WebSphere Message Broker

Deploying and Debugging

Version 6 Release 1

WebSphere Message Broker

Deploying and Debugging

Version 6 Release 1

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

This edition applies to version 6, release 1, modification 0, fix pack 5 of IBM WebSphere Message Broker and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

About this topic collection

This PDF file has been created from the WebSphere Message Broker Version 6.1 (fix pack 5 update, September 2009) information center topics. Always refer to the WebSphere Message Broker online information center to access the most current information. The information center is periodically updated on the document update site and this PDF and others that you can download from that Web site might not contain the most current information.

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

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

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

Part 1. Deploying

Deploying

Deploy resources that you create in the workbench, such as message flows, to execution groups on brokers in your broker domain.

The overview section provides information about the different ways in which you can deploy resources, and about the different types of deployment:

The following topics describe the tasks necessary to deploy a message flow application:

Learn how to perform other types of deployment:

Further topics describe other deployment tasks:

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. Message flows and associated resources are packaged in broker archive (BAR) files for deployment.

When you create application resources such as message flows in the workbench, you must distribute them to the brokers on which you want them to run. Associated with the resources that you create is the configuration associated with those brokers in your broker domain. Data for message flows and associated resources is packaged in a broker archive (BAR) file before being sent to the Configuration Manager, from where it is unpackaged and distributed appropriately.

You can initiate a deployment in the following ways:

- From the workbench
- Using the mqsideploy command
- Using functions defined by the Configuration Manager Proxy API

Depending on your work patterns, you might use all these methods at different times. These options are described in "Deployment methods."

You can also perform different types of deployment, depending on whether you are working with new resources, or updating existing ones. Most types of deployment can typically be configured in one of two ways:

- Complete deployment; in which all resources are deployed (or redeployed) to the whole domain
- Delta or incremental deployment; made either only to update information or to deploy to selected brokers within the domain, depending on the type of deployment

See "Types of deployment" on page 6 for further information about full and delta deployment.

When you have read these overview topics, find detailed instructions for the tasks that you want to complete in subsequent topics in this section.

Read the IBM® Redbooks® publication WebSphere Message Broker Basics for further information about deployment.

Deployment methods

Choose the appropriate method of deployment to suit the way in which you are working. You can use the workbench, the mqsideploy command, or functions described by the Configuration Manager Proxy (CMP).

Using the Message Broker Toolkit

In the Broker Administration perspective of the workbench, the Domain view displays all the objects associated with a specific domain. For example, if you expand a topology, all the brokers in the domain are displayed; if you expand a broker, all the execution groups within that broker are displayed. From the Domain 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 Broker Development view onto an execution group within the Domain Navigator view to deploy the contents of the broker archive.

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

Using the mqsideploy command

You can deploy from the command line using the mqsideploy command. On the command line, you specify the connection details as well as parameters specific to the deployment. See "Types of deployment" on page 6 for more information about what resources you can deploy.

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

WebSphere Message Broker provides two files to help you when writing your own scripts that run the mqsideploy command:

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

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

  Information that you do not explicitly specify as parameters on the mqsideploy command (as shown in the examples in subsequent topics), 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`. On Windows® platforms only, you can use `mqsideploy.bat`.

  You must modify the parameters in this file before you use it.

Using the CMP API

You can control deployment from a Java™ program by using functions described by the CMP API. You can also interrogate the responses from the broker and take appropriate action.

Java applications can also use the CMP API 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. Therefore, you can use the CMP API to create and manipulate an entire domain programmatically.

Synchronous and asynchronous operations

The goals of a deployment are the same, regardless of how you initiate it; however, the method that you choose might affect how the operation works:

- If you use the workbench or the CMP API, the request is asynchronous. Control returns immediately to the application from which you initiated the deployment request. You must request the result of the operation at a later time:
  - If you are using the workbench, switch to the Broker Administration perspective and check the Event log.

    A deployment request always completes, because either the broker has sent a response or the timeout has expired. If you have reason to

believe that the deployment might not be successful, for example if you become aware that a problem with the network or the broker might prevent its completion, you can cancel the deployment request. Cancel requests only in exceptional circumstances; cancelation might cause the state of the execution groups to become unpredictable.

– If you are using the CMP API, you can request responses to the deployment later in your program.

When the request is received by the broker, it communicates with the execution groups that are affected by the contents of the deployment request. The broker waits for a certain amount of time, during which it expects the execution groups to complete the work. If the execution groups do not indicate that they have finished before the time has expired, the broker sends back a negative response with message BIP2066.

- If you use the mqsideploy command, the deployment is synchronous and the command waits for a response. Control is returned to the command line, or to the script that issued the command, when a response is received by the broker, or when the wait time defined by the **-w** parameter has expired, whichever occurs first. If the time expires before a response is received, the command completes with a warning message that informs you what has happened. The warning does not mean that the command has failed, only that a response was not received during the time for which it waited.

Types of deployment

Choose the appropriate type of deployment to achieve your goal; check what types are available, and the circumstances in which to use them.

Follow the links to later topics in this section that describe the types of deployment that you can perform from the workbench, and when to use, or not use, each type.

- To deploy message flows, message sets, and other deployable objects to an execution group, see "Message flow application deployment" on page 7.

  This type of deployment uses a broker archive (BAR) file.
- To deploy configuration details, see "Broker configuration deployment" on page 11.
- In publish/subscribe scenarios, you can deploy topics and topologies:
  – "Publish/subscribe topics hierarchy deployment" on page 13
  – "Publish/subscribe topology deployment" on page 12
- To stop a deployment, see "Cancel deployment" on page 13.

This table lists examples of appropriate ways of deploying in a number of common scenarios:

Scenario	Suggested deployment
Adding a broker to the domain (when not using publish/subscribe)	None required.
Modifying the publish/subscribe topic hierarchy	Delta deployment of the topics hierarchy. (The changed elements only in the topic hierarchy are deployed to all brokers in the domain.)
Connecting publish/subscribe brokers using connections or a collective	Delta topology deployment.

Scenario	Suggested deployment
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.)
Tidying up broker resources after removing it from the topology	If the broker is part of a publish/subscribe network, or if you are using the workbench, 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.
Creating and populating a new BAR file	Message flow application deployment using a complete BAR file deployment.
Adding to, or removing from, a deployed BAR file	Message flow application deployment using an incremental BAR file deployment.
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. Cancel a broker deployment only if you are sure that the broker will never respond to the deployment request.

Message flow application deployment

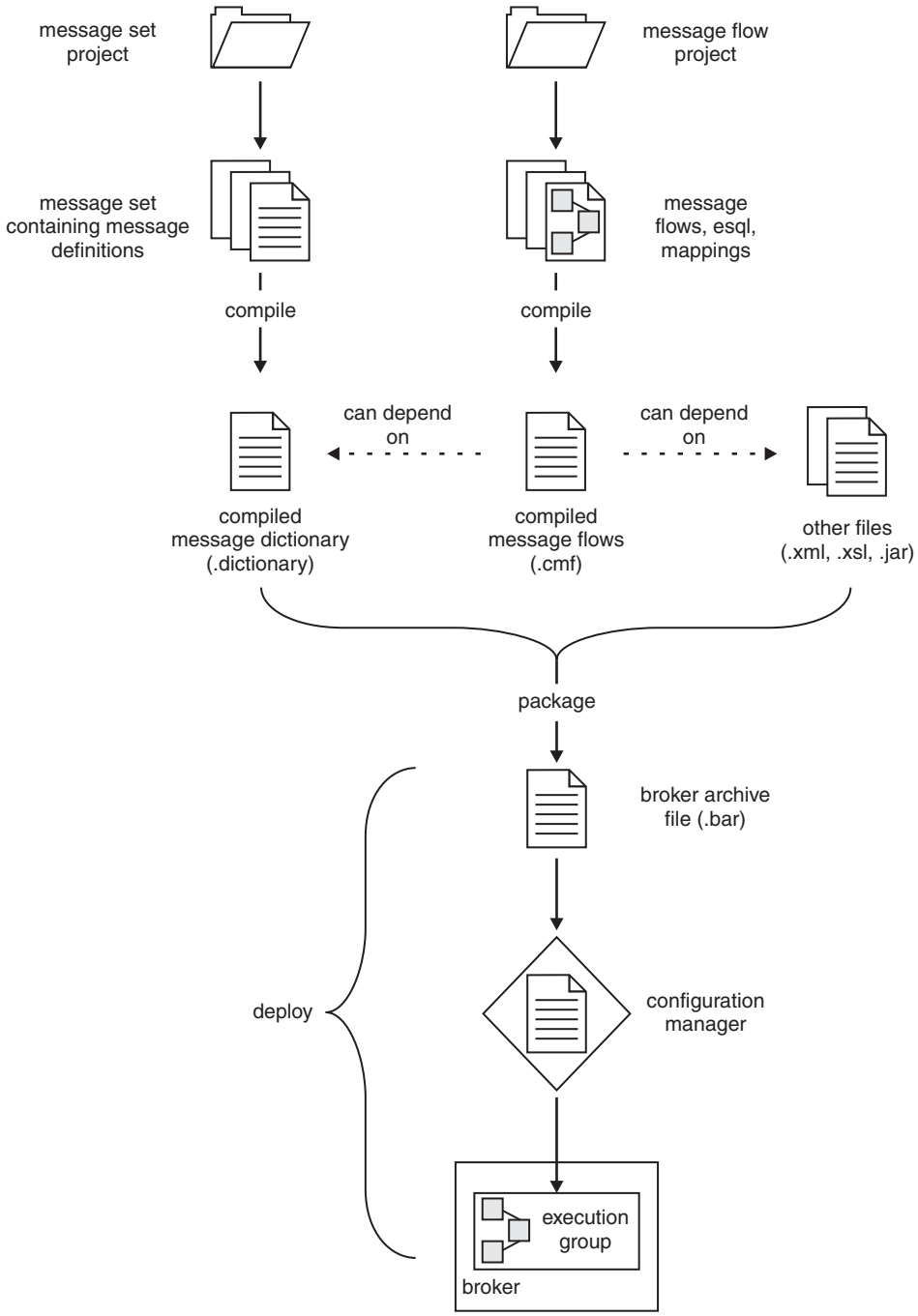
Package all the resources in your message flow into a broker archive (BAR) file for deployment.

