 9`, `A to F`, and `a to f` are valid) up to a maximum of 48 digits. If the ID that you enter is shorter than the size of the CorrelId field, it is padded on the right with `X'00'` characters. This property maps to the MQMO_MATCH_CORREL_ID option of the MQGMO of the MQI.

  Leave this property blank if you do not want the input node to check that the message ID matches.

  More information about the options to which this property maps is available in the *WebSphere MQ Application Programming Reference*.
- Select *Convert* if you want WebSphere MQ to perform data conversion on the message when it is retrieved from the queue.

  WebSphere MQ converts the incoming message to the encoding and coded character set that is specified in the MQMD that the input node supplies on the MQGET call to retrieve the message from the input queue. The message flow generates all of its output messages using these values, and puts them to target queues with these Encoding and CodedCharSetID values set in the MQMD.

  This property maps to the MQGMO_CONVERT option of the MQGMO of the MQI.

  Clear the check box if you do not want WebSphere MQ to convert the message.

  If you select this box, you can also specify:

– *Convert Encoding*. Enter the number that represents the encoding to which you want to convert numeric data in the message body. Valid values include:
  - `546` for DOS and all Windows systems
  - `273` for all Linux and UNIX systems
  - `785` for z/OS systems

  If you do not specify a value, the value in the incoming message MQMD is used.

  If you specify an invalid value, no conversion is done.

– *Convert Coded Char Set ID*. Enter the number that represents the character set identifier to which you want to convert character data in the message body.

  If you do not specify a value, the value in the incoming message MQMD is used.

  If you specify an invalid value, no conversion is done.

For more information about WebSphere MQ data conversion, and why you might choose to use this option, see the *WebSphere MQ Application Programming Guide*. For further information about the values that you can specify for *Convert Encoding* and *Convert Coded Char Set ID*, see the *WebSphere MQ Application Programming Reference*.

- Select *Commit by Message Group* if you want message processing to be committed only after the final message of a group has been received and processed. If you leave this check box cleared, a commit is performed after each message has been propagated completely through the message flow.

  This property is relevant only if you have selected *Logical Order*.

  Set the *Order Mode* property to `By Queue Order` if the messages in a group must be retrieved and processed in the order in which they appear on the queue.

- z/OS On z/OSonly: Enter a serialization token in *z/OS Serialization Token* if you want to use the serialized access to shared resources that is provided by WebSphere MQ.

  The value that you provide for the serialization token must conform to the rules described in the *WebSphere MQ Application Programming Reference*.

  For more information about serialization and queue sharing on z/OS, refer to the *WebSphere MQ Concepts and Planning Guide*.

- You can optionally associate a message with a publish/subscribe topic using the Topic property. You can enter any characters as the topic name. When messages pass through the MQInput node, they assume whatever topic name you have entered. (If you are using publish/subscribe, you can subscribe to a topic and see any messages that passed through the MQInput node and were published under that topic name.) If the incoming message has an MQRFH2 header, you do not need to set a value for the Topic property because the value can be obtained from the `<psc>` folder in the MQRFH2 header; for example:

  ```
  <psc><Topic>stockquote</Topic></psc>
  ```

  If you set a Topic property value, and that value differs from the <Topic> value in the MQRFH2 header, the value in the MQRFH2 header takes precedence.

Connecting the terminals:

The MQInput node routes each message that it retrieves successfully to the Out terminal. If this fails, the message is retried. If the retry timeout expires (as defined by the BackoutThreshold attribute of the input queue), the message is routed to the Failure terminal; you can connect nodes to this terminal to handle this condition. If you have not connected the Failure terminal, the message is written to the backout queue.

If the message is caught by this node after an exception has been thrown further on in the message flow, the message is routed to the Catch terminal. If you have not connected the Catch terminal, the message loops continually through the node until the problem is resolved. You must define a backout queue or a dead-letter queue (DLQ) to prevent the message from looping continually through the node.

Configuring for coordinated transactions:

When you include an MQInput node in a message flow, the value that you set for *Transaction Mode* defines whether messages are received under syncpoint:

- If you set the property to `Yes` (the default), the message is received under syncpoint (that is, within a WebSphere MQ unit of work). Any messages that are sent subsequently by an output node in the same instance of the message flow are put under syncpoint, unless the output node has overridden this explicitly.
- If you set the property to `Automatic`, the message is received under syncpoint if the incoming message is marked as persistent; otherwise, it is not. Any message that is sent subsequently by an output node is put under syncpoint, as determined by the incoming persistence property, unless the output node has overridden this explicitly.
- If you set the property to `No`, the message is not received under syncpoint. Any messages that are sent subsequently by an output node in the message flow are not put under syncpoint, unless an individual output node has specified that the message must be put under syncpoint.

The MQOutput node is the only output node that you can configure to override this option.

Configuring the node to handle message groups:

WebSphere MQ supports message groups; you can specify that a message belongs to a group and that its processing and the processing of all other messages in the group must be handled as one transaction; that is, if the processing on one message in the group fails, all messages in the group are backed out. The message processing is committed when the last message in the group has been processed successfully, only if processing of all messages has been successful.

If you include messages in a group, and it is important that all of the messages within the group are read from the queue and processed in the order in which they are defined in the group, you must complete all the actions stated below:

- Select *Commit by Message Group*.
- Select *Logical Order*.
- Set the *Order Mode* to `By Queue Order` or set the message flow property *Additional Instances* to zero. (You can modify message flow properties when you add the message flow to the bar file for deployment.) If you choose either of these options (or both), the message flow processes the messages on a single thread of execution, and a message is processed to completion before the next message is retrieved from the queue. In all other cases, it is possible that

multiple threads within a single message flow are processing multiple messages, and there is no guarantee that the final message in a group, which prompts the commit or roll back action, is processed to completion after all other messages in the group.

You must also ensure that you do not have another message flow that is retrieving messages from the same input queue. If you do, there is no guarantee about the order in which the messages within a group are processed.

Terminals and properties

The terminals of the MQInput node are described in the following table.

Terminal	Description
Failure	The output terminal to which the message is routed if an error occurs. Even if the Validation property is set, messages propagated to this terminal are not validated.
Out	The output terminal to which the message is routed if it is successfully retrieved from the WebSphere MQ queue.
Catch	The output terminal to which the message is routed if an exception is thrown downstream and caught by this node.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the bar file to deploy it).

The Description properties of the MQInput node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, e.g. Input	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the MQInput node are described in the following table.

Property	M	C	Default	Description
Queue Name	Yes	Yes		The name of the WebSphere MQ input queue from which this node retrieves messages (using MQGET) for processing by this message flow.

The Advanced properties of the MQInput node are described in the following table.

Property	M	C	Default	Description
Transaction Mode	Yes	No	`Yes`	This property controls whether the incoming message is received under syncpoint. Valid values are `Automatic`, `Yes`, and `No`.

Property	M	C	Default	Description
Order Mode	Yes	No	`Default`	The order in which messages are retrieved from the input queue and processed. Valid values are `Default`, `By User ID`, and `By Queue Order`.
Logical Order	Yes	No	Selected	If you select this check box, messages are received in logical order, as defined by WebSphere MQ.
All Messages Available	Yes	No	Cleared	If you select the check box, all messages in a group must be available before retrieval of a message is possible.
Match Message ID	No	No		A message ID that must match the message ID in the MQMD of the incoming message.
Match Correlation ID	No	No		A correlation ID that must match the correlation ID in the MQMD of the incoming message.
Convert	Yes	No	Cleared	If you select this check box, WebSphere MQ converts data in the message to be received, in conformance with the CodedCharSetId and Encoding values set in the MQMD.
Convert Encoding	No	No		The representation used for numeric values in the message data, expressed as an integer value. This property is valid only if you have selected the *Convert* check box.
Convert Coded Character Set ID	No	No		The coded character set identifier of character data in the message data, expressed as an integer value. This property is valid only if you have selected the *Convert* check box.
Commit By Message Group	Yes	No	Cleared	This property controls when a transaction is committed when processing messages that are part of a message group. If you select the check box, the transaction is committed when the message group has been processed.
z/OS Serialization Token	No	No		A user-defined token for serialized application support. The value that is specified must conform to the rules for a valid ConnTag in the WebSphere MQ MQCNO structure. These rules are described in the *WebSphere MQ Application Programming Reference*.
Topic	No	Yes		The default topic for the input message.

MQJMSTransform node

This topic contains the following sections:
- "Purpose"
- "Using the MQJMSTransform node in a message flow" on page 162
- "Configuring the MQJMSTransform node" on page 162
- "Terminals and properties" on page 162

Purpose

Use the MQJMSTransform node to receive messages that have a WebSphere MQ JMS provider message tree format, and transform them into a format that is compatible with messages that are to be sent to JMS destinations.

Use the MQJMSTransform node to send messages to legacy message flows and to interoperate with WebSphere MQ JMS and WebSphere Event Broker publish/subscribe.

The MQJMSTransform node is represented in the workbench by the following icon:

Using the MQJMSTransform node in a message flow

The JMS Nodes sample contains a message flow in which the MQJMSTransform node is used. Refer to this sample for an example of how to use the MQJMSTransform node.

Configuring the MQJMSTransform node

When you have put an instance of the MQJMSTransform node into a message flow, you can configure it. To display its properties, either double-click the node, or right-click the node and click **Properties**.

Optional: On the Description tab, enter a short description, a long description, or both. You can also change the name of the node on this tab.

Terminals and properties

The terminals of the MQJMSTransform node are described in the following table:

Terminal	**Description**
Failure	The output terminal to which the message is routed if an error occurs. Even if the Validation property is set, messages propagated to this terminal are not validated.
Out	The output terminal to which the message is routed if it is successfully retrieved from the WebSphere MQ queue.
In	The input terminal that accepts a message for processing by the node.

The following table describes the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the bar file to deploy it).

The MQJMSTransform node Description properties are described in the following table:

Property	**M**	**C**	**Default**	**Description**
Node name	No	No	The node type	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

MQOptimizedFlow node

The MQOptimizedFlow node is a complete message flow that provides a high-performance publish/subscribe message flow. The node supports publishers and subscribers that use Java Message Service (JMS) application programming interfaces and the WebSphere MQ Enterprise Transport.

To take advantage of any performance gain that this node can provide, ensure that you have installed WebSphere MQ Version 5.3 Fix Pack 10 for distributed systems. Refer to the memo.ptf file for Fix Pack 10 for details of the JMS configuration that is required.

Restriction: The MQOptimizedFlow node cannot be used on z/OS systems.

This topic contains the following sections:
- "Purpose"
- "Using this node in a message flow"
- "Configuring the MQOptimizedFlow node"
- "Terminals and properties" on page 164

Purpose

Use the MQOptimizedFlow node to replace a publish/subscribe message flow that consists of an MQInput node that is connected to a Publication node and that uses the JMS over WebSphere MQ transport.

Use the MQOptimizedFlow node to improve performance, particularly where a single publisher produces a persistent publication for a single subscriber

The MQOptimizedFlow node is represented in the workbench by the following icon:

Using this node in a message flow

Use an MQOptimizedFlow node in a message flow to publish a persistent JMS message to a single subscriber.

The MQOptimizedFlow node has no terminals, so you cannot connect it to any other message flow node.

Configuring the MQOptimizedFlow node

When you have put an instance of the MQOptimizedFlow node into a message flow, you must configure it. To display its properties, either double-click the node, or right-click the node and click **Properties**.

All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

1. Optional: On the Description tab, enter a short description, a long description, or both. You can also rename the node on this tab.
2. On the Basic tab, specify in the *Queue Name* property the name of the WebSphere MQ input queue from which messages are retrieved.

3. On the Advanced tab, set the Transaction Mode property to Yes (the default) if you want the incoming message to be received under syncpoint.

Terminals and properties

The MQOptimizedFlow node has no terminals. It is a complete message flow and you cannot connect it to other message flow nodes to extend the message processing.

The following tables describe the node properties. The column headed M indicates whether the property is mandatory; that is, whether you must enter a value if no default value is defined; an asterisk next to the name of the property denotes this. The column headed C indicates whether the property is configurable; that is, whether you can change the value in the bar file.

The Description properties of the MQOptimizedFlow node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Basic properties of the MQOptimizedFlow node are described in the following table:

Property	M	C	Default	Description
Queue Name	Yes	Yes	none	The name of the WebSphere MQ input queue from which this node retrieves messages for processing by this message flow.

The Advanced properties of the MQOptimizedFlow node are described in the following table:

Property	M	C	Default	Description
Transaction Mode	Yes	No	Yes	This property controls whether the incoming message is received under syncpoint. Valid values are `Automatic`, `Yes`, and `No`.

MQOutput node

This topic contains the following sections:
- "Purpose" on page 165
- "Using this node in a message flow" on page 165
- "Configuring the MQOutput node" on page 165
- "Terminals and properties" on page 168

Purpose

Use the MQOutput node to send messages to clients that connect to the broker using the WebSphere MQ Enterprise Transport and that use the MQI and AMI application programming interfaces.

The MQOutput node delivers an output message from a message flow to a WebSphere MQ queue. The node uses MQPUT to put the message to the destination queue that you specify.

If appropriate, define the queue as a WebSphere MQ clustered queue or shared queue. When you use a WebSphere MQ clustered queue, leave the queue manager name empty.

You can configure the MQOutput node to put a message to a specific WebSphere MQ queue that is defined on any queue manager that is accessible by the broker's queue manager.

Set other properties to control the way in which messages are sent, by causing appropriate MQPUT options to be set; for example, you can request that a message is processed under transaction control. You can also specify that WebSphere MQ can, if appropriate, break the message into segments in the queue manager.

If you create a message flow to use as a subflow, you cannot use a standard output node; use an instance of the Output node to create an Out terminal for the subflow through which to propagate the message.

If you do not want your message flow to send messages to a WebSphere MQ queue, choose another supported output node.

The MQOutput node is represented in the workbench by the following icon:

Using this node in a message flow

For an example of how to use this node, assume that you have written a publishing application that publishes stock updates on a regular basis. The application sends the messages to the broker on an MQInput node, and the message flow makes the publications available to multiple subscribers through a Publication node. You include an MQOutput node to send the message to an application that records each price change that occurs.