You cannot deploy a message flow application directly to an execution group. You must package all the relevant resources into a BAR file, which you then deploy. When you add files to the broker archive, they are automatically compiled as part of the process. JAR files that are required by JavaCompute nodes in message flows are added automatically from your Java project.

The broker archive file is a compressed file, which is sent to the Configuration Manager, where its contents are extracted and distributed to execution groups.

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

This diagram shows the flow of events when you deploy a message flow application:



You can deploy a BAR file in two ways:

- "Incremental BAR file deployment" on page 9. Deployed files are added to the execution group. Files which exist in the execution group are replaced with the new version.
- "Complete BAR file deployment" on page 9. Files that are already deployed to the execution group are removed before the entire contents of the BAR file are deployed. Therefore, nothing is left in the execution group from any previous deployment.

Incremental BAR file deployment

If you run an incremental deployment of a BAR file, the Configuration Manager extracts the contents of the BAR file and sends the contents to the specified execution group. The following conditions are applied when a file is deployed to the BAR file:

- 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, the deployed file is removed from the execution group.

When to use

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

When *not* to use

- To completely clear the contents of the execution group before the BAR file is deployed. Use a complete BAR file deployment if you want this action.

Complete BAR file deployment

If you run a complete deployment of a BAR file, the Configuration Manager extracts the deployable content of the BAR file and sends the contents to the specified execution group, first removing any existing deployed contents of the execution group.

When to use

- To deploy message flows, message sets, or other deployable objects to an execution group.

When *not* to use

- To merge the existing contents of the execution group with the contents of the BAR file. Use an incremental BAR file deployment if you want this action.

Broker archive

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

The BAR file is a compressed file that can contain a number of different files:

- A `.cmf` file for each message flow. This file is a compiled version of the message flow. You can have any number of these files within your BAR file.
- A `.dictionary` file for each message set dictionary. You can have any number of these files within your BAR file.
- One or more XSD compressed files (`.xsdzip`), if XML Schema and WSDL are defined within a message set.
- 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 compressed file and can be modified by using a text editor or shell script.
- One or more XML files (`.xml`), style sheets (`.xsl`), and XSLT files (`.xlst`), if required by nodes in the message flows you have added to this BAR. The XSLTransform node is one that might require these files.
- One or more JAR files, if required by JavaCompute nodes in the message flows you have added to this BAR.

- One or more inbound or outbound adapter files (`.inadapter` or `,outadapter`), if required by WebSphere Adapter nodes (for example, the SiebelInput node) in the message flows you have added to this BAR
- One or more PHP script files (`.php`), if required by PHPCompute nodes in the message flows you have added to this BAR.
- Other files that you might want to associate with this BAR file. For example, you might want to include Java source files, `.msgflow` files, or `.wsdl` files for future reference. BAR files can contain all files types.

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 that are defined in message flows can have 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 or the resources that they work with, such as message mappings, ESQL code, and Java code. 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 using either the Broker Archive editor or the mqsiapplybaroverride command from a command prompt.

Use the supplied editor and command to ensure that the BAR file contents are correct after the changes are applied. You can also edit the XML-format deployment descriptor manually by using an external text editor or shell script; in this case, you must ensure that you have not invalidated the XML content.

Version and keyword information for deployable objects

Use the Broker Archive file editor to view the version and keyword information of deployable objects.

You can display properties of deployed objects, and can modify associated comments:

- "Displaying object version in the Broker Archive editor"
- "Displaying version, deploy time, and keywords of deployed objects" on page 11
- "Populating the Comment and Path columns" on page 11

Displaying object version in the Broker Archive editor

The column in the Broker Archive editor called Version displays the version tag for all objects that have a defined version:

- `.dictionary` files
- `.cmf` files
- Embedded JAR files with a version defined in a `META-INF/keywords.txt` file

You cannot edit the Version column.

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

Displaying version, deploy time, and keywords of deployed objects

The Properties View displays the following properties for all deployed objects:

- Version
- Deploy Time
- All defined keywords

For example, you deploy a message flow with the following literal strings:

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

The Properties View displays these properties:

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

If the keyword information is not available, a message is displayed in the Properties View to indicate the reason. For example, if keyword resolution has not been enabled at deploy time, the Properties View displays the message `Deployed with keyword search disabled`.

If you deploy to a Configuration Manager that is an earlier version than Version 6.0, the message is `Keywords not available on this Configuration Manager`.

Populating the Comment and Path columns

If you add source files, the Path column is populated automatically.

To add a comment, double-click the Comment column and type the text that you require.

Broker configuration deployment

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

When to use

- If you have modified runtime properties in a Configuration Manager Proxy (CMP) application.
- If several CMP programs have modified runtime properties, You can use the mqsideploy command to deploy all of the changes together.

When *not* to use

- If you are adding execution groups. In this case, the first time that you deploy a broker archive (BAR) file, the execution group is automatically initialized.

Publish/subscribe topology deployment

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

You can deploy a topology configuration in two ways:

- 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.

Whichever of these types 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. However, only a complete topology deployment initiates a further subscription.

Complete topology deployment

Deploying a complete topology has the following effects:

- Each broker in the domain is informed of the set of brokers with which it can share publish/subscribe information.
- The Configuration Manager is forced to subscribe again to the broker's status topics, such as start and stop messages.

When to use

- 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 if one broker is able to share publications with a second broker, but not the other way round.

When *not* to use

- 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 deploy only 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

- If you have modified a publish/subscribe network.
- If you are using the workbench to remove a broker from the domain. In this case, the Configuration Manager automatically requests the broker component to stop message flows that are running and to tidy up any resources in use. If this operation fails, you can again request the broker

to tidy up. Deploying a delta topology is the most convenient way to deploy only to those brokers affected by the topology changes.

When *not* to use

- 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.

You can deploy a publish/subscribe topics hierarchy in two ways:

- 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 entire publish/subscribe topics hierarchy to all the brokers in a domain.

When to use

- 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

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

Delta topics deployment

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

When to use

- If you have made changes to the topics hierarchy.

When *not* to use

- If the topics hierarchy has not changed.

Cancel deployment

Canceling a deployment tells the Configuration Manager to assume that a broker will never respond to an outstanding deployment.

You might need to cancel a deployment because 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.

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 an inconsistency subsequently exists between what is running on the broker and the information that is provided by the Configuration Manager.

Because of this risk of inconsistency, cancel a deployment only 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 a deployment, you can manually remove outstanding deployment messages to ensure that they are not processed.

You can cancel a deployment in two ways:

- Cancel deployment to a domain
- Cancel deployment to a broker

Cancel deployment to a domain

Canceling a deployment to a domain has the following effects:

- The Configuration Manager assumes that all brokers in the domain that have outstanding deployments will not respond.
- The locks for all outstanding deployments in the domain are removed.
- Deployment messages that have not yet been processed are *not* removed from any of the brokers in the domain by the Configuration Manager. For brokers that have successfully deployed a configuration, the deployed information remains on the broker.

When to use

Cancel a domain deployment only if both of these conditions are met:

- You receive error message BIP1510 when you attempt a deployment.
- None of the brokers that have outstanding deployments are responding.

When *not* to use

- If the broker is taking a long time to respond to a deployment request, and you are aware of no other problems. 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 deployment to individual brokers instead.

Cancel deployment to a broker

Canceling a deployment to an individual broker has the following effects:

- The Configuration Manager assumes that the specific broker will not respond to outstanding deployments.
- The locks for outstanding deployments to that broker only are removed.
- The Configuration Manager attempts to remove from the broker, deployment messages that have not yet been processed. This succeeds only 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

Cancel a domain deployment only if both of these conditions met:

- You receive error message BIP1510 when you attempt a deployment.
- The broker is not responding.

When *not* to use

- If the broker is taking a long time to respond to a deployment request, and you are aware of no other problems . The broker might have been temporarily stopped, for example.
- The connected Configuration Manager is at Version 6.0 or later. If the version is earlier, canceling deployment to a specific broker has no effect; you must cancel the entire domain deployment instead.

Deploying a message flow application

Deploy message flow applications to execution groups by adding required resources, optionally with their source files, to a broker archive (BAR) file. Send the BAR file to a Configuration Manager, where it is unpacked and the individual files distributed to execution groups on individual brokers.

Before you start:

Before you can deploy a message flow application, you must have created and started a broker.Before you can deploy a message flow application, you must have created and started a Configuration Manager. You must also start a WebSphere MQ listener for the associated queue manager.

Within the workbench, you must create a domain, add a broker to that domain, and create an execution group within the broker. The broker that you add to the domain is a reference, therefore you must also create and start the physical broker on the target system, and start a WebSphere MQ listener on its queue manager. See the links to related tasks at the end of this topic for help with these actions.

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

The tasks in this section describe the process:

1. “Creating a broker archive”
2. “Adding files to a broker archive” on page 16
3. “Refreshing the contents of a broker archive” on page 20
4. “Deploying a broker archive file” on page 21
5. “Checking the results of deployment” on page 30

If your message flows include user-defined nodes, you must also distribute the compiled C or Java code for each node to every broker that uses those message flows. For more details, see Developing user-defined extensions.

Creating a broker archive

Create a separate broker archive (BAR) file for each configuration that you want to deploy to execution groups on brokers in your broker domain.

You can create a BAR file in two ways:

- Using the Message Broker Toolkit
- Using the mqsicreatebar command

Using the Message Broker Toolkit

Follow these steps to create a BAR file by using the workbench:

1. Switch to the Broker Administration perspective.
2. Click **File** → **New** → **Message Broker Archive**.
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 perspective Navigator view, under the Broker Archives folder. The Content editor for the BAR file opens.

Next:

1. Add files to the BAR file
2. Deploy the BAR file

Using the mqsicreatebar command

Follow these steps to create a BAR file by using the mqsicreatebar command:

1. Open a command window that is configured for your environment.
2. Enter the command, typed on a single line. For example:

   ```
   mqsicreatebar -b barName -o filePath -p projectNames -cleanBuild
   ```

   You must specify the **-b** (BAR file name) and **-o** (path for included files) parameters. The **-p** (project names) parameter is optional. For further details, see the mqsicreatebar command.

   If you have made changes to resources in the broker archive by using external tools, add the **-cleanBuild** parameter to refresh all the projects and invoke a clean build. A file with a `.bar` extension is created.

Next:

1. Add files to the BAR file
2. Deploy the BAR file

Adding files to a broker archive

To deploy files to an execution group, you must first include them in a broker archive.

Before you start:

Create a broker archive (BAR) file for each configuration that you want to deploy.

You can add any deployable resources from your workspace to a BAR file. If you select **Include source files**, the source files for all message flows, message sets, or other deployable resources in the broker archive are included.

For further information about the files that you can include in a broker archive, see “Broker archive” on page 9.

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.

Subflows are not displayed in the Build page as separate items, and are added automatically, therefore you have to add only the parent flow to include the subflows.

You can manually add XML, XSL, and JAR files by following these steps. However, JAR files that are required by JavaCompute nodes within message flows are added automatically from your Java project when you add the message flow. XML and XSL files are also added automatically if they are required by the flow.

You do not have to redeploy JAR files unless you have updated them. If one or more JAR files in your BAR file are present on the computer where the broker is running, you can safely remove them from your BAR file before you deploy again. JAR files available to the broker include JAR files that you have deployed as well as JAR files that exist in the shared-classes directory or the `classes` subdirectory of the installation directory. For example, the files `com.ibm.mq.jar`, `ConfigManagerProxy.jar`, `jplugin2.jar`, and `javacompute.jar` are always visible to the broker, and do not have to be deployed separately.

You cannot 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 using the workbench:

1. Switch to either the Broker Administration perspective or the Broker Application Development 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 Manage and Configure page of the Broker Archive editor. (If the BAR file is new, this view is empty.)
3. On the Prepare page of the Broker Archive editor, select deployable workspace resources to add to the broker archive file.
4. Optional: If you want to include your source files, select **Include source files**.
5. Optional: If you are adding a message flow to a broker archive for a second time, and have used the Manage and Configure page to change flow parameters, select **Override configurable property values** to reset configuration settings. If this control is cleared, existing settings are left in place when a flow is replaced.
6. Optional: You can manually remove the resources that you have added to the BAR file using **Remove** in the Manage and Configure page.
7. Click **Build broker archive**. To rebuild selected deployable resources, you can either click **Build broker archive** on the Prepare page, or click **Build** on the Manage and Configure page.

A list of the files that are now in your BAR file is displayed on the Manage and Configure page. You can choose not to display your source files by selecting **Built resources** or **Configurable properties** from the list in the **Filter by** menu.

Next:

If you use configurable properties, see “Editing configurable properties” on page 18.

If you want to have multiple instances of a flow with different values for the configurable properties, see “Adding multiple instances of a message flow to a broker archive” on page 19.

To make further changes to your BAR file, see "Editing a broker archive file manually."

When your BAR file is complete, the next task is: "Deploying a broker archive file" on page 21.

Editing a broker archive file manually

Edit resources that you want to change, in an editor of your choice, by exporting a broker archive (BAR) file from the workbench.

Before you start:

If you have not already created a BAR file, create it now. See "Creating a broker archive" on page 15.

Follow these steps to edit a BAR file manually by using the workbench:

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 for deployment.
   a. From the workbench, click **File** → **Import**. The Import window appears.
   b. Select **Zip file** from the list and click **Next**
   c. Specify the name and location of your BAR file.
   d. Select the project that you want to contain the BAR file.
   e. Click **Finish**.

Next:

"Deploying a broker archive file" on page 21.

Editing configurable properties

You can edit configurable properties in the deployment descriptor file (typically `broker.xml`) of your broker archive.

Before you start:

If you have not already created a BAR file, create it now. See "Creating a broker archive" on page 15.

You can edit configurable properties in two ways:

- Using the Message Broker Toolkit
- Using the mqsiapplybaroverride command

Using the Message Broker Toolkit:

Follow these steps to edit properties using the workbench:

1. Switch to the Broker Administration perspective or Broker Application Development perspective.
2. Open your broker archive, and select the **Manage and Configure** tab. The resources in your broker archive are listed.
3. Optional: You can view the properties that can be configured for your message flows by selecting **Configurable properties** from the **Filter by** list.
4. Expand the message flow in the **Manage and Configure** tab to display the nodes that you can configure. Click on the node you want to configure. The values that you can configure for the node are displayed in the **Properties** view.
5. Select the value that you want to edit in the **Properties** view, and enter the new value. Repeat these steps for all the properties that you want to configure in your message flow.
6. Save your BAR file.

Next:

"Deploying a broker archive file" on page 21.

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).
3. Enter the command, typed on a single line, specifying the location of your broker archive deployment descriptor (typically `broker.xml`) and the file that contains the properties to be changed. See mqsiapplybaroverride for examples on how to use the command. A file with a `.bar` extension is created.

Next:

"Deploying a broker archive file" on page 21.

Adding multiple instances of a message flow to a broker archive

Edit the name of your files in the broker archive (BAR) file so that you can deploy multiple instances of a message flow with different values for the configurable properties.

Before you start:

Add the file to the broker archive. See "Adding files to a broker archive" on page 16.

To deploy multiple instances of the flow with different values for the configurable properties:

1. Rename the message flow file (`.cmf`) in the broker archive editor. Ensure you keep the `.cmf` file extension when you change the file name. You are unable to configure the file if you change the extension.
2. Clear **Remove contents of Broker Archive before building** to prevent your renamed message flow file from being removed from the broker archive when you build the broker archive.
3. Add the message flow to the BAR file again. It is added to the BAR file with the original name.

4. Click the **Manage and Configure** tab. You can now edit the configurable properties for both message flows.

Tip: The names assigned in the BAR file are also used on the command line; for example, if you run mqsilist on your execution group or if you run mqsichangetrace for a message flow.

Next:

Deploy the BAR file. Both message flows are deployed to the execution group and use the values for the configurable properties that you set in the BAR file.

Configuring a message flow at deployment time with user-defined properties

Use user-defined properties (UDPs) to configure message flows at deployment and run time, without modifying program code. You can give a UDP an initial value when you declare it in your program, or when you use the Message Flow editor to create or modify a message flow.

For an overview of user-defined properties, see User-defined properties.

See the DECLARE statement for an example of how to code a UDP statement.

In ESQL, you can define UDPs at the module or schema level.