Configuring the MQOutput node

When you have put an instance of the MQOutput node into a message flow, you can configure it. To display its properties, either double-click the node, or right-click the node and click **Properties**. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

Configure the MQOutput node as follows:

1. Optional: On the Description tab, enter a short description, a long description, or both. You can also rename the node on this tab.

2. On the Basic tab:
   a. To send the output message to a single destination queue that is defined by this node, enter the name of the queue to which the message flow sends messages in *Queue Name*.
   b. Enter the name of the queue manager to which this queue is defined in *Queue Manager Name*. You must set these properties if you set the *Destination Mode* property on the Advanced tab (described below) to `Queue Name`. If you set *Destination Mode* to another value, these properties are ignored.
3. The properties on the Advanced tab define the transactional control for the message and the way in which the message is put to the queue. Many of these properties map to options on the MQPUT call.
   - Select the *Destination Mode* from the drop-down list. This property identifies the queues to which the output message is put:
     – `Queue Name` (the default). The message is sent to the queue that is named in the *Queue Name* property. The *Queue Manager Name* and *Queue Name* properties on the Basic tab are mandatory if you select this option.
     – `Reply To Queue`. The message is sent to the queue that is named in the ReplyToQ field in the MQMD.
   - Select the *Transaction Mode* from the drop-down list to determine how the message is put:
     – If you select `Automatic` (the default), the message transactionality is derived from the way that it was specified at the input node.
     – If you select `Yes`, the message is put transactionally.
     – If you select `No`, the message is put non-transactionally.

     See "Configuring for coordinated transactions" on page 168 for more information.
   - Select the *Persistence Mode* from the drop-down list to determine whether the message is put persistently:
     – If you select `Automatic` (the default), the persistence is as specified in the incoming message.
     – If you select `Yes`, the message is put persistently.
     – If you select `No`, the message is put non-persistently.
     – If you select `As Defined for Queue`, the message persistence is set as defined for the WebSphere MQ queue.
   - Select *New Message ID* to generate a new message ID for this message. This property maps to the MQPMO_NEW_MSG_ID option of the MQPMO of the MQI.

     Clear the check box if you do not want to generate a new ID. A new message ID is still generated if you select *Request* on the Request tab.

     More information about the options to which this property maps is available in the *WebSphere MQ Application Programming Reference*.
   - Select *New Correlation ID* to generate a new correlation ID for this message. This property maps to the MQPMO_NEW_CORREL_ID option of the MQPMO of the MQI. Clear the check box if you do not want to generate a new ID.

     More information about the options to which this property maps is available in the *WebSphere MQ Application Programming Reference*.
   - Select *Segmentation Allowed* if you want WebSphere MQ to segment the message within the queue manager when appropriate. You must also set

`MQMF_SEGMENTATION_ALLOWED` in the MsgFlags field in the MQMD for segmentation to occur. Clear the check box if you do not want segmentation to occur.

  More information about the options to which this property maps is available in the *WebSphere MQ Application Programming Reference*.
- Select the *Message Context* to indicate how origin context is handled. Choose one of the following options:
  - `Pass All` maps to the MQPMO_PASS_ALL_CONTEXT option of the MQPMO of the MQI.
  - `Pass Identity` maps to the MQPMO_PASS_IDENTITY_CONTEXT option of the MQPMO of the MQI.
  - `Set All` maps to the MQPMO_SET_ALL_CONTEXT option of the MQPMO of the MQI.
  - `Set Identity` maps to the MQPMO_SET_IDENTITY_CONTEXT option of the MQPMO of the MQI.
  - `Default` maps to the MQPMO_DEFAULT_CONTEXT option of the MQPMO of the MQI.
  - `None` maps to the MQPMO_NO_CONTEXT option of the MQPMO of the MQI.

  More information about the options to which these properties map is available in the *WebSphere MQ Application Programming Reference*.
- Select *Alternate User Authority* if you want the MQOO_ALTERNATE_USER_AUTHORITY option to be set in the open options (MQOO) of the MQI. If you select this check box, this option is specified when the queue is opened for output. The alternate user information is retrieved from the context information in the message. Clear the check box if you do not want to specify alternate user authority. If you clear the check box, the broker service user ID is used when the message is put.

4. On the Request tab, set the properties to define the characteristics of each output message generated.
   - Select *Request* to mark each output message in the MQMD as a request message (MQMT_REQUEST), and clear the message identifier field (which is set to MQMI_NONE) to ensure that WebSphere MQ generates a new identifier. Clear the check box to indicate that each output message is not marked as a request message. You cannot select this check box if you have selected a *Destination Mode* of `Reply To Queue`.

     A new message identifier is generated even if the *New Message ID* check box is not selected on the Advanced tab.
   - Enter a queue manager name in *Reply-to Queue Manager*. This name is inserted into the MQMD of each output message as the reply-to queue manager.
   - Enter a queue name in *Reply-to Queue*. This name is inserted into the MQMD of each output message as the reply-to queue.

Connecting the terminals:

Connect the In terminal to the node from which outbound messages bound are routed.

Connect the Out or Failure terminal of this node to another node in this message flow to send the message to an additional destination.

Configuring for coordinated transactions:

When you define an MQOutput node, the option that you select for the *Transaction Mode* property defines whether the message is written under syncpoint:

- If you select `Yes`, the message is written under syncpoint (that is, within a WebSphere MQ unit of work).
- If you select `Automatic` (the default), the message is written under syncpoint if the incoming input message is marked as persistent.
- If you select `No`, the message is not written under syncpoint.

Another property of the MQOutput node, *Persistence Mode*, defines whether the output message is marked as persistent when it is put to the output queue:

- If you select `Yes`, the message is marked as persistent.
- If you select `Automatic` (the default), the message persistence is determined from the properties of the incoming message, as set in the MQMD (the WebSphere MQ message descriptor).
- If you select `No`, the message is not marked as persistent.
- If you select `As Defined for Queue`, the message persistence is set as defined in the WebSphere MQ queue by the MQOutput node specifying the MQPER_PERSISTENCE_AS_Q_DEF option in the MQMD.

Terminals and properties

The MQOutput node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the message is routed if a failure is detected when the message is put to the output queue.
Out	The output terminal to which the message is routed if it has been successfully put to the output queue, and if further processing is required within this message flow.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the bar file to deploy it).

The MQOutput node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, e.g. Input	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The MQOutput node Basic properties are described in the following table.

Property	M	C	Default	Description
Queue Manager Name	No	Yes		The name of the WebSphere MQ queue manager to which the output queue, which is specified in *Queue Name*, is defined.
Queue Name	No	Yes		The name of the WebSphere MQ output queue to which this node puts messages (using MQPUT).

The MQOutput node Advanced properties are described in the following table.

Property	M	C	Default	Description
Destination Mode	Yes	No	`Queue Name`	The queues to which the output message is sent. Valid values are , `Reply To Queue`, and `Queue Name`.
Transaction Mode	Yes	No	`Automatic`	This property controls whether the message is put transactionally. Valid values are `Automatic`, `Yes`, and `No`.
Persistence Mode	Yes	No	`Automatic`	This property controls whether the message is put persistently. Valid values are `Automatic`, `Yes`, `No`, and `As Defined for Queue`.
New Message ID	Yes	No	Cleared	If you select this check box, WebSphere MQ generates a new message identifier to replace the contents of the MsgId field in the MQMD.
New Correlation ID	Yes	No	Cleared	If you select this check box, WebSphere MQ generates a new correlation identifier to replace the contents of the CorrelId field in the MQMD.
Segmentation Allowed	Yes	No	Cleared	If you select this check box, when appropriate, WebSphere MQ breaks the message into segments in the queue manager.
Message Context	Yes	No	`Pass All`	This property controls how origin context is handled. Valid values are `Pass All`, `Pass Identity`, `Set All`, `Set Identity`, and `Default`.
Alternate User Authority	Yes	No	Cleared	If you select this check box, alternate authority is used when the output message is put.

The MQOutput node Request properties are described in the following table.

Property	M	C	Default	Description
Request	Yes	No	Cleared	If you select the check box, each output message is generated as a request message.
Reply-to Queue Manager	No	Yes		The name of the WebSphere MQ queue manager to which the output queue, which is specified in *Reply-to Queue*, is defined.
Reply-to Queue	No	Yes		The name of the WebSphere MQ queue to which to put a reply to this request.

Output node

This topic contains the following sections:
- "Purpose" on page 170
- "Configuring the Output node" on page 170
- "Terminals and properties" on page 170

Purpose

The Output node provides an out terminal for an embedded message flow (a subflow). You can use a subflow to provide a common destination for output messages.

You must use an Output node to provide the Out terminal to a subflow; you cannot use a standard output node (a built-in output node such as MQOutput, or a user-defined output node).

You can include one or more Output nodes in a subflow. Each one that you include provides a terminal through which you can propagate messages to subsequent nodes in the message flow in which you include the subflow.

The Output node is represented in the workbench by the following icon:

When you select and include a subflow in a message flow, it is represented by the icon:

When you include the subflow in a message flow, this icon exhibits a terminal for each Output node that you included in the subflow, and the name of the terminal (which you can see when you hover over it) matches the name of that instance of the Output node. Give your Output nodes meaningful names, you can easily recognize them when you use their corresponding terminal on the subflow node in your message flow.

Configuring the Output node

When you have put an instance of the Output node into a message flow, you can configure it. To display its properties, either double-click the node, or right-click the node and click **Properties**.

Optional: On the Description tab, enter a short description, a long description, or both. You can also rename the node on this tab.

Terminals and properties

The Output node terminals are described in the following table.

Terminal	Description
In	The output terminal that defines an out terminal for the subflow.

The following table describes the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the bar file to deploy it).

The Output node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, e.g. Input	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

Publication node

This topic contains the following sections:
- “Purpose”
- “Using this node in a message flow”
- “Configuring the Publication node” on page 172
- “Terminals and properties” on page 172

Purpose

Use the Publication node to filter output messages from a message flow and transmit them to subscribers who have registered an interest in a particular set of topics. The Publication node must always be an output node of a message flow and has no output terminals of its own.

Use the Publication node (or a user-defined node that provides a similar service) if your message flow supports publish/subscribe applications. Applications that expect to receive publications must register a subscription with a broker, and can optionally qualify the publications that they get by providing restrictive criteria (such as a specific publication topic).

If your subscriber applications use the WebSphere MQ Enterprise Transport to connect to the broker, you can define the queues to which messages are published as WebSphere MQ clustered queues or shared queues.

Publications can also be sent to subscribers within a WebSphere MQ cluster if a cluster queue is nominated as the subscriber queue. In this case, the subscriber should use the name of an ″imaginary″ queue manager that is associated with the cluster, and should ensure that a corresponding blank queue manager alias definition for this queue manager is made on the broker that satisfies the subscription.

The Publication node is represented in the workbench by the following icon:

Using this node in a message flow

Look at the following sample to see how to use this node:
- Soccer Results sample

For an example of how to use this node, assume that you have written a publishing application that publishes stock updates on a regular basis. The application sends the messages to the broker on an MQInput node, and the stock publications are made available to multiple subscribers through a Publication node.

Configuring the Publication node

When you have put an instance of the Publication node into a message flow, you can configure it. To display its properties, either double-click the node, or right-click the node and click **Properties**. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

Configure the Publication node as follows:

1. On the Description tab. enter a short description, a long description, or both. You can also rename the node on this tab.
2. On the Basic tab:
   a. Select *Implicit Stream Naming* to take the name of the WebSphere MQ queue on which the message was received by the message flow as the stream name. This property provides forward compatibility with WebSphere MQ Publish/Subscribe, and applies to messages with an MQRFH header when MQPSStream is not specified.

      Clear the check box if you do not want this action to be taken.

   b. Specify the *Subscription Point* for this Publication node. If you do not specify a value for this property, the default subscription point is assumed. This value uniquely identifies the node, and can be used by subscribers to get a specific publication (as described in the example scenario above).

      For more information, refer to Subscription points.

Terminals and properties

The Publication node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined), the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the bar file to deploy it).

The Publication node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, e.g. Input	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Publication node Basic properties are described in the following table.

Property	M	C	Default	Description
Implicit Stream Naming	Yes	No	Cleared	If you select this check box, the name of the WebSphere MQ queue on which the input message was received is taken as the stream name.
Subscription Point	No	No		The subscription point value for the node.

Real-timeInput node

This topic contains the following sections:
- "Purpose"
- "Using this node in a message flow"
- "Configuring the Real-timeInput node"
- "Terminals and properties" on page 174

Purpose

Use the Real-timeInput node to receive messages from clients that connect to the broker using the WebSphere MQ Real-time Transport or the WebSphere MQ Multicast Transport, and that use JMS application programming interfaces.

An output node in a message flow that starts with a Real-timeInput node can be any of the supported output nodes, including user-defined output nodes. You can create a message flow that receives messages from real-time clients and generates messages for clients that use all supported transports to connect to the broker, because you can configure the message flow to request the broker to provide any conversion that is required.

If you create a message flow to use as a subflow, you cannot use a standard input node; you must use an instance of the Input node as the first node to create an In terminal for the subflow.

If your message flow does not receive messages from JMS applications, choose one of the supported input nodes.

The Real-timeInput node is represented in the workbench by the following icon:

Using this node in a message flow

For an example of how to use this node, assume that you have written a publishing application that publishes stock updates on a regular basis. The application sends the messages to the broker on a Real-timeInput node, and the message flow makes the publications available to multiple subscribers through a Publication node.

Configuring the Real-timeInput node

When you have put an instance of the Real-timeInput node into a message flow, you can configure it. To display its properties, either double-click the node, or right-click the node and click **Properties**.