After a UDP has been defined by the Message Flow editor, you can modify its value before you deploy it.

To configure UDPs:

1. Switch to the Broker Administration perspective or Broker Application Development perspective.
2. Double-click the broker archive (BAR) file in the Navigator view. The contents of the BAR file are shown in the Manage and Configure page of the Broker Archive editor.
3. Click the **Manage and Configure** tab. This tab shows the message flows in your broker archive; expand a flow to show the individual nodes that it contains.
4. Click the message flow that you are interested in. The UDPs that are defined in that message flow are displayed with their values in the **Properties** view.
5. If the value of the UDP is unsuitable for your current environment or task, change it to the value that you want. The value of the UDP is set at the flow level, and is the same for all eligible nodes that are contained in the flow. If a subflow includes a UDP that has the same name as a UDP in the main flow, the value of the UDP in the subflow is not changed.
6. Save your broker archive.

Now you are ready to deploy the message flow. See “Deploying a broker archive file” on page 21.

Refreshing the contents of a broker archive

Refresh the contents of a broker archive by using **Build** on the Manage and Configure page or **Build broker archive** on the Prepare page in the Broker Archive editor. Alternatively, remove resources from your broker archive and, having made the required changes, add them back again.

Before you start:

See “Creating a broker archive” on page 15 and “Adding files to a broker archive” on page 16.

You are likely at some time to make changes to resources that you have already added to your broker archive (BAR) file. Follow these steps to refresh the contents of a broker archive so that they are reflected in the archive before you deploy it.

1. Switch to the Broker Administration perspective or Broker Application Development perspective.

   BAR files that need to be refreshed are shown with an 'out-of-sync' icon in the Navigator view. (When any changes are made to deployable files in the workspace, that have previously been built in the broker archive, the BAR file is considered to be inconsistent. The BAR file is also inconsistent if any changes are made to the project that the files belong to.)
2. Double-click your BAR file in the Navigator view to open it.

   The contents of the BAR file are shown in the Manage and Configure page of the Broker Archive editor.
3. To refresh all the resources in the broker archive, click either **Build** .

   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 archive contents by right-clicking a BAR file in the Navigator view and selecting **Build Broker Archive**. 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 details about the build of an individual deployable resource in the Manage and Configure page, right-click the deployable resource and click **Details**.

   The Properties view opens (if it is not already in the perspective) and the Details tab is displayed. The Details tab shows the following details about the deployable resource:

   - Workspace Resource, with references to the linked workspace resources (`.msgflow`, `.mset`, `.xml`, and `.xslt` files, for example).
   - Last Compile Status, which shows the user log entry for the last compilation. You can copy text, but you cannot modify it.

Next:

“Deploying a broker archive file”

Deploying a broker archive file

After you have created and populated a broker archive (BAR) file, deployment of the file is required to an execution group on a broker, so that the file can take effect in the broker domain.

Before you start:

You must create a BAR file. See “Creating a broker archive” on page 15.

This topic describes how to deploy a broker archive by using each of the following methods:

- "Using the Message Broker Toolkit"
- "Using the mqsideploy command" on page 23
- Using the CMP API

If you change a BAR file, and want to propagate those changes to one or more brokers, you can redeploy the updated BAR file by using one of the methods listed previously:

- "Redeploying a broker archive file" on page 24

If the execution group to which you want to deploy is restricted by an ACL, you must have appropriate access rights to complete this task.

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

Using the Message Broker Toolkit

Follow these steps to deploy a BAR file by using the Message Broker Toolkit:

1. Switch to the Broker Administration perspective.
2. Optional: Typically, an incremental BAR file deployment is performed. 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.

   Do not **Remove Deployed Children** if you want only to refresh one or more of the child processes with the contents of the BAR file. For an explanation of the difference between a complete and an incremental BAR file deployment, see "Message flow application deployment" on page 7.
3. In the Navigator view, select the BAR file that you want to deploy.
4. Deploy the BAR file to an execution group by using one of the following methods:
   - Drag the file onto your target execution group, shown in the Domains view.
   - Right-click the BAR file, and click **Deploy file**. A window opens, and lists all the broker domains, and all execution groups in those broker domains to which the workbench is connected, to which you can deploy the BAR file.

     Select an execution group, and click **OK** to deploy the BAR file.

     If you select a broker topology that is not connected to a domain, an attempt is made to connect the broker topology. If you click **Cancel**, the broker topology remains unconnected to a domain.

   Whichever method you use, you can select (and deploy to) only one execution group at a time.
5. If you have not saved the BAR file since you last edited it, you are asked whether you want to save the file 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, which deploys the file contents (message flows and message sets, for example) to the execution group. In the Domains view, the assigned message flows and message sets are added to the appropriate execution group.

Next: Continue by checking the results of the deployment; see "Checking the results of deployment" on page 30.

Using the mqsideploy command

Follow these steps to deploy a BAR file by using the mqsideploy command:

1. Open a command window that is configured for your environment.
2. Enter the appropriate command for your platform and configuration, using the following examples as a guide.

   On distributed platforms:

   ```
   mqsideploy -i ipAddress -p port -q qmgr -b broker -e egroup -a barfile
   ```

   The command 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 computer.

   You must also specify the **-b** (broker name), **-e** (execution group name), and **-a** (BAR file name) parameters.

   On z/OS®:

   ```
   /f MQ01CMGR,dp b=broker e=egroup a=barfile
   ```

   The command performs an incremental deployment. Add the `m=yes` parameter to perform a complete BAR file deployment.

   In the example, MQ01CMGR is the name of the Configuration Manager component. You must also specify the names of the broker, execution group, and BAR file (the b=, e=, and a= parameters).

The command reports when responses are received from the Configuration Manager and all brokers that are affected by the request. If the command completes successfully, it returns 0.

Next: Continue by checking the results of the deployment; see "Checking the results of deployment" on page 30.

Using the CMP API

Use the deploy method of the ExecutionGroupProxy class.

The following code shows how an application can perform an incremental deployment:

```
import com.ibm.broker.config.proxy.*;
import java.io.IOException;

public class DeployBar {
  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();
    }
  }
}
```

By default, the deploy method performs an incremental deployment. To perform a complete deployment, use a variant of the method that includes a false value for the Boolean isIncremental parameter. For example, e.deploy("deploy.bar",false,0). Set this parameter to true requests an incremental deployment.

Next: Continue by checking the results of the deployment; see "Checking the results of deployment" on page 30.

Redeploying a broker archive file

If you change a BAR file, and want to propagate those changes to one or more brokers, you can redeploy the updated BAR file to one or more execution groups, by using one of the deployment methods described previously. You do not have to stop the message flows that you deployed previously; all resources in the execution group or groups that are in the redeployed BAR file are replaced and new resources are applied.

If your updates to the BAR file include the deletion of resources, a redeployment does not result in their deletion from the broker. For example, assume that your BAR file contains message flows F1, F2, and F3. Update the file by removing F2 and adding message flow F4. If you redeploy the BAR file, all four flows are available in the execution group when the redeployment has completed. F1 and F3 are replaced by the contents of the redeployed BAR file.

If you want to clear previously deployed resources from the execution group before you redeploy, perhaps because you are deleting resources, use one of the methods described earlier:

- To use the workbench, follow the instructions for a complete deployment, making sure that you select **Remove Deployed Children** before deploying.
- To use the mqsideploy command, follow the instructions, making sure that you add the **–m** parameter to perform a complete BAR file deployment.
- To use the CMP API, follow the instructions for a complete deployment.

If your message flows are not transactional, stop the message flows before you redeploy to be sure that all the applications complete cleanly and are in a known and consistent state. You can stop individual message flows, execution groups, or brokers.

If your message flows are transactional, the processing logic that handles commitment or rollback ensures that resource integrity and consistency are maintained.

Next: Continue by checking the results of the redeployment. See "Checking the results of deployment" on page 30.

Deploying a message flow application that uses WebSphere Adapters

Deploy the resources that are generated when you run the Adapter Connection wizard by adding them to a broker archive (BAR) file.

Before you start:

- Read WebSphere Adapters nodes.
- Perform the steps in Preparing the environment for WebSphere Adapters nodes
- Perform the steps in Connecting to an EIS by using the Adapter Connection wizard.

To deploy the message flow successfully, you must deploy the WebSphere Adapters component, either on its own or in the same BAR file as your message flow. If the WebSphere Adapters component is not available, deployment of the message flow fails. The following list includes the file extensions of the resources that you deploy:

- `.msgflow` (the message flow)
- `.inadapter` (the inbound WebSphere Adapters component)
- `.outadapter` (the outbound WebSphere Adapters component)
- `.xsdzip` (the message set)

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

1. For details of the steps that you must perform before you can deploy a message flow application, see "Deploying a message flow application" on page 15.
2. Add the message flow to the BAR file. (For a description of how to add files to a BAR file, see "Adding files to a broker archive" on page 16.) When you add a message flow that contains one or more WebSphere Adapters nodes to a BAR file, a dialog box opens so that you can identify the following resources:
   - One or more WebSphere Adapters components to be used by the WebSphere Adapters nodes
   - One or more message sets that contain an XSD for the business objects that are used by the WebSphere Adapters nodes
3. When you have added the message flow, WebSphere Adapters components, and message set, deploy the BAR file.

Deploying a broker configuration

If you have modified runtime properties, including details of execution groups, and multicast and inter-broker settings, use a broker configuration deployment to inform the broker of your changes.

You can deploy a broker configuration in three ways:

- Using the Message Broker Toolkit
- Using the mqsideploy command
- Using the Configuration Manager Proxy API

Using the Message Broker Toolkit

You do not need to deploy a broker configuration manually from the workbench. If you modify multicast or interbroker settings in the Broker Administration perspective, a broker configuration deployment starts automatically when you apply the changes. This process runs in the background.

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 the examples below, enter the appropriate command, specifying the broker to which you want to deploy:

   On distributed platforms:

   ```
   mqsideploy -i ipAddress -p port -q qmgr -b broker
   ```

   where -i (IP address), -p (port), and -q (queue manager) represent the connection details of the queue manager workstation.

   On z/OS:

   ```
   /f MQ01CMGR,dp b=broker
   ```

   where MQ01CMGR is the name of the Configuration Manager component.

   If you specify 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

Use the deploy method of the BrokerProxy class. By default, the deploy method performs an incremental (delta) deployment. To deploy the complete hierarchy, use a variant of the method that includes the Boolean isDelta parameter set to false. Setting this parameter to true indicates an incremental deployment.

To perform an incremental deployment, 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

When you make a change to your publish/subscribe topology these changes must be deployed to your broker domain.

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.

You can deploy topology information in three ways:

- Using the Message Broker Toolkit
- Using the mqsideploy command
- Using the Configuration Manager Proxy API

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 used only 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 Broker Toolkit

You can configure the workbench so that topology information is automatically deployed after a change. See Changing Broker Administration preferences

Follow these steps to manually 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:

"Checking the results of deployment" on page 30

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 the example below, enter the appropriate command, typed on a single line:

   z/OS On z/OS:

   ```
   /f MQ01CMGR,dp l=yes
   ```

   This command performs a delta deployment. Add the `m=yes` parameter to deploy the entire configuration. MQ01CMGR is the name of the Configuration Manager component.

   On other platforms:

   ```
   mqsideploy –i ipAddress –p port –q qmgr –l
   ```

   This command 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:

"Checking the results of deployment" on page 30

Using the Configuration Manager Proxy

Use the deploy method of the TopologyProxy class. By default, the deploy method performs an incremental (delta) deployment. To deploy the complete hierarchy, use a variant of the method that includes the Boolean isDelta parameter set to false. Setting this parameter to true indicates an incremental deployment.

To perform a complete 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:

"Checking the results of deployment" on page 30

Deploying a publish/subscribe topics hierarchy

Deploy your topics hierarchy using the workbench, the mqsideploy command, or the Configuration Manager Proxy.

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 a delta deployment.

You can deploy a topics hierarchy in three ways:

- Using the Message Broker 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 you have made a change.

Using the Message Broker 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:

"Checking the results of deployment" on page 30

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 the examples below, enter the appropriate command, typed on a single line:

   z/OS On z/OS:

   ```
   /f MQ01CMGR,dp t=yes
   ```

   This command 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 command 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:

"Checking the results of deployment"

Using the Configuration Manager Proxy

Use the deploy method of the TopicRootProxy class. By default, the deploy method performs an incremental (delta) deployment. To deploy the complete hierarchy, use a variant of the method that includes the Boolean isDelta parameter set to false. Setting this parameter to true indicates an incremental deployment.

To perform a complete 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:

"Checking the results of deployment"

Checking the results of deployment

After you have made a deployment, check that the operation has completed successfully.

You can check the results of a deployment in three ways:

- Using the Message Broker Toolkit
- Using the mqsideploy command
- Using the CMP 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 Broker 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 **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

If you use the mqsideploy command to deploy, it returns numeric values from the Configuration Manager and all brokers affected by the deployment, to indicate the outcome. If the deployment completes successfully, the command returns 0. For details of other values that you might see returned, see mqsideploy command.

Using the CMP API

If you are using a CMP API application, you can find out the result of a publish/subscribe topology deployment operation, for example, by using code like this snippet:

```
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 other processes until all affected brokers have responded to the deployment request.

When the method returns, the DeployResult object 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 exception is thrown at the time of deployment. If the Configuration Manager receives the deployment message, log messages for the overall deployment are displayed, followed by completion codes specific to each broker affected by the deployment. The completion code. shown in the following table, is one of the static instances from the CompletionCodeType class.

Completion code	Description
`pending`	The deployment is held in a batch and is not sent until you call ConfigManagerProxy.sendUpdates().
`submitted`	The deployment message was sent to the Configuration Manager but no response was received before the timeout period expired.
`initiated`	The Configuration Manager indicated that deployment has started, but no broker responses were received before the timeout period expired.
`successSoFar`	The Configuration Manager indicated that deployment has started and some, but not all, brokers responded successfully before the timeout period expired. No brokers responded negatively.
`success`	The Configuration Manager indicated that deployment has started and all relevant brokers responded successfully before the timeout period expired.
`failure`	The Configuration Manager indicated that deployment has started and at least one broker responded negatively. You can use getLogEntriesForBroker method of the DeployResult class to get more information about the deployment failure. This method returns an enumeration of available LogEntry objects.
`notRequired`	The deployment request submitted to the Configuration Manager was not sent to the broker, because the broker configuration is already up to date.

Canceling a deployment that is in progress

You can cancel all outstanding deployments in the domain, or just those sent to a particular broker. But cancel a deployment only as a last resort and be sure that the brokers affected, will never be able to process a previous deployment request.

Before you start:

Make sure that you understand the implications of this action. See “Cancel deployment” on page 13.

Make sure that you have the necessary access authority:

- When canceling deployment across the domain, you must have full access authority on the Configuration Manager.
- When canceling deployment to a specific broker, you must have full access authority on that broker.

To ensure that previous deployment messages are not processed when an affected broker is restarted, first remove all existing deployment messages:

1. Stop the broker.
2. Check the two queues used by the broker: SYSTEM.BROKER.ADMIN.QUEUE and SYSTEM.BROKER.EXECUTIONGROUP.QUEUE. Manually remove all deployment messages.
3. Proceed to cancel the deployment.

You can cancel a deployment in three ways:

- Using the Message Broker Toolkit
- Using the mqsideploy command
- Using the Configuration Manager Proxy API

Using the Message Broker Toolkit

Check the details at the start of this topic, and then 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:

"Checking the results of deployment" on page 30. A BIP0892I information message is displayed to show that the request was received by the Configuration Manager.

Using the mqsideploy command

Check the details at the start of this topic, and then follow these steps to cancel a deployment using the mqsideploy command:

1. Open a command window that is configured for your environment.
2. Using the examples below, enter the appropriate command, typed on a single line:

   z/OS On z/OS:

   ```
   /f MQ01CMGR,dp t=yes b=B1
   ```

   This command cancels deployment to the broker called B1. Omit the b argument to cancel all outstanding deployments in the domain. MQ01CMGR is the name of the Configuration Manager component.

   On other platforms:

   ```
   mqsideploy -i ipAddress -p port -q qmgr –c –b B1
   ```

   This command 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.

Next:

"Checking the results of deployment" on page 30. A BIP0892I information message is displayed to show that the request was received by the Configuration Manager.

Using the Configuration Manager Proxy

First, check the details at the start of this topic

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:

"Checking the results of deployment" on page 30. 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. You must change it in the broker archive, and then redeploy the broker archive (BAR) file.

Follow the steps in the following topics:

1. “Removing a deployed object from an execution group”
2. Rename the object
3. “Refreshing the contents of a broker archive” on page 20
4. “Deploying a broker archive file” on page 21

Removing a deployed object from an execution group

Remove deployed objects from an execution group; for example, to rename them.

Before you start:

Stop all message flows in the execution group. See Starting and stopping message flows.

You can remove deployed objects from an execution group in the following ways:

- “Using the Message Broker Toolkit”
- “Using the mqsideploy command”
- “Using the CMP API” on page 36

Using the Message Broker Toolkit

Follow these steps to remove an object from an execution group using the workbench.

1. Switch to the Broker Administration perspective.
2. In the Domains view, right-click the object that you want to remove.
3. Click **Remove**, then **OK** to confirm.

An automatic deployment is performed for the updated broker and a BIP08921 information message is produced, which confirms that the request was received by the Configuration Manager.

Next: If you have removed one or more message flows, you can now remove the resource files that are associated with those message flows; for example, JAR files.

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. Enter the appropriate command for your platform and configuration, using the following examples as a guide.

   On distributed platforms:

   ```
   mqsideploy -i ipAddress -p port -q qmgr –b broker –e egroup
              –d file1.cmf:file2.cmf:file3.dictionary:file4.xml
   ```

   where -i *IP address*, -p *port*, and -q *qmgr* specify the connection details for the Configuration Manager.

   On z/OS:

   ```
   /f MQ01CMGR,dp t=yes b=broker e=egroup
              d=file1.cmf:file2.cmf:file3.dictionary:file4.xml
   ```

where *MQ01CMGR* is the name of the Configuration Manager component.

The -d parameter (d= on z/OS) is a colon-separated list of files that you want to remove from the named execution group. When you run the command, the deployed objects (`file1.cmf, file2.cmf, file3.dictionary, file4.xml`) are removed from the specified execution group.

Optionally, specify -m (m= on z/OS) to clear the contents of the execution group. This option tells the execution group to completely clear all existing data before the new BAR file is deployed.

The command reports when responses are received from the Configuration Manager and all brokers that are affected by the deployment. If the command completes successfully, it returns 0.

Next: If you have removed one or more message flows, you can now remove the resource files that are associated with those message flows; for example, JAR files.

Using the CMP API

To remove deployed objects from an execution group, get a handle to the relevant `ExecutionGroupProxy` object, and then invoke the deleteDeployedObjectsByName method. Use the following code as an 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.cmf",
                           "file3.dictionary",
                           "file4.xml" }, 0);
    }
    catch (ConfigManagerProxyException e) {
      e.printStackTrace();
    }
  }
}
```