All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

Configure the Real-timeInput node as follows:

1. Optional: On the Description tab, enter a short description, a long description, or both. You can also rename the node on this tab.
2. On the Basic tab:
   a. In *Port*, identify the number of the port on which the node listens for publish or subscribe requests from JMS applications. Ensure that the port number that you specify does not conflict with any other listener service. No default value is provided for this property; you must enter a value.
   b. To authenticate users that send messages on receipt of their messages, select *Authentication*. If you clear the check box (the default setting), users are not authenticated.
   c. For clients to use HTTP tunneling, select *Tunnel through HTTP*. If you clear the check box (the default setting), messages do not use HTTP tunneling. If you select the check box, all client applications that connect must use this feature. If they do not use this feature, their connection is rejected. The client application cannot use this option in conjunction with the connect-via proxy setting, which is activated from the client side.
   d. In *Read Threads*, enter the number of threads that you want the broker to allocate to read messages. The broker starts as many instances of the message flow as are necessary to process current messages, up to this limit. The default setting is `10`.
   e. In *Write Threads*, enter the number of threads that you want the broker to allocate to write messages. The broker starts as many instances of the message flow as are necessary to process current messages, up to this limit. The default setting is `10`.
   f. In *Authentication Threads*, enter the number of threads that you want the broker to allocate to user authentication checks. The user authentication check is performed when a message is received. The broker starts as many instances of the message flow as are necessary to process current messages, up to this limit. The default setting is `10`.

Connecting the terminals:

The Real-timeInput node routes each message that it retrieves successfully to the Out terminal. If this routing fails, the message is retried.

Terminals and properties

The Real-timeInput node terminals are described in the following table.

Terminal	Description
Out	The output terminal to which the message is routed if it is successfully retrieved from JMS.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined), the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the bar file to deploy it).

The Real-timeInput node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Real-timeInput node Basic properties are described in the following table.

Property	M	C	Default	Description
Port	Yes	Yes	0	The port number on which the input node listens for publish or subscribe requests.
Authentication	Yes	No	Cleared	If you select this check box, users are authenticated.
Tunnel through HTTP	Yes	No	Cleared	Select the check box to indicate that users use HTTP tunneling. Clear the check box to indicate that HTTP tunneling is not used.
Read Threads	No	Yes	10	The number of threads used for reading.
Write Threads	No	Yes	10	The number of threads used for writing.
Authentication Threads	No	Yes	10	The number of threads used for accepting connections and authenticating users.

The properties of the General Message Options for the Real-timeInput node are described in the following table.

Real-timeOptimizedFlow node

This topic contains the following sections:
- "Purpose"
- "Using this node in a message flow" on page 176
- "Configuring the Real-timeOptimizedFlow node" on page 176
- "Terminals and properties" on page 177

Purpose

Use the Real-timeOptimizedFlow node to receive messages from clients that connect to the broker using the WebSphere MQ Real-time Transport or the WebSphere MQ Multicast Transport, and that use JMS application programming interfaces.

The Real-timeOptimizedFlow node is a complete message flow that provides a high performance publish/subscribe message flow. The actions that are taken by this node are all internalized; you cannot influence the node's operation except by configuring its properties, and you cannot connect it to any other node.

This node also supports publication to, or subscription from, standard WebSphere MQ applications, but its performance for these applications is not as good as the performance achieved for JMS applications.

The Real-timeOptimizedFlow node is represented in the workbench by the following icon:

Using this node in a message flow

Include the Real-timeOptimizedFlow node in a message flow when you want to distribute messages through a broker to and from client applications that use JMS.

Configuring the Real-timeOptimizedFlow node

When you have put an instance of the Real-timeOptimizedFlow node into a message flow, you can configure it. To display its properties, either double-click the node, or right-click the node and click **Properties**.

All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

Configure the Real-timeOptimizedFlow node as follows:

1. Optional: On the Description tab, enter a short description, a long description, or both. You can also rename the node on this tab.
2. On the Basic tab:
   a. In *Port*, identify the number of the port on which the node listens for publish or subscribe requests from JMS applications. Ensure that the port number that you specify does not conflict with any other listener service. No default value is provided for this property; you must enter a value.
   b. For users to authenticate that send messages on receipt of their messages, select *Authentication*. If you clear the check box (the default setting), users are not authenticated.
   c. For clients to use HTTP tunneling, select *Tunnel through HTTP*. If you clear the check box (the default setting), messages do not use HTTP tunneling. If you select the check box, all client applications that connect must use this feature. If they do not use this feature, their connection is rejected. The client application cannot use this option in conjunction with the connect-via-proxy setting, which is activated from the client side.
   d. In *Read Threads*, enter the number of threads that you want the broker to allocate to read messages. The broker starts as many instances of the message flow as are necessary to process current messages, up to this limit. The default setting is `10`.
   e. In *Write Threads*, enter the number of threads that you want the broker to allocate to write messages. The broker starts as many instances of the message flow as are necessary to process current messages, up to this limit. The default setting is `10`.
   f. In *Authentication Threads*, enter the number of threads that you want the broker to allocate to user authentication checks. The user authentication check is performed when a message is received. The broker starts as many instances of the message flow as are necessary to process current messages, up to this limit. The default setting is `10`.

Terminals and properties

The Real-timeOptimizedFlow node has no terminals. It is a complete message flow and cannot be connected to other nodes to extend the message processing.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined), the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the bar file to deploy it).

The Real-timeOptimizedFlow node Description properties are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The Real-timeOptimizedFlow node Basic properties are described in the following table.

Property	M	C	Default	Description
Port	Yes	Yes		The port number on which the node listens for publish or subscribe requests. You must provide a value for this property.
Authentication	Yes	No	Cleared	If you select this check box, users are authenticated.
Tunnel through HTTP	Yes	No	Cleared	If you select this check box, clients use HTTP tunneling. Clear the check box to indicate that HTTP tunneling is not used.
Read Threads	No	Yes	`10`	The number of threads used for reading.
Write Threads	No	Yes	`10`	The number of threads used for writing.
Authentication Threads	No	Yes	`10`	The number of threads used for accepting connections and authenticating users.

SCADAInput node

This topic contains the following sections:
- "Purpose"
- "Using this node in a message flow" on page 178
- "Configuring the SCADAInput node" on page 178
- "Terminals and properties" on page 180

Purpose

Use the SCADAInput node to receive messages from clients that connect to the broker across the WebSphere MQ Telemetry Transport. SCADA device clients use the MQIsdp protocol to send messages, which are converted by the SCADAInput node into a format that is recognized by WebSphere Event Broker. The node also establishes the processing environment for these messages.

Message flows that handle messages that are received from SCADA devices must always start with a SCADAInput node. Set the SCADAInput node's properties to control the way in which messages are received; for example, you can indicate that a message is to be processed under transaction control.

When you deploy message flows that contain SCADA nodes to a broker, deploy them to a single execution group, regardless of the number of message flows.

SCADA is primarily publish/subscribe, so you typically include a Publication node to terminate the flow. In scenarios where you do not want to use a Publication node, include a SCADAOutput node. If you include a SCADAOutput node, you must also include a SCADAInput node, regardless of the source of the messages, because the SCADAInput node provides the connectivity information that is required by the SCADAOutput node.

If you include an output node in a message flow that starts with a SCADAInput node, it can be any of the supported output nodes, including user-defined output nodes. You can create a message flow that receives messages from SCADA devices, and generates messages for clients that use all supported transports to connect to the broker, because you can configure the message flow to request the broker to provide any necessary conversion.

You can request that the broker starts or stops a SCADA listener by publishing messages with a specific topic. This can be done for all ports or for a single port that is identified in the message.

z/OS You cannot use SCADAInput nodes in message flows that are to be deployed on z/OS systems.

If you create a message flow to use as a subflow, you cannot use a standard input node; you must use an instance of the Input node as the first node to create an In terminal for the subflow.

If your message flow does not receive messages across SCADA connections, choose one of the supported input nodes.

The SCADAInput node is represented in the workbench by the following icon:

Using this node in a message flow

For an example of how to use this node, assume that you create a message flow with a SCADAInput node that receives messages from a remote sensor when it detects a change in its operating environment (for example, a drop in outside temperature). You connect the node to an MQOutput node, which makes these messages available on a queue that is serviced by a WebSphere MQ application that analyses and responds to the information that is received.

Configuring the SCADAInput node

When you have put an instance of the SCADAInput node into a message flow, you can configure it. To display its properties, either double-click the node, or right-click the node and click **Properties**. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

Configure the SCADAInput node as follows:

1. Optional: On the Description tab, enter a short description, a long description, or both. You can also rename the node on this tab.
2. On the Basic tab, set the following properties:
   a. The *Enable listener on startup* check box is initially selected, which means that the listener for MQIsdp clients is initialized when the message flow is deployed.

      You can update the status of the listener by publishing on the control topic `$SYS/SCADA/MQIsdpListener/<port_number>` with the Payload part of the message set to `ON` or `OFF`.
   b. Specify the *Port* number on which the MQIsdp server listens. This value must be a unique port number, and must not conflict with other listeners (for example, those set up for WebSphere MQ or WebSphere MQ Everyplace). The default number is `1883`.
   c. Set the *Max Threads* value to indicate the maximum number of threads available to the MQIsdp server to support clients. The default value is `500`.

      If you are using DB2 as your broker database, specify a value that is less than or equal to the value that you have set for the DB2 configuration parameters *maxappls* and *maxagents*. See Enabling connections to the databases for further information.
   d. Select *Use Thread Pooling* if you want the node to use a pool of threads to service clients. If you select this option, the number of threads that are available to the MQIsdp server is limited by *Max Threads*, which is most effective when set to a value between 20 and 40. If you do not select this option, a new thread is created for each client that connects. The check box is cleared initially.

      Use this option only if you expect a large number of clients (greater than 200) to connect.
3. On the Advanced tab, set the required value for *Transaction Mode* to define the transactional characteristics of how this message is handled:
   - If you select `Automatic`, the incoming message is received under syncpoint if it is marked as persistent; otherwise, it is not. The transactionality of any derived messages that are sent subsequently by an output node is determined by the incoming persistence property, unless the output node has overridden transactionality explicitly.
   - If you select `Yes`, the incoming message is received under syncpoint. Any derived messages that are sent subsequently by an output node in the same instance of the message flow are sent transactionally, unless the output node has overridden transactionality explicitly.
   - If you select `No`, the incoming message is not received under syncpoint. Any derived messages that are sent subsequently by an output node in the flow are sent non-transactionally, unless the output node has specified that the message should be put under syncpoint.

Connecting the terminals:

The SCADAInput node routes each message that it retrieves successfully to the Out terminal. If this fails, the message is propagated to the Failure terminal; you can connect nodes to this terminal to handle this condition. If you have not connected the Failure terminal, the message loops continually through the node until the problem is resolved.

If the message is caught by this node after an exception has been thrown further on in the message flow, the message is routed to the Catch terminal. If you have not connected the Catch terminal, the message loops continually through the node until the problem is resolved. Ensure that a node is always connected to this terminal if there is the possibility of the message rolling back within a message flow.

Configuring for coordinated transactions:

When you include a SCADAInput node in a message flow, the value that you set for *Transaction Mode* defines whether messages are received under syncpoint:

- If you set this property to `Yes` (the default), the message is received under syncpoint (that is, within a WebSphere MQ unit of work). Any messages that are sent subsequently by an output node in the same instance of the message flow are put under syncpoint, unless the output node has overridden this explicitly.
- If you set this property to `Automatic`, the message is received under syncpoint if the incoming message is marked as persistent; otherwise, it is not. Any message that is sent subsequently by an output node is put under syncpoint, as determined by the incoming persistence property, unless the output node has overridden this explicitly.
- If you set this property to `No`, the message is not received under syncpoint. Any messages that are sent subsequently by an output node in the message flow are not put under syncpoint, unless an individual output node has specified that the message should be put under syncpoint.

The MQOutput node is the only output node that you can configure to override this option.

Terminals and properties

The SCADAInput node terminals are described in the following table.

Terminal	Description
Failure	The output terminal to which the message is routed if an error occurs.
Out	The output terminal to which the message is routed if it is successfully retrieved from the queue.
Catch	The output terminal to which the message is routed if an exception is thrown downstream and caught by this node.

The following tables describe the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the bar file to deploy it).

The Description properties of the SCADAInput node are described in the following table.

Property	M	C	Default	Description
Node name	No	No	The node type, e.g. Input	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

The SCADAInput node Basic properties are described in the following table.

Property	M	C	Default	Description
Enable listener on startup	Yes	No	Selected	This property controls when the listener is started. If you select the check box, the listener starts when the message flow is started by the broker. If you clear the check box, the listener starts on the arrival of a message on the specified port.
Port	Yes	Yes	`1883`	The port on which the SCADA protocol is listening.
Max Threads	Yes	Yes	`500`	The maximum number of threads to be started to support SCADA devices.
Use Thread Pooling	Yes	Yes	Cleared	If you select the check box, thread pooling is used.

The SCADAInput node Advanced property are described in the following table.

Property	M	C	Default	Description
Transaction Mode	Yes	No	`Yes`	This property controls whether the incoming message is received under syncpoint. Valid values are `Automatic`, `Yes`, and `No`.

SCADAOutput node

This topic contains the following sections:
- "Purpose"
- "Using this node in a message flow" on page 182
- "Configuring the SCADAOutput node" on page 182
- "Terminals and properties" on page 182

Purpose

Use the SCADAOutput node to send a message to a client that connects to the broker using the MQIsdp protocol across the WebSphere MQ Telemetry Transport. You would typically use the Publication node to send output to a SCADA client. The SCADAOutput node lets you write your own Publication node.

If you include a SCADAOutput node in a message flow, also include a SCADAInput node, regardless of the source of the messages, because the SCADAInput node provides the connectivity information that is required by the SCADAOutput node.

When you deploy message flows that contain SCADA nodes to a broker, deploy them to a single execution group, regardless of the number of message flows.

You cannot use the SCADAOutput node to change the transactional characteristics of the message flow. The transactional characteristics that are set by the message flow's input node determine the transactional behavior of the flow.

z/OS You cannot use SCADAOutput nodes in message flows that you deploy to z/OS systems.

If you create a message flow to use as a subflow, you cannot use a standard output node; use an instance of the Output node to create an out terminal for the subflow through which the message can be propagated.