Next: If you have removed one or more message flows, you can now remove the resource files that are associated with those message flows; for example, JAR files.

Part 2. Debugging

Testing and debugging message flow applications

Use the flow debugger to track messages through your message flows and check for errors, or use the Test Client to test your message flows in a safe environment before they are used on a production system.

Before you start:

To use the flow debugger effectively, you must have a basic understanding of message flows and their representation in the workbench. See Message flows overview.

The IBM Redbooks publication WebSphere Message Broker Basics also provides information about using the debugger for your message flows.

- "Flow debugger overview"

  Learn about the function provided by the flow debugger, and why you might want to use it.
- "Debugging a message flow" on page 40

  Start the flow debugger and set options to test and debug the message flow.
- "Debugging by using trace" on page 62

  Use trace in various ways to debug messages flows.
- "Testing message flows by using the Test Client" on page 64

  Use the Test Client to monitor the output nodes in the message flow, and provide information about the path that a test message takes through a message flow.

Flow debugger overview

Use the flow debugger in the workbench to track messages through your message flows.

Use the Debug perspective in the workbench to use the flow debugger. For an introduction to the Debug perspective and the views it presents, see: Debug perspective.

You can set breakpoints in a flow and then step through the flow. While you are stepping through, you can examine and change the message variables and the variables used by ESQL code, Java code, and mappings. You can debug a wide variety of error conditions in flows, including the following:

- Nodes that are wired incorrectly (for example, outputs that are connected to the wrong inputs)
- Incorrect conditional branching in transition conditions
- Unintended infinite loops in flow

From a single workbench, you can attach the debugger to one or more execution groups, and debug multiple flows in different execution groups (and therefore multiple messages) at the same time. However, an execution group can be debugged by only one user at a time. Therefore, if you attach your debugger to an execution group, another user cannot attach a debugger to that same execution group until you have ended your debugging session.

When you debug message flows, use a broker that is not being used in a production environment. Debugging might degrade the performance of all message flows in the same execution group and those in other execution groups that share the same broker because they might be affected by potential resource contention.

Debugging code and mappings in message flow nodes

You can use the flow debugger to examine the behavior of code and mappings in message flow nodes.

After you have deployed a message flow, you can set a breakpoint just before one of the nodes listed below so that, when the flow pauses at the breakpoint, you can step through the code or mappings line by line. This allows you to examine the logic, and check the actions taken and their results. You can set additional breakpoints and you can also examine and change variables.

The following nodes can contain ESQL code modules:
- Compute node
- Filter node
- Database node

The following nodes can contain Java code modules:
- User-defined nodes
- JavaCompute node

The following nodes can contain mappings:
- Mapping node
- DataInsert node
- DataUpdate node
- DataDelete node
- Extract node
- Warehouse node

Restrictions

The following restrictions apply when you debug a message flow:

- You must use the same version of the broker and the Message Broker Toolkit; for example, you cannot use the Message Broker Toolkit Version 6.1 to debug a message flow that you have deployed to a broker at an earlier version.
- You should not debug message flows over the Internet; there might be security issues.

Debugging a message flow

Use the tasks described in this section of the documentation to manage and work with the flow debugger.

Before you start

If you are new to debugging, see: “Flow debugger overview” on page 39.

Deploy your message flow to an execution group in a broker and make sure that the broker is running. See: “Deploying a message flow application” on page 15.

To debug a message flow, perform the following tasks. You might want to vary the tasks you perfom and repeat certain tasks, depending on your particular debugging requirements.

1. Start the flow debugger.

   Set required preferences and then start debugging by attaching the flow debugger to an execution group. You can then send test messages along the flow. See: “Starting the flow debugger.”
2. Work with breakpoints.

   Add and manipulate breakpoints in your message flow. See: “Working with breakpoints in the flow debugger” on page 46.
3. Follow the progress of a test message.

   Use breakpoints to pause the progress of a test message so that you can observe its behavior. See: “Stepping through message flow instances in the debugger” on page 50.
4. View message data.

   View (and change) data in messages, ESQL code, Java code or mappings as debugging progresses. See: “Debugging data” on page 54.
5. Manage message flows.

   During a debugging session, there are various administrative tasks you might need to do. When you have finished debugging, detach the debugger from the execution group. See: “Managing flows and flow instances during debugging” on page 58.

Starting the flow debugger

To start the flow debugger, you must attach it to an execution group. You might first want to set certain parameters. When the flow debugger is started, you can introduce test messages to your message flow.

Complete the following tasks to start the debugger:

1. Optional: “Setting flow debugger preferences”
2. “Attaching the flow debugger to an execution group for debugging” on page 42
3. Optional: “Debug: putting a test message on an input queue” on page 44
4. Optional: “Debug: getting a test message from an output queue” on page 45

Setting flow debugger preferences

Set your own preferences for the flow debugging environment in the Broker Application Development perspective of the workbench.

Complete these steps to set preferences for the flow debugging environment in the workbench:

1. Switch to the Broker Application Development perspective.
2. Click **Run** → **Debug** to display the Create, Manage, and Run configurations panel.
3. Click **Message Broker Debug** → **New Configuration**.
4. Assign a new port in the **Java Debug Port** field and click **Select Execution Group** to attach the port to an execution group from the displayed list.
5. Click **Debug** to launch the debugger or **Close** to close the wizard and save your changes.

Next:

"Attaching the flow debugger to an execution group for debugging"

Attaching the flow debugger to an execution group for debugging

Before you can debug your message flow, you must attach the flow debugger to the execution group where your flow is deployed, and then start a debugging session.

Before you start:

- Create a message flow. See: Developing message flows
- Deploy your message flow to a broker execution group. See: "Deploying a message flow application" on page 15
- Start the broker. See: Starting and stopping a broker

From a single workbench, you can attach the flow debugger to multiple execution groups that are running on the same or on different host computers, and debug their flows (and therefore multiple messages) simultaneously.

An execution group can be debugged by only one user at a time. Therefore, if you attach your debugger to an execution group, another user cannot attach a debugger to that same execution group until you have ended your debugging session.

The flow debugger can debug runtime brokers from previous versions. Select the version of the broker that you want to debug by checking the corresponding option on the Engine Selection panel in the Debug wizard, as described previously.

To attach the debugger to an execution group:

1. Switch to the Broker Administration perspective. Note the name of your message flow as it is displayed in the Domains pane.
2. Set a Java debug port number. To configure the broker JVM with a debug port number, use one of the following methods:
   - Right-click the execution group with which you want to work, and click **Preferences**. You can now set the port number.
   - Set the Java debug port by running the mqsichangeproperties command (all on one line) in the Command Console:

     ```
     mqsichangeproperties broker_name -e execution_group_name
     -o ComIbmJVMManager -n jvmDebugPort -v port_number
     ```

     For example:

     ```
     mqsichangeproperties TEST -e default
     -o ComIbmJVMManager -n jvmDebugPort -v 3920
     ```

   When this command has completed, restart the broker. See: Starting and stopping a broker.

   The Java JIT (just-in-time) compiler is disabled if the jvmDebugPort parameter is set to an integer greater than zero. If you are not debugging a message flow, reset the jvmDebugPort parameter to zero to maximize performance.
3. Open the message flow that you want to debug in the Message Flow editor by double-clicking its name in the Broker Administration Navigator pane.
4. Add a breakpoint to a connection that leads out of the input node to ensure that the message flow does not run to completion before you can begin to debug it.

The breakpoint appears as  . For information about adding a breakpoint, see "Working with breakpoints in the flow debugger" on page 46.