If you do not want your message flow to send messages to a SCADA device, choose another supported output node.

The SCADAOutput node is represented in the workbench by the following icon:

Using this node in a message flow

Use the Publication node to publish messages for SCADA devices. Use this node to process the publication messages in a particular way for these devices.

Configuring the SCADAOutput node

When you have put an instance of the SCADAOutput node into a message flow, you can configure it. To display its properties, either double-click the node, or right-click the node and click **Properties**. All mandatory properties for which you must enter a value (those that do not have a default value defined) are marked with an asterisk.

Configure the SCADAOutput node as follows:

1. Optional: On the Description tab, enter a short description, a long description, or both. You can also rename the node on this tab.

Connecting the terminals:

Connect the In terminal to the node from which messages that are bound for SCADA destinations are routed.

Connect the Out or Failure terminal of this node to another node in this message flow to send the message to an additional destination.

Terminals and properties

The SCADAOutput node terminals are described in the following table.

Terminal	Description
In	The input terminal that accepts a message for processing by the node.
Failure	The output terminal to which the message is routed if a failure is detected when the message is put to the output queue.
Out	The output terminal to which the message is routed if it has been successfully put to the output queue, and if further processing is required within this message flow.

The following table describes the node properties. The column headed M indicates whether the property is *mandatory* (marked with an asterisk if you must enter a value when no default is defined); the column headed C indicates whether the property is *configurable* (you can change the value when you add the message flow to the bar file to deploy it).

The SCADAOutput node Description properties are described in the following table.

Property	**M**	**C**	**Default**	**Description**
Node name	No	No	The node type, e.g. Input	The name of the node.
Short Description	No	No		A brief description of the node.
Long Description	No	No		Text that describes the purpose of the node in the message flow.

User-defined nodes

You can deploy user-defined nodes that are created and supplied by WebSphere Message Broker Version 6.0 users or by independent software vendors and other companies. Use these nodes in WebSphere Event Broker message flows, to add to the function provided by the supplied or built-in nodes.

WebSphere Message Broker Version 6.0 users and other vendors can provide help information for user-defined nodes. If help information has been provided, it is displayed after this topic when you install the user-defined node.

Supported code pages

Application messages must conform to supported code pages.

The message flows that you create, configure, and deploy to a broker can process application messages in any code page that is listed in the table. You can also generate a new code page converter.

For detailed information about Chinese code page GB18030 support, see "Chinese code page GB18030" on page 210.

By default, WebSphere Event Broker supports the code pages that are given in the following tables. To find a code page for a specific CCSID, search for an internal converter name in the form ibm-*ccsid*, where *ccsid* is the CCSID for which you are looking.

- Unicode converters
- European and American language converters
- Asian language converters
- Windows US and European converters
- MAC-related converters
- Hebrew, Cyrillic and ECMA language converters
- Indian language converters
- EBCDIC converters

Unicode converters

Internal converter name	Aliases
UTF-8	UTF-8 ibm-1208 ibm-1209 ibm-5304 ibm-5305 windows-65001 cp1208
UTF-16	UTF-16 ISO-10646-UCS-2 unicode csUnicode ucs-2
UTF-16BE	UTF-16BE x-utf-16be ibm-1200 ibm-1201 ibm-5297 ibm-13488 ibm-17584 windows-1201 cp1200 cp1201 UTF16_BigEndian
UTF-16LE	UTF-16LE x-utf-16le ibm-1202 ibm-13490 ibm-17586 UTF16_LittleEndian windows-1200
UTF-32	UTF-32 ISO-10646-UCS-4 csUCS4 ucs-4
UTF-32BE	UTF-32BE UTF32_BigEndian ibm-1232 ibm-1233
UTF-32LE	UTF-32LE UTF32_LittleEndian ibm-1234
UTF16_PlatformEndian	UTF16_PlatformEndian

Internal converter name	Aliases
UTF16_OppositeEndian	UTF16_OppositeEndian
UTF32_PlatformEndian	UTF32_PlatformEndian
UTF32_OppositeEndian	UTF32_OppositeEndian
UTF-7	UTF-7 windows-65000
IMAP-mailbox-name	IMAP-mailbox-name
SCSU	SCSU
BOCU-1	BOCU-1 csBOCU-1
CESU-8	CESU-8

European and American language converters

Internal converter name	Aliases
ISO-8859-1	ISO-8859-1 ibm-819 IBM819 cp819 latin1 8859_1 csISOLatin1 iso-ir-100 ISO_8859-1:1987 l1 819
US-ASCII	US-ASCII ASCII ANSI_X3.4-1968 ANSI_X3.4-1986 ISO_646.irv:1991 iso_646.irv:1983 ISO646-US us csASCII iso-ir-6 cp367 ascii7 646 windows-20127
gb18030	gb18030 ibm-1392 windows-54936
ibm-367_P100-1995	ibm-367_P100-1995 ibm-367 IBM367

Internal converter name	Aliases
ibm-912_P100-1995	ibm-912_P100-1995 ibm-912 iso-8859-2 ISO_8859-2:1987 latin2 csISOLatin2 iso-ir-101 l2 8859_2 cp912 912 windows-28592
ibm-913_P100-2000	ibm-913_P100-2000 ibm-913 iso-8859-3 ISO_8859-3:1988 latin3 csISOLatin3 iso-ir-109 l3 8859_3 cp913 913 windows-28593
ibm-914_P100-1995	ibm-914_P100-1995 ibm-914 iso-8859-4 latin4 csISOLatin4 iso-ir-110 ISO_8859-4:1988 l4 8859_4 cp914 914 windows-28594

Internal converter name	Aliases
ibm-915_P100-1995	ibm-915_P100-1995 ibm-915 iso-8859-5 cyrillic csISOLatinCyrillic iso-ir-144 ISO_8859-5:1988 8859_5 cp915 915 windows-28595
ibm-1089_P100-1995	ibm-1089_P100-1995 ibm-1089 iso-8859-6 arabic csISOLatinArabic iso-ir-127 ISO_8859-6:1987 ECMA-114 ASMO-708 8859_6 cp1089 1089 windows-28596 ISO-8859-6-I ISO-8859-6-E
ibm-813_P100-1995	ibm-813_P100-1995 ibm-813 iso-8859-7 greek greek8 ELOT_928 ECMA-118 csISOLatinGreek iso-ir-126 ISO_8859-7:1987 8859_7 cp813 813 windows-28597

Internal converter name	Aliases
ibm-916_P100-1995	ibm-916_P100-1995 ibm-916 iso-8859-8 hebrew csISOLatinHebrew iso-ir-138 ISO_8859-8:1988 ISO-8859-8-I ISO-8859-8-E 8859_8 cp916 916 windows-28598
ibm-920_P100-1995	ibm-920_P100-1995 ibm-920 iso-8859-9 latin5 csISOLatin5 iso-ir-148 ISO_8859-9:1989 l5 8859_9 cp920 920 windows-28599 ECMA-128
ibm-921_P100-1995	ibm-921_P100-1995 ibm-921 iso-8859-13 8859_13 cp921 921
ibm-923_P100-1998	ibm-923_P100-1998 ibm-923 iso-8859-15 Latin-9 l9 8859_15 latin0 csisolatin0 csisolatin9 iso8859_15_fdis cp923 923 windows-28605

Asian language converters

Internal converter name	Aliases
ibm-942_P12A-1999	ibm-942_P12A-1999 ibm-942 ibm-932 cp932 shift_jis78 sjis78 ibm-942_VSUB_VPUA ibm-932_VSUB_VPUA
ibm-943_P15A-2003	ibm-943_P15A-2003 ibm-943 Shift_JIS MS_Kanji csShiftJIS windows-31j csWindows31J x-sjis x-ms-cp932 cp932 windows-932 cp943c IBM-943C ms932 pck sjis ibm-943_VSUB_VPUA
ibm-943_P130-1999	ibm-943_P130-1999 ibm-943 Shift_JIS cp943 943 ibm-943_VASCII_VSUB_VPUA
ibm-33722_P12A-1999	ibm-33722_P12A-1999 ibm-33722 ibm-5050 EUC-JP Extended_UNIX_Code_Packed_Format_for_Japanese csEUCPkdFmtJapanese X-EUC-JP eucjis windows-51932 ibm-33722_VPUA IBM-eucJP

Internal converter name	Aliases
ibm-33722_P120-1999	ibm-33722_P120-1999 ibm-33722 ibm-5050 cp33722 33722 ibm-33722_VASCII_VPUA
ibm-954_P101-2000	ibm-954_P101-2000 ibm-954 EUC-JP
ibm-1373_P100-2002	ibm-1373_P100-2002 ibm-1373 windows-950
windows-950-2000	windows-950-2000 Big5 csBig5 windows-950 x-big5
ibm-950_P110-1999	ibm-950_P110-1999 ibm-950 cp950 950
macos-2566-10.2	macos-2566-10.2 Big5-HKSCS big5hk HKSCS-BIG5
ibm-1375_P100-2003	ibm-1375_P100-2003 ibm-1375 Big5-HKSCS
ibm-1386_P100-2002	ibm-1386_P100-2002 ibm-1386 cp1386 windows-936 ibm-1386_VSUB_VPUA
windows-936-2000	windows-936-2000 GBK CP936 MS936 windows-936

Internal converter name	Aliases
ibm-1383_P110-1999	ibm-1383_P110-1999 ibm-1383 GB2312 csGB2312 EUC-CN ibm-eucCN hp15CN cp1383 1383 ibm-1383_VPUA
ibm-5478_P100-1995	ibm-5478_P100-1995 ibm-5478 GB_2312-80 chinese iso-ir-58 csISO58GB231280 gb2312-1980 GB2312.1980-0
ibm-964_P110-1999	ibm-964_P110-1999 ibm-964 EUC-TW ibm-eucTW cns11643 cp964 964 ibm-964_VPUA
ibm-949_P110-1999	ibm-949_P110-1999 ibm-949 cp949 949 ibm-949_VASCII_VSUB_VPUA
ibm-949_P11A-1999	ibm-949_P11A-1999 ibm-949 cp949c ibm-949_VSUB_VPUA
ibm-970_P110-1995	ibm-970_P110-1995 ibm-970 EUC-KR KS_C_5601-1987 windows-51949 csEUCKR ibm-eucKR KSC_5601 5601 ibm-970_VPUA

Internal converter name	Aliases
ibm-971_P100-1995	ibm-971_P100-1995 ibm-971 ibm-971_VPUA
ibm-1363_P11B-1998	ibm-1363_P11B-1998 ibm-1363 KS_C_5601-1987 KS_C_5601-1989 KSC_5601 csKSC56011987 korean iso-ir-149 5601 cp1363 ksc windows-949 ibm-1363_VSUB_VPUA
ibm-1363_P110-1997	ibm-1363_P110-1997 ibm-1363 ibm-1363_VASCII_VSUB_VPUA
windows-949-2000	windows-949-2000 windows-949 KS_C_5601-1987 KS_C_5601-1989 KSC_5601 csKSC56011987 korean iso-ir-149 ms949
ibm-1162_P100-1999	ibm-1162_P100-1999 ibm-1162
ibm-874_P100-1995	ibm-874_P100-1995 ibm-874 ibm-9066 cp874 TIS-620 tis620.2533 eucTH cp9066
windows-874-2000	windows-874-2000 TIS-620 windows-874 MS874

Windows US and European converters

Internal converter name	Aliases
ibm-437_P100-1995	ibm-437_P100-1995 ibm-437 IBM437 cp437 437 csPC8CodePage437 windows-437
ibm-850_P100-1995	ibm-850_P100-1995 ibm-850 IBM850 cp850 850 csPC850Multilingual windows-850
ibm-851_P100-1995	ibm-851_P100-1995 ibm-851 IBM851 cp851 851 csPC851
ibm-852_P100-1995	ibm-852_P100-1995 ibm-852 IBM852 cp852 852 csPCp852 windows-852
ibm-855_P100-1995	ibm-855_P100-1995 ibm-855 IBM855 cp855 855 csIBM855 csPCp855
ibm-856_P100-1995	ibm-856_P100-1995 ibm-856 cp856 856
ibm-857_P100-1995	ibm-857_P100-1995 ibm-857 IBM857 cp857 857 csIBM857 windows-857

Internal converter name	Aliases
ibm-858_P100-1997	ibm-858_P100-1997 ibm-858 IBM00858 CCSID00858 CP00858 PC-Multilingual-850+euro cp858
ibm-860_P100-1995	ibm-860_P100-1995 ibm-860 IBM860 cp860 860 csIBM860
ibm-861_P100-1995	ibm-861_P100-1995 ibm-861 IBM861 cp861 861 cp-is csIBM861 windows-861
ibm-862_P100-1995	ibm-862_P100-1995 ibm-862 IBM862 cp862 862 csPC862LatinHebrew DOS-862 windows-862
ibm-863_P100-1995	ibm-863_P100-1995 ibm-863 IBM863 cp863 863 csIBM863
ibm-864_X110-1999	ibm-864_X110-1999 ibm-864 IBM864 cp864 csIBM864

Internal converter name	Aliases
ibm-865_P100-1995	ibm-865_P100-1995 ibm-865 IBM865 cp865 865 csIBM865
ibm-866_P100-1995	ibm-866_P100-1995 ibm-866 IBM866 cp866 866 csIBM866 windows-866
ibm-867_P100-1998	ibm-867_P100-1998 ibm-867 cp867
ibm-868_P100-1995	ibm-868_P100-1995 ibm-868 IBM868 cp868 868 csIBM868 cp-ar
ibm-869_P100-1995	ibm-869_P100-1995 ibm-869 IBM869 cp869 869 cp-gr csIBM869 windows-869
ibm-878_P100-1996	ibm-878_P100-1996 ibm-878 KOI8-R koi8 csKOI8R cp878
ibm-901_P100-1999	ibm-901_P100-1999 ibm-901_P100-1999 ibm-901
ibm-902_P100-1999	ibm-902_P100-1999 ibm-902