5. Switch to the Debug perspective.
6. Click the down-arrow on the **Debug** icon  on the toolbar, and click **Debug** to invoke the Debug (Create, manage, and run configurations) wizard.

   You are now creating a debug launch configuration. If you have created one previously, you can relaunch it by clicking directly on the **Debug** icon  itself. This action generates an error if any of the following conditions are true:

   - You have not already created a debug launch configuration.
   - The broker and execution group to which you previously attached are no longer running.
   - The broker and execution group have been restarted and therefore have a new process ID.
7. In the list of configurations, select **Message Broker Debug** and click **New**. A tab menu displays, beginning with the Connect tab.

   You cannot click **Debug** until you complete the fields on the Connect panel. You can then choose to complete the fields on the other panels, or click **Debug** straight away.The panels in the wizard are:

   - Java debug setting: use this panel to debug a message flow. The Java port is the port number that is specified for the broker JVM. If you do not specify a port, Java debugging is disabled.
   - Deployment Location: click the **Select Execution Group** button to display a list of Execution groups.
   - Source: use this panel to tell the debugger where to look for your source files for flow, mapping, ESQL, or Java, during debugging. The lookup path can be an Eclipse project name, an external folder, or a compressed (`.zip`) file. You can specify multiple locations, but the debugger always looks first in the message flow project that you specified on the Connect panel. If you do not correctly configure the location of the source files, message flow, ESQL, and Java files might not be displayed during debugging. If the source lookup path is not specified for a mapping node, you might encounter unexpected behavior when you use the debugger to step through the message map.
   - Common: this panel is not directly used by the flow debugger; however setting options in this panel does effect the debugger. See the Workbench User Guide for details.
8. Click **Debug**. In the Debug view, the name of the selected host computer and execution group are displayed.
9. When the next message comes into your flow and arrives at breakpoint you added after the input node, the flow pauses, the breakpoint icon is highlighted:  , and you can start debugging.
10. In the Debug view, double-click the message flow that you want to debug. The message flow opens in the Message Flow editor. You can now add more breakpoints, start stepping over the flow, and so on.

Next:

Continue with one of the following tasks:

- Optional: “Debug: putting a test message on an input queue” and “Debug: getting a test message from an output queue” on page 45. These tasks involve putting messages to, and taking messages from, WebSphere MQ queues and are therefore useful only if your message flow includes MQInput and MQOutput nodes.
- “Working with breakpoints in the flow debugger” on page 46.

Debug: putting a test message on an input queue

You can put a message on an input queue to test a message flow that you are debugging.

Before you start

Complete the steps described in: “Attaching the flow debugger to an execution group for debugging” on page 42.

If your message flow includes MQInput and MQOutput nodes, you can test the flow by putting a message on the input queue of your first MQInput node.

You can use the command line interfaces or WebSphere MQ Explorer to put a message to a queue.

You can also use the Test Client as a repeatable alternative. To use the Test Client, complete the steps described in the following sections:

- “Using enqueue in the Test Client”
- “Adding data to your message” on page 45
- “Optional: Using a file of sample data” on page 45

If the message is processed by the message flow and is put on an output queue, you can retrieve it from that queue. See: “Debug: getting a test message from an output queue” on page 45.

Using enqueue in the Test Client:

To configure an enqueue event in the Test Client so that you can use it to send a test message:

1. Switch to the Broker Application Development perspective.
2. Click **File** → **New** → **Other**. The **New** dialog opens.
3. Select **Message Broker Test Client** in the Message Brokers category and click **Next**. The wizard opens and displays its first panel.
4. Select the project in which you want to create the Test Client file.
5. Enter a name for the Test Client file and click **Finish**. The Test Client editor opens.
6. On the toolbar at the top right of the Test Client editor, click the **Put a message onto a queue** icon  .
7. Enter the names of the queue manager and the queue for the input node for this flow. Queue manager names are case-sensitive; check that you enter the name correctly.

   If you are putting a message onto an input queue that is on a remote computer, ensure that the queue manager of the associated broker has a server-connection channel called SYSTEM.BKR.CONFIG.

8. If you are putting a message onto a remote queue, enter values to identify the host and port of the computer that is hosting the queue.
9. Click **File** → **Save** to save the file.

Adding data to your message:

If you want to add just a small amount of test data in your test message, type the data into the **Source** section of the **Message** pane:

1. Open your Test Client file.
2. Type your test data directly into the **Source** section of the Message pane.
3. Put the test message by clicking **Send Message**.

Optional: Using a file of sample data:

If you want your test message to contain a larger quantity of sample data (for example some structured XML), you can import a file containing that data into the Test Client:

1. In the **Events** tab of your Test Client file, click **Import Source**.
2. Select the file you want to use as the content for the test message, and click **Open**. The contents of the selected file is added to the **Source** pane.
3. Click **File** → **Save** when you have finished.
4. Put the test message by clicking the **Send Message** button.

Debug: getting a test message from an output queue

You can get a message from an output queue to test a message flow that you are debugging.

Before you start

Completed the following tasks:

- Developing message flows
- "Deploying" on page 3
- "Attaching the flow debugger to an execution group for debugging" on page 42
- "Debug: putting a test message on an input queue" on page 44

If your message flow includes MQInput and MQOutput nodes, you can test the flow by putting a message on the input queue of your first MQInput node and retrieving it from an MQOutput node.

You can use the command line interfaces or WebSphere MQ Explorer to get a message from an output queue.

You can also use the Test Client as a repeatable alternative. To use the Test Client, complete the following steps:

1. Switch to the Broker Application Development perspective.
2. Click **File** → **New** → **Other**. The **New** dialog opens.
3. Select **Message Broker Test Client** in the Message Brokers category and click **Next**. The wizard opens and displays its first panel.
4. Select the project in which you want to create the Test Client file.
5. Enter a name for the Test Client file and click **Finish**. The Test Client editor opens.

6. On the toolbar at the top right of the Test Client editor, click the **Get a message from a Queue** icon  .
7. Enter the name of the queue manager and output node queue.
8. Click **Get Message** to read a message from the queue.

When a message is available on an output queue, you can see it in the Test Client editor.

Working with breakpoints in the flow debugger

When you have started a debugging session by attaching the debugger to an execution group, you can set breakpoints to control where the message flow will pause.

Use the following tasks to manage breakpoints:

- "Adding breakpoints in the flow debugger"
- "Restricting breakpoints in the flow debugger to specific flow instances" on page 47
- "Enabling and disabling breakpoints in the flow debugger" on page 48
- "Removing breakpoints in the flow debugger" on page 49

Next:

After you have set one or more breakpoints in the message flow, continue your debugging session by stepping through the message flow, pausing at each active breakpoint. See: "Stepping through message flow instances in the debugger" on page 50.

You can also examine message data, code, and mappings at appropriate points. See: "Debugging data" on page 54.

Adding breakpoints in the flow debugger

Add breakpoints to connections in your message flow to control where flow processing will pause.

Before you start:

Attach the flow debugger to the execution group where your flow is deployed. See: "Attaching the flow debugger to an execution group for debugging" on page 42.

You can add breakpoints to the connections of a message flow that is open in the Message Flow editor. Each breakpoint that you add to a flow is also automatically added to all other instances of the flow and you do not need to restart any of the instances.

Every breakpoint is automatically enabled when you add it to a connection and the connection is flagged with the enabled breakpoint symbol  .

Manually set a breakpoint after the collector node or any other multithreaded node. When you use the Debug perspective on the node, you see that the thread has been ended.

To add breakpoints to the connections of a message flow:

1. Switch to the Debug perspective.
2. Add breakpoints to the appropriate connections. Use any of the following methods:

Option	Method
Add breakpoints individually to selected connections.	1. In the Message Flow editor, right-click the connection where you want to set the breakpoint. 2. Click **Add Breakpoint**.
Add breakpoints simultaneously to all connections *entering* a selected node.	1. In the Message Flow editor, right-click the node before which you want to set breakpoints. 2. Click **Add Breakpoints Before Node**.
Add breakpoints simultaneously to all connections *leaving* a selected node.	1. In the Message Flow editor, right-click the node after which you want to set breakpoints. 2. Click **Add Breakpoints After Node**.

Next:

After you have set one or more breakpoints in the message flow, step through the flow, pausing at each active breakpoint. See: "Stepping through message flow instances in the debugger" on page 50.

You can also examine message data, code, and mappings at appropriate points. See: "Debugging data" on page 54.

Restricting breakpoints in the flow debugger to specific flow instances

Breakpoints can be applied to particular flow instances, instead of all instances, which is the default behavior.

Before you start:

Add one or more breakpoints to your message flow. See: "Adding breakpoints in the flow debugger" on page 46.

When you add a breakpoint to a message flow in the Message Flow editor, the breakpoint automatically applies to all instances of the flow. However, you can choose to restrict a breakpoint to one or more instances of a flow. This enables you to work more easily with just those instances that you are currently interested in, rather than with all instances.