Internal converter name	Aliases
ibm-922_P100-1999	ibm-922_P100-1999 ibm-922 cp922 922
ibm-4909_P100-1999	ibm-4909_P100-1999 ibm-4909
ibm-5346_P100-1998	ibm-5346_P100-1998 ibm-5346 windows-1250 cp1250
ibm-5347_P100-1998	ibm-5347_P100-1998 ibm-5347 windows-1251 cp1251
ibm-5348_P100-1997	ibm-5348_P100-1997 ibm-5348 windows-1252 cp1252
ibm-5349_P100-1998	ibm-5349_P100-1998 ibm-5349 windows-1253 cp1253
ibm-5350_P100-1998	ibm-5350_P100-1998 ibm-5350 windows-1254 cp1254
ibm-9447_P100-2002	ibm-9447_P100-2002 ibm-9447 windows-1255 cp1255
windows-1256-2000	windows-1256-2000 windows-1256 cp1256
ibm-9449_P100-2002	ibm-9449_P100-2002 ibm-9449 windows-1257 cp1257
ibm-5354_P100-1998	ibm-5354_P100-1998 ibm-5354 windows-1258 cp1258

Internal converter name	Aliases
ibm-1250_P100-1995	ibm-1250_P100-1995 ibm-1250 windows-1250
ibm-1251_P100-1995	ibm-1251_P100-1995 ibm-1251 windows-1251
ibm-1252_P100-2000	ibm-1252_P100-2000 ibm-1252 windows-1252
ibm-1253_P100-1995	ibm-1253_P100-1995 ibm-1253 windows-1253
ibm-1254_P100-1995	ibm-1254_P100-1995 ibm-1254 windows-1254
ibm-1255_P100-1995	ibm-1255_P100-1995 ibm-1255
ibm-5351_P100-1998	ibm-5351_P100-1998 ibm-5351 windows-1255
ibm-1256_P110-1997	ibm-1256_P110-1997 ibm-1256
ibm-5352_P100-1998	ibm-5352_P100-1998 ibm-5352 windows-1256
ibm-1257_P100-1995	ibm-1257_P100-1995 ibm-1257
ibm-5353_P100-1998	ibm-5353_P100-1998 ibm-5353 windows-1257
ibm-1258_P100-1997	ibm-1258_P100-1997 ibm-1258 windows-1258

MAC-related converters

Internal converter name	Aliases
macos-0_2-10.2	macos-0_2-10.2 macintosh mac csMacintosh windows-10000

Internal converter name	Aliases
macos-6-10.2	macos-6-10.2 x-mac-greek windows-10006 macgr
macos-7_3-10.2	macos-7_3-10.2 x-mac-cyrillic windows-10007 maccy
macos-29-10.2	macos-29-10.2 x-mac-centraleurroman windows-10029 x-mac-ce macce
macos-35-10.2	macos-35-10.2 x-mac-turkish windows-10081 mactr
ibm-1051_P100-1995	ibm-1051_P100-1995 ibm-1051 hp-roman8 roman8 r8 csHPRoman8
ibm-1276_P100-1995	ibm-1276_P100-1995 ibm-1276 Adobe-Standard-Encoding csAdobeStandardEncoding
ibm-1277_P100-1995	ibm-1277_P100-1995 ibm-1277 Adobe-Latin1-Encoding

Hebrew, Cyrillic, and ECMA language converters

Internal converter name	Aliases
ibm-1006_P100-1995	ibm-1006_P100-1995 ibm-1006 cp1006 1006
ibm-1098_P100-1995	ibm-1098_P100-1995 ibm-1098 cp1098 1098

Internal converter name	Aliases
ibm-1124_P100-1996	ibm-1124_P100-1996 ibm-1124 cp1124 1124
ibm-1125_P100-1997	ibm-1125_P100-1997 ibm-1125 cp1125
ibm-1129_P100-1997	ibm-1129_P100-1997 ibm-1129
ibm-1131_P100-1997	ibm-1131_P100-1997 ibm-1131 cp1131
ibm-1133_P100-1997	ibm-1133_P100-1997 ibm-1133
ibm-1381_P110-1999	ibm-1381_P110-1999 ibm-1381 cp1381 1381
ISO_2022,locale=ja,version=0	ISO_2022,locale=ja,version=0 ISO-2022-JP csISO2022JP
ISO_2022,locale=ja,version=1	ISO_2022,locale=ja,version=1 ISO-2022-JP-1 JIS JIS_Encoding
ISO_2022,locale=ja,version=2	ISO_2022,locale=ja,version=2 ISO-2022-JP-2 csISO2022JP2
ISO_2022,locale=ja,version=3	ISO_2022,locale=ja,version=3 JIS7 csJISEncoding
ISO_2022,locale=ja,version=4	ISO_2022,locale=ja,version=4 JIS8
ISO_2022,locale=ko,version=0	ISO_2022,locale=ko,version=0 ISO-2022-KR csISO2022KR
ISO_2022,locale=ko,version=1	ISO_2022,locale=ko,version=1 ibm-25546
ISO_2022,locale=zh,version=0	ISO_2022,locale=zh,version=0 ISO-2022-CN
ISO_2022,locale=zh,version=1	ISO_2022,locale=zh,version=1 ISO-2022-CN-EXT

Internal converter name	Aliases
HZ	HZ HZ-GB-2312
ibm-897_P100-1995	ibm-897_P100-1995 ibm-897 JIS_X0201 X0201 csHalfWidthKatakana

Indian language converters

Internal converter name	Aliases
ISCII,version=0 ISCII,version=0	x-iscii-de windows-57002 iscii-dev
ISCII,version=1 ISCII,version=1	x-iscii-be windows-57003 iscii-bng windows-57006 x-iscii-as
ISCII,version=2 ISCII,version=2	x-iscii-pa windows-57011 iscii-gur
ISCII,version=3 ISCII,version=3	x-iscii-gu windows-57010 iscii-guj
ISCII,version=4 ISCII,version=4	x-iscii-or windows-57007 iscii-ori
ISCII,version=5 ISCII,version=5	x-iscii-ta windows-57004 iscii-tml
ISCII,version=6 ISCII,version=6	x-iscii-te windows-57005 iscii-tlg
ISCII,version=7 ISCII,version=7	x-iscii-ka windows-57008 iscii-knd
ISCII,version=8 ISCII,version=8	x-iscii-ma windows-57009 iscii-mlm

EBCDIC converters

Internal converter name	Aliases
LMBCS-1	LMBCS-1 lmbcs
LMBCS-2	LMBCS-2
LMBCS-3	LMBCS-3
LMBCS-4	LMBCS-4
LMBCS-5	LMBCS-5
LMBCS-6	LMBCS-6
LMBCS-8	LMBCS-8
LMBCS-11	LMBCS-11
LMBCS-16	LMBCS-16
LMBCS-17	LMBCS-17
LMBCS-18	LMBCS-18
LMBCS-19	LMBCS-19
ibm-37_P100-1995	ibm-37_P100-1995 ibm-37 IBM037 ibm-037 ebcdic-cp-us ebcdic-cp-ca ebcdic-cp-wt ebcdic-cp-nl csIBM037 cp037 037 cpibm37 cp37
ibm-273_P100-1995	ibm-273_P100-1995 ibm-273 IBM273 CP273 csIBM273 ebcdic-de cpibm273 273

Internal converter name	Aliases
ibm-277_P100-1995	ibm-277_P100-1995 ibm-277 IBM277 cp277 EBCDIC-CP-DK EBCDIC-CP-NO csIBM277 ebcdic-dk cpibm277 277
ibm-278_P100-1995	ibm-278_P100-1995 ibm-278 IBM278 cp278 ebcdic-cp-fi ebcdic-cp-se csIBM278 ebcdic-sv cpibm278 278
ibm-280_P100-1995	ibm-280_P100-1995 ibm-280 IBM280 CP280 ebcdic-cp-it csIBM280 cpibm280 280
ibm-284_P100-1995	ibm-284_P100-1995 ibm-284 IBM284 CP284 ebcdic-cp-es csIBM284 cpibm284 284
ibm-285_P100-1995	ibm-285_P100-1995 ibm-285 IBM285 CP285 ebcdic-cp-gb csIBM285 ebcdic-gb cpibm285 285

Internal converter name	Aliases
ibm-290_P100-1995	ibm-290_P100-1995 ibm-290 IBM290 cp290 EBCDIC-JP-kana csIBM290
ibm-297_P100-1995	ibm-297_P100-1995 ibm-297 IBM297 cp297 ebcdic-cp-fr csIBM297 cpibm297 297
ibm-420_X120-1999	ibm-420_X120-1999 IBM420 cp420 ebcdic-cp-ar1 csIBM420 420
ibm-424_P100-1995	ibm-424_P100-1995 ibm-424 IBM424 cp424 ebcdic-cp-he csIBM424 424
ibm-500_P100-1995	ibm-500_P100-1995 ibm-500 IBM500 CP500 ebcdic-cp-be csIBM500 ebcdic-cp-ch cpibm500 500
ibm-803_P100-1999	ibm-803_P100-1999 ibm-803 cp803

Internal converter name	Aliases
ibm-838_P100-1995	ibm-838_P100-1995 ibm-838 IBM-Thai csIBMThai cp838 838 ibm-9030
ibm-870_P100-1995	ibm-870_P100-1995 ibm-870 IBM870 CP870 ebcdic-cp-roece ebcdic-cp-yu csIBM870
ibm-871_P100-1995	ibm-871_P100-1995 ibm-871 IBM871 ebcdic-cp-is csIBM871 CP871 ebcdic-is cpibm871 871
ibm-875_P100-1995	ibm-875_P100-1995 ibm-875 IBM875 cp875 875
ibm-918_P100-1995	ibm-918_P100-1995 ibm-918 IBM918 CP918 ebcdic-cp-ar2 csIBM918
ibm-930_P120-1999	ibm-930_P120-1999 ibm-930 ibm-5026 cp930 cpibm930 930

Internal converter name	Aliases
ibm-933_P110-1995	ibm-933_P110-1995 ibm-933 cp933 cpibm933 933
ibm-935_P110-1999	ibm-935_P110-1999 ibm-935 cp935 cpibm935 935
ibm-937_P110-1999	ibm-937_P110-1999 ibm-937 cp937 cpibm937 937
ibm-939_P120-1999	ibm-939_P120-1999 ibm-939 ibm-931 ibm-5035 cp939 939
ibm-1025_P100-1995	ibm-1025_P100-1995 ibm-1025 cp1025 1025
ibm-1026_P100-1995	ibm-1026_P100-1995 ibm-1026 IBM1026 CP1026 csIBM1026 1026
ibm-1047_P100-1995	ibm-1047_P100-1995 ibm-1047 IBM1047 cpibm1047
ibm-1097_P100-1995	ibm-1097_P100-1995 ibm-1097 cp1097 1097
ibm-1112_P100-1995	ibm-1112_P100-1995 ibm-1112 cp1112 1112

Internal converter name	Aliases
ibm-1122_P100-1999	ibm-1122_P100-1999 ibm-1122 cp1122 1122
ibm-1123_P100-1995	ibm-1123_P100-1995 ibm-1123 cp1123 1123 cpibm1123
ibm-1130_P100-1997	ibm-1130_P100-1997 ibm-1130
ibm-1132_P100-1998	ibm-1132_P100-1998 ibm-1132
ibm-1140_P100-1997	ibm-1140_P100-1997 ibm-1140 IBM01140 CCSID01140 CP01140 cp1140 cpibm1140 ebcdic-us-37+euro
ibm-1141_P100-1997	ibm-1141_P100-1997 ibm-1141 IBM01141 CCSID01141 CP01141 cp1141 cpibm1141 ebcdic-de-273+euro
ibm-1142_P100-1997	ibm-1142_P100-1997 ibm-1142 IBM01142 CCSID01142 CP01142 cp1142 cpibm1142 ebcdic-dk-277+euro ebcdic-no-277+euro

Internal converter name	Aliases
ibm-1143_P100-1997	ibm-1143_P100-1997 ibm-1143 IBM01143 CCSID01143 CP01143 cp1143 cpibm1143 ebcdic-fi-278+euro ebcdic-se-278+euro
ibm-1144_P100-1997	ibm-1144_P100-1997 ibm-1144 IBM01144 CCSID01144 CP01144 cp1144 cpibm1144 ebcdic-it-280+euro
ibm-1145_P100-1997	ibm-1145_P100-1997 ibm-1145 IBM01145 CCSID01145 CP01145 cp1145 cpibm1145 ebcdic-es-284+euro
ibm-1146_P100-1997	ibm-1146_P100-1997 ibm-1146 IBM01146 CCSID01146 CP01146 cp1146 cpibm1146 ebcdic-gb-285+euro
ibm-1147_P100-1997	ibm-1147_P100-1997 ibm-1147 IBM01147 CCSID01147 CP01147 cp1147 cpibm1147 ebcdic-fr-297+euro

Internal converter name	Aliases
ibm-1148_P100-1997	ibm-1148_P100-1997 ibm-1148 IBM01148 CCSID01148 CP01148 cp1148 cpibm1148 ebcdic-international-500+euro
ibm-1149_P100-1997	ibm-1149_P100-1997 ibm-1149 IBM01149 CCSID01149 CP01149 cp1149 cpibm1149 ebcdic-is-871+euro
ibm-1153_P100-1999	ibm-1153_P100-1999 ibm-1153 cpibm1153
ibm-1154_P100-1999	ibm-1154_P100-1999 ibm-1154 cpibm1154
ibm-1155_P100-1999	ibm-1155_P100-1999 ibm-1155 cpibm1155
ibm-1156_P100-1999	ibm-1156_P100-1999 ibm-1156 cpibm1156
ibm-1157_P100-1999	ibm-1157_P100-1999 ibm-1157 cpibm1157
ibm-1158_P100-1999	ibm-1158_P100-1999 ibm-1158 cpibm1158
ibm-1160_P100-1999	ibm-1160_P100-1999 ibm-1160 cpibm1160
ibm-1164_P100-1999	ibm-1164_P100-1999 ibm-1164 cpibm1164
ibm-1364_P110-1997	ibm-1364_P110-1997 ibm-1364 cp1364

Internal converter name	Aliases
ibm-1371_P100-1999	ibm-1371_P100-1999 ibm-1371 cpibm1371
ibm-1388_P103-2001	ibm-1388_P103-2001 ibm-1388 ibm-9580
ibm-1390_P110-2003	ibm-1390_P110-2003 ibm-1390 cpibm1390
ibm-1399_P110-2003	ibm-1399_P110-2003 ibm-1399
ibm-16684_P110-2003	ibm-16684_P110-2003 ibm-16684
ibm-4899_P100-1998	ibm-4899_P100-1998 ibm-4899 cpibm4899
ibm-4971_P100-1999	ibm-4971_P100-1999 ibm-4971 cpibm4971
ibm-12712_P100-1998	ibm-12712_P100-1998 ibm-12712 cpibm12712 ebcdic-he
ibm-16804_X110-1999	ibm-16804_X110-1999 ibm-16804 cpibm16804 ebcdic-ar
ibm-1137_P100-1999	ibm-1137_P100-1999 ibm-1137
ibm-5123_P100-1999	ibm-5123_P100-1999 ibm-5123
ibm-8482_P100-1999	ibm-8482_P100-1999 ibm-8482
ibm-37_P100-1995,swaplfnl	ibm-37_P100-1995,swaplfnl ibm-37-s390 ibm037-s390
ibm-1047_P100-1995,swaplfnl	ibm-1047_P100-1995,swaplfnl ibm-1047-s390
ibm-1140_P100-1997,swaplfnl	ibm-1140_P100-1997,swaplfnl ibm-1140-s390

Internal converter name	Aliases
ibm-1142_P100-1997,swaplfnl	ibm-1142_P100-1997,swaplfnl ibm-1142-s390
ibm-1143_P100-1997,swaplfnl	ibm-1143_P100-1997,swaplfnl ibm-1143-s390
ibm-1144_P100-1997,swaplfnl	ibm-1144_P100-1997,swaplfnl ibm-1144-s390
ibm-1145_P100-1997,swaplfnl	ibm-1145_P100-1997,swaplfnl ibm-1145-s390
ibm-1146_P100-1997,swaplfnl	ibm-1146_P100-1997,swaplfnl ibm-1146-s390
ibm-1147_P100-1997,swaplfnl	ibm-1147_P100-1997,swaplfnl ibm-1147-s390
ibm-1148_P100-1997,swaplfnl	ibm-1148_P100-1997,swaplfnl ibm-1148-s390
ibm-1149_P100-1997,swaplfnl	ibm-1149_P100-1997,swaplfnl ibm-1149-s390
ibm-1153_P100-1999,swaplfnl	ibm-1153_P100-1999,swaplfnl ibm-1153-s390
ibm-12712_P100-1998,swaplfnl	ibm-12712_P100-1998,swaplfnl ibm-12712-s390
ibm-16804_X110-1999,swaplfnl	ibm-16804_X110-1999,swaplfnl ibm-16804-s390
ebcdic-xml-us	ebcdic-xml-us

Chinese code page GB18030

There are some restrictions if you are working with messages in Chinese code page GB18030.

The broker can input and output application messages encoded in code page IBM-5488 (GB18030 support) with the following restrictions:

- To enable support for `GB18030` in the workbench and Configuration Manager:
  – If you run a workbench or Configuration Manager that requires `GB18030` support on a Windows 2000 system, apply the `GB18030` patch supplied by Microsoft. This support is included in Windows XP.
  – Create the Configuration Manager configuration repository with a code set of `UTF-8`.
  – Change the text font preference in the workbench to use GB18030:
    - Select **Preferences** in the **Window** menu.
    - Expand the **Workbench** item on the left side of the Preferences dialog (click the plus sign) and select **Fonts**.
    - In the Fonts display, select Text Font. Click **Change**, and select the correct values in the Fonts selection dialog.

- Click **OK** to confirm the selection and close the dialog.
- Click **Apply** to apply the change, then **OK** to close the Preference dialog.

Data integrity within message flows

Code pages in which data is manipulated must be compatible between brokers and databases.

Subscription data retrieved from client applications (for example, topics from publishers and subscribers, and content filters from subscribers) and the character data entered through the workbench (for example, message flow names) are stored in the configuration repository. This data is translated from its originating code page to the code page of the process in which the broker or Configuration Manager is running, and then by the database manager to the code page in which the database or databases were created.

To preserve data consistency and integrity, ensure that all this subscription data and workbench character data is originated in a compatible code page to the two code pages to which it is translated. If you do not do so, you might get unpredictable results and lose data.

Data stored in the broker database is not affected in this way.

Configurable message flow properties

When you add a message flow to a broker archive (bar) file in preparation for deploying it to a broker, you can set additional properties that influence its runtime operation. These properties are available for review and update when you select the **Configure** tab for the broker archive file.

Additional Instances

Specifies the number of additional threads that the broker can use to service the message flow. These additional threads are created only if there are sufficient input messages. You can have up to 256 threads. The default value is 0. Additional threads can increase the throughput of a message flow but you should consider the potential impact on message order.

If the message flow processes WebSphere MQ messages, you can configure the message flow to control the message order. Set the *Order Mode* property on the MQInput node accordingly. You might also need to set the *Commit by Message Group* and *Logical Order* properties.

The broker opens the input queue as shared (using the MQOO_INPUT_SHARED option), so you must ensure that the input queue has been defined with the SHARE property to enable multiple broker threads to read from the same input queue.

For more information about the node properties, refer to the "MQInput node" on page 154.

Commit Count

Specifies how many input WebSphere MQ messages are processed by a message flow before a syncpoint is taken (by issuing an MQCMIT).

Set this property only if you have set *Additional Instances* to `0`.

The default value of 1 is also the minimum permitted value. Change this property to avoid frequent MQCMIT calls when messages are being processed quickly and the lack of an immediate commit can be tolerated by the receiving application.

Use the *Commit Interval* to ensure that a commit is performed periodically when not enough messages are received to fulfill the *Commit Count*.

This property has no effect if the message flow does not process WebSphere MQ messages.

Commit Interval

For WebSphere MQ messages, specifies a time interval at which a commit is taken when the *Commit Count* property is greater than 1 (that is, where the message flow is batching messages), but the number of messages processed has not reached the value of the *Commit Count* property. It ensures that a commit is performed periodically when not enough messages are received to fulfill the *Commit Count*.

The time interval is specified in seconds , as a decimal number with a maximum of 3 decimal places (millisecond granularity). The value must be in the range 0.000 through 60.000. The default value is 0.

Set this property only if you have set *Additional Instances* to `0`.

This property has no effect if the message flow does not process WebSphere MQ messages.

You can view and update other configurable properties for the message flow. The properties that are displayed depend on the nodes within the message flow; some have no configurable properties to display. The node properties that are configurable are predominantly system-related properties that are likely to change for each broker to which the message flow is deployed. These properties include the names of WebSphere MQ queues and queue managers. For full details of configurable properties for a node, see the appropriate node description.

Message flow porting considerations

If you have configured a message flow that runs on a broker on a distributed system, and you now want to deploy it to a broker that runs on z/OS, be aware of the following:

WebSphere MQ queue manager and queue names

There are restrictions on WebSphere MQ resource names on z/OS:

- The queue manager name cannot be greater than four characters.
- All queue names must be in uppercase. Although using quotation marks preserves the case, certain WebSphere MQ activities on z/OS cannot find the queue names being referenced.

For more information about configuring on z/OS, refer to the *WebSphere MQ for z/OS Concepts and Planning Guide*.

File system references

File system references must reflect a UNIX file path. If you deploy a message flow to z/OS that you have previously run on Windows, you might have to make changes. If you have previously deployed the message flow to a UNIX system (AIX, Linux, Solaris, or HP-UX), you do not have to make any changes.

Message flow accounting and statistics data

This section provides information for message flow accounting and statistics data.

Details of the information that is collected, and the output formats in which it can be recorded, are provided:
- Statistics details
- Data formats
- Example output

You can also find information on how to use accounting and statistics data to improve the performance of a message flow in this developerWorks article on message flow performance.

Message flow accounting and statistics details

This topic identifies the statistics that are collected for message flows.

The details that are available are:

Message flow statistics

One record is created for each message flow in an execution group. Each record contains the following details:
- Message flow name and UUID
- Execution group name and UUID
- Broker name and UUID
- Start and end times for data collection
- Type of data collected (snapshot or archive)
- CPU and elapsed time spent processing messages
- CPU and elapsed time spent waiting for input
- Number of messages processed
- Minimum, maximum, and average message sizes
- Number of threads available and maximum assigned at any time
- Number of messages committed and backed out
- Accounting origin

Thread statistics

One record is created for each thread assigned to the message flow. Each record contains the following details:
- Thread number (this has no significance and is for identification only)
- CPU and elapsed time spent processing messages
- CPU and elapsed time spent waiting for input
- Number of messages processed
- Minimum, maximum, and average message sizes

Node statistics

One record is created for each node in the message flow. Each record contains the following details:
- Node name
- Node type (for example MQInput)
- CPU time spent processing messages
- Elapsed time spent processing messages
- Number of times that the node is invoked
- Number of messages processed
- Minimum, maximum, and average message sizes

Terminal statistics

One record is created for each terminal on a node. Each record contains the following details:
- Terminal name
- Terminal type (input or output)
- Number of times that a message is propagated through this terminal

For further details about specific output formats, see the following topics:
- "User trace entries for message flow accounting and statistics data" on page 218
- "XML publication for message flow accounting and statistics data"
- "z/OS SMF records for message flow accounting and statistics data" on page 221

Message flow accounting and statistics output formats

The message flow accounting and statistics data is written in one of three formats:
- User trace entries
- XML publication
- z/OS SMF records

XML publication for message flow accounting and statistics data

This topic describe the information that is written to the XML publication for message flow accounting and statistics data. The data is created within the folder WMQIStatisticsAccounting, which contains subfolders that provide more detailed information. All folders are present within the publication even if you set current data collection parameters to specify that the relevant data is not collected.

Snapshot data is used for performance analysis, and is published as retained and non-persistent. Archive data is used for accounting where an audit trail might be required, and is published as retained and persistent. All publications are global and can be collected by a subscriber that has registered anywhere in the network. They can also be collected by more than one subscriber.

One XML publication is generated for each message flow that is producing data for the time period you have chosen. For example, if MessageFlowA and MessageFlowB, are both producing archive data over a period of 60 minutes, both MessageFlowA and MessageFlowB will produce an XML publication every 60 minutes.

If you are concerned about the safe delivery of these messages, for example for charging purposes, use a secure delivery mechanism such as WebSphere MQ.

The folders and subfolders in the XML publication have the following identifiers:
- WMQIStatisticsAccounting
- MessageFlow
- Threads
- ThreadStatistics
- Nodes
- NodesStatistics
- TerminalStatistics

The tables provided here describe the contents of each of these folders in the order listed above.

The table below describes the general accounting and statistics information, created in folder WMQIStatisticsAccounting.

Field	Data type	Details
RecordType	Character	Type of output, one of: • Archive • Snapshot
RecordCode	Character	Reason for output, one of: • MajorInterval • Snapshot • Shutdown • ReDeploy • StatsSettingsModified

The table below describes the message flow statistics information, created in folder MessageFlow.

Field	Data type	Details
BrokerLabel	Character (maximum 32)	Broker name
BrokerUUID	Character (maximum 32)	Broker universal unique identifier
ExecutionGroupName	Character (maximum 32)	Execution group name
ExecutionGroupUUID	Character (maximum 32)	Execution group universal unique identifier
MessageFlowName	Character (maximum 32)	Message flow name
StartDate	Character	Interval start date (YYYY-MM-DD)
StartTime	Character	Interval start time (HH:MM:SS:NNNNNN)
EndDate	Character	Interval end date (YYYY-MM-DD)
EndTime	Character	Interval end time (HH:MM:SS:NNNNNN)
TotalElapsedTime	Numeric	Total elapsed time spent processing input messages (microseconds)
MaximumElapsedTime	Numeric	Maximum elapsed time spent processing an input message (microseconds)
MinimumElapsedTime	Numeric	Minimum elapsed time spent processing an input message (microseconds)
TotalCPUTime	Numeric	Total CPU time spent processing input messages (microseconds)
MaximumCPUTime	Numeric	Maximum CPU time spent processing an input message (microseconds)

Field	Data type	Details
MinimumCPUTime	Numeric	Minimum CPU time spent processing an input message (microseconds)
CPUTimeWaitingForInputMessage	Numeric	Total CPU time spent waiting for input messages (microseconds)
ElapsedTimeWaitingForInputMessage	Numeric	Total elapsed time spent waiting for input messages (microseconds)
TotalInputMessages	Numeric	Total number of messages processed
TotalSizeOfInputMessages	Numeric	Total size of input messages (bytes)
MaximumSizeOfInputMessages	Numeric	Maximum input message size (bytes)
MinimumSizeOfInputMessages	Numeric	Minimum message input size (bytes)
NumberOfThreadsInPool	Numeric	Number of threads in pool
TimesMaximumNumberofThreadsReached	Numeric	Number of times the maximum number of threads is reached
TotalNumberOfMQErrors[1]	Numeric	Number of MQGET errors (MQInput node)
TotalNumberOfMessagesWithErrors[2]	Numeric	Number of messages that contain errors
TotalNumberOfErrorsProcessingMessages	Numeric	Number of errors processing a message
TotalNumberOfTimeOutsWaitingForRepliesToAggregateMessages	Numeric	Number of timeouts processing a message (AggregateReply node only)
TotalNumberOfCommits	Numeric	Number of transaction commits
TotalNumberOfBackouts	Numeric	Number of transaction backouts
AccountingOrigin	Character (maximum 32)	Accounting origin

Notes:
1. For example, a conversion error occurs when the message is got from the queue.
2. These include exceptions that are thrown downstream of the input node, and errors detected by the input node after it has successfully retrieved the message from the queue but before it has propagated it to the out terminal (for example, a format error).

The table below describes the thread statistics information, created in folder Threads.

Field	Data type	Details
Number	Numeric	Number of thread statistics subfolders within Threads folder

The table below describes the thread statistics information for each individual thread, created in folder ThreadStatistics, a subfolder of Threads.

Field	Data type	Details
Number	Numeric	Relative thread number in pool
TotalNumberOfInputMessages	Numeric	Total number of messages processed by thread
TotalElapsedTime	Numeric	Total elapsed time spent processing input messages (microseconds)
TotalCUPTime	Numeric	Total CPU time spent processing input messages (microseconds)
CPUTimeWaitingForInputMessage	Numeric	Total CPU time spent waiting for input messages (microseconds)
ElapsedTimeWaitingForInputMessage	Numeric	Total elapsed time spent waiting for input messages (microseconds)
TotalSizeOfInputMessages	Numeric	Total size of input messages (bytes)
MaximumSizeOfInputMessages	Numeric	Maximum size of input messages (bytes)
MinimumSizeOfInputMessages	Numeric	Minimum size of input messages (bytes)

The table below describes the node statistics information, created in folder Nodes.

Field	Data type	Details
Number	Numeric	Number of node statistics subfolders within Nodes folder

The table below describes the node statistics information for each individual node, created in folder NodesStatistics, a subfolder of Nodes.

Field	Data type	Details
Label	Character	Name of node (Label)
Type	Character	Type of node
TotalElapsedTime	Numeric	Total elapsed time spent processing input messages (microseconds)
MaximumElapsedTime	Numeric	Maximum elapsed time spent processing input messages (microseconds)
MinimumElapsedTime	Numeric	Minimum elapsed time spent processing input messages (microseconds)
TotalCPUTime	Numeric	Total CPU time spent processing input messages (microseconds)
MaximumCPUTime	Numeric	Maximum CPU time spent processing input messages (microseconds)
MinimumCPUTime	Numeric	Minimum CPU time spent processing input messages (microseconds)
CountOfInvocations	Numeric	Total number of messages processed by this node
NumberOfInputTerminals	Numeric	Number of input terminals
NumberOfOutputTerminals	Numeric	Number of output terminals

The table below describes the terminal statistics information, created in folder TerminalStatistics.

Field	Data type	Details
Label	Character	Name of terminal
Type	Character	Type of terminal, one of: • Input • Output
CountOfInvocations	Numeric	Total number of invocations

User trace entries for message flow accounting and statistics data

This topic describes the information that is written to the user trace log for message flow accounting and statistics data.

The data records are identified by the following message numbers:
- BIP2380I
- BIP2381I
- BIP2382I
- BIP2383I

The inserts for each message are described in the following tables, in the order shown above.

The following table describes the inserts in message BIP2380I. One message is written for the message flow.

Field	Data type	Details
ProcessID	Numeric	Process ID
Key	Numeric	Key used to associate related accounting and statistics BIP messages
Type	Character	Type of output, one of: • Archive • Snapshot
Reason	Character	Reason for output, one of: • MajorInterval • Snapshot • Shutdown • ReDeploy • StatsSettingsModified
BrokerLabel	Character (maximum 32)	Broker name
BrokerUUID	Character (maximum 32)	Broker universal unique identifier
ExecutionGroupName	Character (maximum 32)	Execution group name
ExecutionGroupUUID	Character (maximum 32)	Execution group universal unique identifier
MessageFlowName	Character (maximum 32)	Message flow name

Field	Data type	Details
StartDate	Character	Interval start date (YYYY-MM-DD)
StartTime	Character	Interval start time (HH:MM:SS:NNNNNN)
EndDate	Character	Interval end date (YYYY-MM-DD)
EndTime	Character	Interval end time (HH:MM:SS:NNNNNN)
TotalElapsedTime	Numeric	Total elapsed time spent processing input messages (microseconds)
MaximumElapsedTime	Numeric	Maximum elapsed time spent processing an input message (microseconds)
MinimumElapsedTime	Numeric	Minimum elapsed time spent processing an input message (microseconds)
TotalCPUTime	Numeric	Total CPU time spent processing input messages (microseconds)
MaximumCPUTime	Numeric	Maximum CPU time spent processing an input message (microseconds)
MinimumCPUTime	Numeric	Minimum CPU time spent processing an input message (microseconds)
CPUTimeWaitingForInputMessage	Numeric	Total CPU time spent waiting for input messages (microseconds)
ElapsedTimeWaitingForInputMessage	Numeric	Total elapsed time spent waiting for input messages (microseconds)
TotalInputMessages	Numeric	Total number of messages processed
TotalSizeOfInputMessages	Numeric	Total size of input messages (bytes)
MaximumSizeOfInputMessages	Numeric	Maximum input message size (bytes)
MinimumSizeOfInputMessages	Numeric	Minimum input message size (bytes)
NumberOfThreadsInPool	Numeric	Number of threads in pool
TimesMaximumNumberofThreadsReached	Numeric	Number of times the maximum number of threads is reached
TotalNumberOfMQErrors[1]	Numeric	Number of MQGET errors (MQInput node)
TotalNumberOfMessagesWithErrors[2]	Numeric	Number of messages that contain errors
TotalNumberOfErrorsProcessingMessages	Numeric	Number of errors processing a message

Field	Data type	Details
TotalNumberOfTimeOutsWaitingForRepliesToAggregateMessages	Numeric	Number of timeouts processing a message (AggregateReply node only)
TotalNumberOfCommits	Numeric	Number of transaction commits
TotalNumberOfBackouts	Numeric	Number of transaction backouts
AccountingOrigin	Character (maximum 32)	Accounting origin

Notes:
1. For example, a conversion error occurs when the message is got from the queue.
2. These include exceptions that are thrown downstream of the input node, and errors detected by the input node after it has successfully retrieved the message from the queue (for example, a format error).

The following table describes the inserts in message BIP2381I. One message is written for each thread.

Field	Data type	Details
ProcessID	Numeric	Process ID
Key	Numeric	Key used to associate related accounting and statistics BIP messages
Number	Numeric	Relative thread number in pool
TotalNumberOfInputMessages	Numeric	Total number of messages processed by thread
TotalElapsedTime	Numeric	Total elapsed time spent processing input messages (microseconds)
TotalCUPTime	Numeric	Total CPU time spent processing input messages (microseconds)
CPUTimeWaitingForInputMessage	Numeric	Total CPU time spent waiting for input messages (microseconds)
ElapsedTimeWaitingForInputMessage	Numeric	Total elapsed time spent waiting for input messages (microseconds)
TotalSizeOfInputMessages	Numeric	Total size of input messages (bytes)
MaximumSizeOfInputMessages	Numeric	Maximum size of input messages (bytes)
MinimumSizeOfInputMessages	Numeric	Minimum size of input messages (bytes)

The following table describes the inserts in message BIP2382I. One message is written for each node.

Field	Data type	Details
ProcessID	Numeric	Process ID
Key	Numeric	Key used to associate related accounting and statistics BIP messages
Label	Character	Name of node (Label)
Type	Character	Type of node
TotalElapsedTime	Numeric	Total elapsed time spent processing input messages (microseconds)

Field	Data type	Details
MaximumElapsedTime	Numeric	Maximum elapsed time spent processing input messages (microseconds)
MinimumElapsedTime	Numeric	Minimum elapsed time spent processing input messages (microseconds)
TotalCPUTime	Numeric	Total CPU time spent processing input messages (microseconds)
MaximumCPUTime	Numeric	Maximum CPU time spent processing input messages (microseconds)
MinimumCPUTime	Numeric	Minimum CPU time spent processing input messages (microseconds)
CountOfInvocations	Numeric	Total number of messages processed by this node
NumberOfInputTerminals	Numeric	Number of input terminals
NumberOfOutputTerminals	Numeric	Number of output terminals

The following table describes the inserts in message BIP2383I. One message is written for each terminal on each node.

Field	Data type	Details
ProcessID	Numeric	Process ID
Key	Numeric	Key used to associate related accounting and statistics BIP messages
Label	Character	Name of terminal
Type	Character	Type of terminal, one of: • Input • Output
CountOfInvocations	Numeric	Total number of invocations

z/OS SMF records for message flow accounting and statistics data

This topic describes the information that is written to z/OS SMF records for message flow accounting and statistics data.

The data records are type 117 records with the following identifiers:
- BipSMFDate
- BipSMFRecordHdr
- BipSMFTriplet
- BipSMFMessageFlow
- BipSMFThread
- BipSMFNode
- BipSMFTerminal

The following tables describe the contents of each of these records in the order listed above.

The following table describes the contents of the BipSMFDate record.

Field	Data type	Details
YYYY	signed short int	2 byte year

Field	Data type	Details
MM	char	1 byte month
DD	char	1 byte day

The following table describes the contents of the BipSMFRecordHdr record.

Field	Data type	Details
SM117LEN	unsigned short int	SMF record length
SM117SEG	unsigned short int	System reserved
SM117FLG	char	System indicator
SM117RTY	char	Record type 117 (x'75')
SM117TME	unsigned int	Time when SMF moved the record (time since midnight in hundredths of a second)
SM117DTE	unsigned int	Date when SMF moved the record in packed decimal form 0cyydddF where: c is 0 (19xx) or 1 (20xx) yy is the current year (0-99) ddd is the current day (1-366) F is the sign
SM117SID	unsigned int	System ID
SM117SSI	unsigned int	Subsystem ID
SM117STY	unsigned short int	Record subtype, one of : • 1 (only message flow or threads data is being collected) • 2 (node data is being collected)[1]
SM117TCT	unsigned int	Count of triplets
SM117SRT	unsigned char	Record type, one of: • Archive • Snapshot
SM117SRC	unsigned char	Record code, one of: • 00 = None • 01 = Major Interval • 02 = Snapshot • 03 = Shutdown • 04 = Redeploy • 05 = Stats Settings Modified
SM117RSQ	unsigned short int	Sequence number of the record when multiple records are written for a collection interval.
SM117NOR	unsigned short int	Total number of related records in a collection interval.
Note: 1. When only nodes data is being collected, a single subtype 2 record is written. If nodes and terminals data is being collected, multiple subtype 2 records are written.		

The following table describes the contents of the BipSMFTriplet record.

Field	Data type	Details
TRPLTOSE	signed int	Offset of record from start of SMF record
TRPLTDLE	signed short int	Length of data type

Field	Data type	Details
TRPLTNDR	signed short int	Number of data types in SMF record

The following table describes the contents of the BipSMFMessageFlow record.

Field	Data type	Details
IMFLID	short int	Control block hex ID (BipSMFMessageFlow_ID)
IMFLLEN	short int	Length of control block
IMFLEYE	char[4]	Eyecatcher (IMFL)
IMFLVER	int	Version number (BipSMFRecordVersion)
IMFLBKNM	char32]	Broker name
IMFLBKID	char[36]	Broker universal unique identifier
IMFLEXNM	char[32]	Execution group name
IMFLEXID	char[36]	Execution group universal unique identifier
IMFLMFNM	char[32]	Message flow name
IMFLSTDT	BipSMFDate	Interval start date
IMFLSTTM	unsigned int	Interval start time (format as for SM117TME)
IMFLENDT	BipSMFDate	Interval end date
IMFLENTM	unsigned int	Interval end time (format as for SM117TME)
IMFLTPTM	long long int	Total elapsed time spent processing input messages (8 bytes binary, microseconds)
IMFLMXTM	long long int	Maximum elapsed time spent processing an input message (8 bytes binary, microseconds)
IMFLMNTM	long long int	Minimum elapsed time spent processing an input message (8 bytes binary, microseconds)
IMFLTPCP	long long int	Total CPU time spent processing input messages (8 bytes binary, microseconds)
IMFLMXCP	long long int	Maximum CPU time spent processing an input message (8 bytes binary, microseconds)
IMFLMNCP	long long int	Minimum CPU time spent processing an input message (8 bytes binary, microseconds)
IMFLWTCP	long long int	Total CPU time spent waiting for input messages (8 bytes binary, microseconds)
IMFLWTIN	long long int	Total elapsed time spent waiting for input messages (8 bytes binary, microseconds)
IMFLTPMG	unsigned int	Total number of messages processed
IMFLTSMG	long long int	Total size of input messages (bytes)
IMFLMXMG	long long int	Maximum input message size (bytes)
IMFLMNMG	long long int	Minimum input message size (bytes)
IMFLTHDP	unsigned int	Number of threads in pool
IMFLTHDM	unsigned int	Number of times the maximum number of threads is reached
IMFLERMQ[1]	unsigned int	Number of MQGET errors (MQInput node)
IMFLERMG[2]	unsigned int	Number of messages that contain errors
IMFLERPR	unsigned int	Number of errors processing a message

Field	Data type	Details
IMFLTMOU	unsigned int	Number of timeouts processing a message (AggregateReply node only)
IMFLCMIT	unsigned int	Number of transaction commits
IMFLBKOU	unsigned int	Number of transaction backouts
IMFLACCT	char[32]	Accounting origin

Notes:
1. For example, a conversion error occurs when the message is got from the queue.
2. These include exceptions that are thrown downstream of the input node, and errors detected by the input node after it has successfully retrieved the message from the queue (for example, a format error).

The following table describes the contents of the BipSMFThread record.

Field	Data type	Details
ITHDID	short int	Control block hex ID (BipSMFThread_ID)
ITHDLEN	short int	Length of control block
ITHDEYE	char[4]	Eyecatcher (ITHD)
ITHDVER	int	Version number (BipSMFRecordVersion)
ITHDNBR	unsigned int	Relative thread number in pool
ITHDTPMG	unsigned int	Total number of messages processed by thread
ITHDTPTM	long long int	Total elapsed time spent processing input messages (8 bytes binary, microseconds)
ITHDTPCP	long long int	Total CPU time spent processing input messages (8 bytes binary, microseconds)
ITHDWTCP	long long int	Total CPU time spent waiting for input messages (8 bytes binary, microseconds)
ITHDWTIN	long long int	Total elapsed time spent waiting for input messages (8 bytes binary, microseconds)
ITHDTSMG	long long int	Total size of input messages (bytes)
ITHDMXMG	long long int	Maximum size of input messages (bytes)
ITHDMNMG	long long int	Minimum size of input messages (bytes)

The following table describes the contents of the BipSMFNode record.

Field	Data type	Details
INODID	short int	Control block hex ID (BipSMFNode_ID)
INODLEN	short int	Length of control block
INODEYE	char[4]	Eyecatcher (INOD)
INODVER	int	Version number (BipSMFRecordVersion)
INODNDNM	char[32]	Name of node (Label)
INODTYPE	char[32]	Type of node
INODTPTM	long long int	Total elapsed time spent processing input messages (8 bytes binary, microseconds)
INODMXTM	long long int	Maximum elapsed time spent processing input messages (8 bytes binary, microseconds)

Field	Data type	Details
INODMNTM	long long int	Minimum elapsed time spent processing input messages (8 bytes binary, microseconds)
INODTPCP	long long int	Total CPU time spent processing input messages (8 bytes binary, microseconds)
INODMXCP	long long int	Maximum CPU time spent processing input messages (8 bytes binary, microseconds)
INODMNCP	long long int	Minimum CPU time spent processing input messages (8 bytes binary, microseconds)
INODTPMG	unsigned int	Total number of messages processed by this node
INODNITL	unsigned int	Number of input terminals
INODNOTL	unsigned int	Number of output terminals

The following table describes the contents of the BipSMFTerminal record.

Field	Data type	Details
ITRMID	short int	Control block hex ID (BipSMFTerminal_ID)
ITRMLEN	short int	Length of control block
ITRMEYE	char[4]	Eyecatcher (ITRM)
ITRMVER	int	Version number (BipSMFRecordVersion)
ITRMTLNM	char[32]	Name of terminal
ITRMTYPE	char[8]	Type of terminal, one of: • Input • Output
ITRMTINV	unsigned int	Total number of invocations

Example message flow accounting and statistics data

The following topics give example output in two formats:
- XML publication
- User trace entries

An example is not provided for z/OS SMF records, because these contain hexadecimal data and are not easily viewed in that form. To view SMF records, use any available utility program that processes SMF records. For example, you can download WebSphere MQ SupportPac IS11, which generates formatted SMF records that are very similar to formatted user trace entries.

Example of an XML publication for message flow accounting and statistics

This topic shows an XML publication that contains message flow accounting and statistics data.

The following example shows the output generated for a snapshot report. The content of this publication message shows that the message flow is called XMLflow, and that it is running in an execution group named `default` on broker MQ02BRK. The message flow contains the following nodes:
- An MQInput node called INQueue3
- An MQOutput node called OUTQueue

- An MQOutput node called FAILQueue

The MQInput node out terminal is connected to the OUTQueue node. The MQInput node failure terminal is connected to the FAILQueue node.

During the interval for which statistics have been collected, this message flow processed no messages.

A publication generated for this data always includes the appropriate folders, even if there is no current data.

The following command has been issued to achieve these results:

```
mqsichangeflowstats MQ02BRK -s -c active -e default -f XMLFlow -n advanced -t basic -b basic -o xml
```

Blank lines have been added between folders to improve readability.

The following example is the subscription message. The `<psc>` and `<mcd>` elements are part of the RFH header, and the `<WMQIStatisticsAccounting>` element is the message body.

```
<psc>
  <Command>Publish</Command>
  <PubOpt>RetainPub</PubOpt>
  <Topic>$SYS/Broker/MQ02BRK/StatisticsAccounting/SnapShot/default/XMLflow
  </Topic>
</psc>


<mcd>
  <Msd>xml</Msd>
</mcd>


<WMQIStatisticsAccounting RecordType="SnapShot" RecordCode="Snapshot">
```

The following example is the publication that the broker generates:

```
<MessageFlow BrokerLabel="MQ02BRK"
 BrokerUUID="7d951e31-f200-0000-0080-efe1b9d849dc"
 ExecutionGroupName="default"
 ExecutionGroupUUID="77cf1e31-f200-0000-0080-efe1b9d849dc"
 MessageFlowName="XMLflow" StartDate="2003-01-17"
 StartTime="14:44:34.581320" EndDate="2003-01-17" EndTime="14:44:44.582926"
 TotalElapsedTime="0"
 MaximumElapsedTime="0" MinimumElapsedTime="0" TotalCPUTime="0"
 MaximumCPUTime="0"MinimumCPUTime="0" CPUTimeWaitingForInputMessage="685"
 ElapsedTimeWaitingForInputMessage="10001425" TotalInputMessages="0"
 TotalSizeOfInputMessages="0" MaximumSizeOfInputMessages="0"
 MinimumSizeOfInputMessages="0" NumberOfThreadsInPool="1"
 TimesMaximumNumberOfThreadsReached="0" TotalNumberOfMQErrors="0"
 TotalNumberOfMessagesWithErrors="0" TotalNumberOfErrorsProcessingMessages="0"
 TotalNumberOfTimeOutsWaitingForRepliesToAggregateMessages="0"
 TotalNumberOfCommits="0" TotalNumberOfBackouts="0" AccoutingOrigin="Anonymous"/>


<Threads Number="1">
<ThreadStatistics Number="5" TotalNumberOfInputMessages="0"
TotalElapsedTime="0" TotalCPUTime="0" CPUTimeWaitingForInputMessage="685"
ElapsedTimeWaitingForInputMessage="10001425" TotalSizeOfInputMessages="0"
MaximumSizeOfInputMessages="0" MinimumSizeOfInputMessages="0"/>
</Threads>


<Nodes Number="3">
```

```
  <NodeStatistics Label="FAILQueue" Type="MQOutput" TotalElapsedTime="0"
    MaximumElapsedTime="0" MinimumElapsedTime="0" TotalCPUTime="0"
    MaximumCPUTime="0" MinimumCPUTime="0" CountOfInvocations="0"
    NumberOfInputTerminals="1" NumberOfOutputTerminals="2">
   <TerminalStatistics Label="failure" Type="Output" CountOfInvocations="0"/>
   <TerminalStatistics Label="in" Type="Input" CountOfInvocations="0"/>
   <TerminalStatistics Label="out" Type="Output" CountOfInvocations="0"/>
  </NodeStatistics>


  <NodeStatistics Label="INQueue3" Type="MQInput" TotalElapsedTime="0"
    MaximumElapsedTime="0" MinimumElapsedTime="0" TotalCPUTime="0"
    MaximumCPUTime="0" MinimumCPUTime="0"CountOfInvocations="0"
    NumberOfInputTerminals="0" NumberOfOutputTerminals="3">
   <TerminalStatistics Label="catch" Type="Output" CountOfInvocations="0"/>
   <TerminalStatistics Label="failure" Type="Output" CountOfInvocations="0"/>
   <TerminalStatistics Label="out" Type="Output" CountOfInvocations="0"/>
  </NodeStatistics>


  <NodeStatistics Label="OUTQueue" Type="MQOutput" TotalElapsedTime="0"
    MaximumElapsedTime="0" MinimumElapsedTime="0" TotalCPUTime="0"
    MaximumCPUTime="0" MinimumCPUTime="0" CountOfInvocations="0"
    NumberOfInputTerminals="1" NumberOfOutputTerminals="2">
    <TerminalStatistics Label="failure" Type="Output" CountOfInvocations="0"/>
    <TerminalStatistics Label="in" Type="Input" CountOfInvocations="0"/>
    <TerminalStatistics Label="out" Type="Output" CountOfInvocations="0"/>
   </NodeStatistics>


 </Nodes>


</WMQIStatisticsAccounting>
```

Example of user trace entries for message flow accounting and statistics

This topic shows a user trace that contains message flow accounting and statistics data.

The following example shows the output generated for a snapshot report. The messages written to the trace show that the message flow is called myExampleFlow, and that it is running in an execution group named `default` on broker MQ01BRK. The message flow contains the following nodes:
- An MQInput node called inNode
- An MQOutput node called outNode

The nodes are connected together (out terminal to in terminal for each connection).

During the interval for which statistics have been collected, this message flow processed 150 input messages.

The records show that there are two threads assigned to this message flow. One thread is assigned when the message flow is deployed (this is the default number); an additional thread (thread 0) listens on the input queue. The listening thread starts additional threads to process input messages dependent on the number of instances that you have configured for the message flow, and the rate of arrival of the input messages on the input queue.

The following command has been issued to achieve these results:

```
mqsichangeflowstats MQ01BRK -s -c active -e default -f myExampleFlow -n advanced -t basic -b basic
```

The trace entries have been retrieved with the mqsireadlog command and formatted using the mqsiformatlog command. The output from mqsiformatlog is shown below. Line breaks have been added to aid readability.

```
BIP2380I: WMQI message flow statistics. ProcessID='328467', Key='6', Type='SnapShot', Reason='Snapshot',
BrokerLabel='MQ01BRK', BrokerUUID='18792e66-e100-0000-0080-f197e5ed81bd',
ExecutionGroupName='default', ExecutionGroupUUID='15d4314a-3607-11d4-8000-09140f7b0000',
MessageFlowName='myExampleFlow',
StartDate='2003-05-20', StartTime='13:44:31.885862',
EndDate='2003-05-20', EndTime='13:44:51.310080',
TotalElapsedTime='9414843', MaximumElapsedTime='1143442', MinimumElapsedTime='35154',
TotalCPUTime='760147', MaximumCPUTime='70729', MinimumCPUTime='3124',
CPUTimeWaitingForInputMessage='45501', ElapsedTimeWaitingForInputMessage='11106438',
TotalInputMessages='150', TotalSizeOfInputMessages='437250',
MaximumSizeOfInputMessages='2915', MinimumSizeOfInputMessages='2915',
NumberOfThreadsInPool='1', TimesMaximumNumberOfThreadsReached='150',
TotalNumberOfMQErrors='0', TotalNumberOfMessagesWithErrors='0',
TotalNumberOfErrorsProcessingMessages='0', TotalNumberOfTimeOuts='0',
TotalNumberOfCommits='150', TotalNumberOfBackouts='0', AccountingOrigin="Anonymous".
Statistical information for message flow 'myExampleFlow' in broker 'MQ01BRK'.
This is an information message produced by WMQI statistics.

BIP2381I: WMQI thread statistics. ProcessID='328467', Key='6', Number='0',
TotalNumberOfInputMessages='0',
TotalElapsedTime='0', TotalCPUTime='0', CPUTimeWaitingForInputMessage='110',
ElapsedTimeWaitingForInputMessage='5000529', TotalSizeOfInputMessages='0',
MaximumSizeOfInputMessages='0', MinimumSizeOfInputMessages='0'.
Statistical information for thread '0'.
This is an information message produced by WMQI statistics.

BIP2381I: WMQI thread statistics. ProcessID='328467', Key='6', Number='18',
TotalNumberOfInputMessages='150',
TotalElapsedTime='9414843', TotalCPUTime='760147', CPUTimeWaitingForInputMessage='45391',
ElapsedTimeWaitingForInputMessage='6105909', TotalSizeOfInputMessages='437250',
MaximumSizeOfInputMessages='2915', MinimumSizeOfInputMessages='2915'.
Statistical information for thread '18'.
This is an information message produced by WMQI statistics.
BIP2382I: WMQI node statistics. ProcessID='328467', Key='6',
Label='inNode', Type='MQInputNode',
TotalElapsedTime='1813446', MaximumElapsedTime='1040209', MinimumElapsedTime='1767',
TotalCPUTime='70565', MaximumCPUTime='686', MinimumCPUTime='451',
CountOfInvocations='150', NumberOfInputTerminals='0', NumberOfOutputTerminals='3'.
Statistical information for node 'inNode'.
This is an information message produced by WMQI statistics.

BIP2383I: WMQI terminal statistics. ProcessID='328467', Key='6',
Label='catch', Type='Output', CountOfInvocations='0',
Statistical information for terminal 'catch'.
This is an information message produced by WMQI statistics.

BIP2383I: WMQI terminal statistics. ProcessID='328467', Key='6',
Label='failure', Type='Output', CountOfInvocations='0',
Statistical information for terminal 'failure'.
This is an information message produced by WMQI statistics.

BIP2383I: WMQI terminal statistics. ProcessID='328467', Key='6',
Label='out', Type='Output', CountOfInvocations='150',
Statistical information for terminal 'out'.
This is an information message produced by WMQI statistics.

BIP2382I: WMQI node statistics. ProcessID='328467', Key='6',
Label='outNode', Type='MQOutputNode',
TotalElapsedTime='1172582', MaximumElapsedTime='177516', MinimumElapsedTime='3339',
TotalCPUTime='85522', MaximumCPUTime='762', MinimumCPUTime='536',
CountOfInvocations='150', NumberOfInputTerminals='1', NumberOfOutputTerminals='2'.
Statistical information for node 'outNode'.
This is an information message produced by WMQI statistics.

BIP2383I: WMQI terminal statistics. ProcessID='328467', Key='6',
Label='failure', Type='Output', CountOfInvocations='0',
Statistical information for terminal 'failure'.
This is an information message produced by WMQI statistics.

BIP2383I: WMQI terminal statistics. ProcessID='328467', Key='6',
Label='in', Type='Input', CountOfInvocations='150',
Statistical information for terminal 'in'.
This is an information message produced by WMQI statistics.

BIP2383I: WMQI terminal statistics. ProcessID='328467', Key='6',
Label='out', Type='Output', CountOfInvocations='0',
Statistical information for terminal 'out'.
This is an information message produced by WMQI statistics.
```

Part 5. Appendixes

Appendix. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this information in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this information. The furnishing of this information does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032,
Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:

INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

Trademarks

AIX	CICS	Cloudscape
DB2	DB2 Connect	DB2 Universal Database
developerWorks	Domino	
Everyplace	FFST	First Failure Support Technology
IBM	IBMLink	IMS
IMS/ESA	iSeries	Language Environment
Lotus	MQSeries	MVS
NetView	OS/400	OS/390
POWER	pSeries	RACF
Rational	Redbooks	RETAIN
RS/6000	SupportPac	S/390
Tivoli	VisualAge	WebSphere
xSeries	z/OS	zSeries

Index

Printed in USA