To restrict a breakpoint to one or more flow instances:

1. Switch to the Debug perspective.
2. In the Breakpoints view, right-click the breakpoint that you want to restrict, then click **Properties** to open the Flow Breakpoints Properties window.
3. In the **Restrict to Selected Flow Instance(s)** list box, select those instances to which you want to restrict the breakpoint.
   - You must have at least one instance active; if not, the **Restrict to Selected Flow Instance(s)** list box is empty.

- If any instance is currently paused at the breakpoint, all check boxes in the **Restrict to Selected Flow Instance(s)** list box are disabled and you cannot select them.

4. Click **OK**.

Next:

Now you can add additional breakpoints (if needed), step through the flow instance, and work with data:

- "Adding breakpoints in the flow debugger" on page 46
- "Stepping through message flow instances in the debugger" on page 50
- "Debugging data" on page 54

Enabling and disabling breakpoints in the flow debugger

You can disable breakpoints that are currently enabled, and vice versa.

Before you start:

Add one or more breakpoints to your message flow. See: "Adding breakpoints in the flow debugger" on page 46.

Message flow processing pauses only at breakpoints that are enabled. By controlling which breakpoints are enabled and which are disabled, you can, for example, allow processing to continue to the part of a flow that you are interested in without having to continually add and remove breakpoints.

The following symbols identify breakpoints:


Enabled breakpoint


Disabled breakpoint

If you disable all the breakpoints in a message flow, you cannot perform any other debugging tasks until you add a new breakpoint, or enable an existing breakpoint.

To change the state of breakpoints:

1. Switch to the Debug perspective.
2. In the Breakpoints view, select one or more breakpoints that you want to enable or disable.
3. Right-click the selected breakpoints and click **Enable** or **Disable**.
4. Optional: to change the state of a single breakpoint, right-click the breakpoint and click **Properties**. Select or clear the **Enabled** check box as required, then click **OK**.

The state of breakpoints is changed in all instances of the message flow where they are set.

Next:

If you have finished debugging, continue with: "Debug: ending a session" on page 60.

Removing breakpoints in the flow debugger

Remove breakpoints that are no longer required from connections in your message flow.

Before you start:

Add one or more breakpoints to your message flow. See: "Adding breakpoints in the flow debugger" on page 46.

The following symbols identify breakpoints:

- Enabled breakpoint
- Disabled breakpoint

If you remove a breakpoint from a message flow, it is automatically removed from all instances of the message flow where it is set.

If you remove all the breakpoints that you have added to your message flow, you cannot perform any other debugging tasks until you add a new breakpoint.

To remove breakpoints:

1. Switch to the Debug perspective.
2. Remove the breakpoints. Use one of the following methods, depending on how many breakpoints you want to remove:

Option	Method
Remove individual breakpoints.	1. In the Message Flow editor, right-click the breakpoint that you want to remove, then click **Remove Breakpoint**.
Remove several breakpoints simultaneously.	1. Click the **Flow Breakpoints** tab to show the Breakpoints view. 2. Select one or more breakpoints that you want to remove. 3. Click the **Remove Selected Breakpoints** icon on the toolbar, or right-click the selected breakpoints and then click Remove.
Remove all breakpoints simultaneously.	1. Click the **Breakpoints** tab to show the Breakpoints view. 2. Click the **Remove All Breakpoints** icon on the toolbar, or right-click any breakpoint and then click Remove All.

Next:

If you have finished debugging, continue with: "Debug: ending a session" on page 60.

Stepping through message flow instances in the debugger

After you have added one or more breakpoints to a message flow in the degugger, you can step through the flow, pausing as required.

Before you start:

Add one or more breakpoints to your message flow. See: "Adding breakpoints in the flow debugger" on page 46.

The message flow debugger pauses flow processing at the first breakpoint it encounters. You can then continue with one or more of the following tasks, as appropriate:

- "Debug: resuming message flow processing"
- "Debug: running to completion" on page 51
- "Debug: stepping over nodes" on page 51
- "Debug: stepping into subflows" on page 52
- "Debug: stepping out of subflows" on page 52
- "Debug: stepping through source code" on page 53

Next:

As you step through the message flow you can look at the processing data. When you have finished, end your debugging session:

- "Debugging data" on page 54
- "Debug: ending a session" on page 60

Debug: resuming message flow processing

Each time message flow processing pauses at an active breakpoint, you can investigate the state of the flow and then resume processing.

Before you start:

Add one or more breakpoints to your message flow. See: "Adding breakpoints in the flow debugger" on page 46.

When message flow processing has paused at a breakpoint, you can resume processing:

1. Switch to the Debug perspective.
2. In the Debug view:
   - either, click **Resume Flow Execution**  on the toolbar.
   - or, right-click the flow stack frame, then click **Resume**  .

Message flow processing continues until the next breakpoint that is set in the logical processing of the current message. If there is no further enabled breakpoint at which the flow instance can pause, processing runs to completion and the flow instance is removed from the Debug view.

Next:

If you have completed debugging this message flow, you can remove the breakpoints, or end the debugging session:

- "Removing breakpoints in the flow debugger" on page 49
- "Debug: ending a session" on page 60

Debug: running to completion

When message flow processing pauses at an active breakpoint, you can choose to let it continue to the end of the flow, ignoring other breakpoints.

Before you start:

Add one or more breakpoints to your message flow. See: "Adding breakpoints in the flow debugger" on page 46.

When message flow processing has paused at a breakpoint, you can restart processing so that the message flow runs to completion.

If you want the flow to continue processing, but you want to pause at the next enabled breakpoint instead of running to completion, see: "Debug: resuming message flow processing" on page 50.

1. Switch to the Debug perspective.
2. In the Debug view:
   - either, click **Run to completion** on the toolbar.
   - or, right-click the flow stack frame, then click **Run to completion** .

The flow instance ignores all breakpoints and processing continues to the end. The flow instance is automatically removed from the Debug view.

Next:

If you have finished debugging, you can remove the breakpoints, or end the debugging session:

- "Removing breakpoints in the flow debugger" on page 49
- "Debug: ending a session" on page 60

Debug: stepping over nodes

When message flow processing pauses at an active breakpoint, you can choose to step over the node and continue processing until the next active breakpoint, ignoring breakpoints that might have been set in code within the node.

Before you start:

Add one or more breakpoints to your message flow. See: "Adding breakpoints in the flow debugger" on page 46.

To step over the next node and continue message flow processing:

1. Switch to the Debug perspective.
2. In the Debug view:
   - either, click **Step Over Node** on the toolbar.
   - or, right-click the flow stack frame, then click **Step Over Node** .

Message flow processing continues until the next breakpoint that is set in the logical processing of the current message. If there is no further enabled breakpoint

at which the flow instance can pause, processing runs to completion and the flow instance is removed from the Debug view.

Next:

If you have finished debugging, you can remove the breakpoints, or end the debugging session:

- "Removing breakpoints in the flow debugger" on page 49
- "Debug: ending a session" on page 60

Debug: stepping into subflows

When message flow processing pauses at a breakpoint, you can step into the subflow that follows.

Before you start:

Add one or more breakpoints to your message flow. See: "Adding breakpoints in the flow debugger" on page 46.

To step into a subflow:

1. Switch to the Debug perspective.
2. In the Debug view:
   - either, click **Step Into Subflow**  on the toolbar.
   - or, right-click the flow stack frame, then click **Step Into Subflow**  .

The subflow opens in the Message Flow editor and displaces the parent message flow. Message flow processing continues until the next breakpoint that is set in the logical processing of the current message. If there is no further enabled breakpoint at which the flow instance can pause, processing runs to completion and the flow instance is removed from the Debug view.

Next:

If you have finished debugging, you can remove the breakpoints, or end the debugging session:

- "Removing breakpoints in the flow debugger" on page 49
- "Debug: ending a session" on page 60

Debug: stepping out of subflows

When message flow processing has paused at a breakpoint in a subflow, you can step out of the subflow.

Before you start

Complet the following tasks:

- "Adding breakpoints in the flow debugger" on page 46
- "Debug: stepping into subflows"

To step out of a subflow:

1. Switch to the Debug perspective.
2. In the Debug view:

- either, click **Step Out of Subflow** on the toolbar.
- or, right-click the flow stack frame, then click **Step Out of Subflow** .

The debugger continues processing until it reaches the connection from the output terminal of the subflow, where it pauses. The parent flow opens in the Message Flow editor, displacing the subflow.

Next:

If you have finished debugging, you can remove the breakpoints, or end the debugging session:

- "Removing breakpoints in the flow debugger" on page 49
- "Debug: ending a session" on page 60

Debug: stepping through source code

When message flow processing has paused at a breakpoint on entry to a node that contains ESQL code, Java code, or mappings, you can step through the code.

Before you start:

Add one or more breakpoints to your message flow. See: "Adding breakpoints in the flow debugger" on page 46.

The nodes that can contain ESQL code, Java code, or mappings are listed in: "Flow debugger overview" on page 39. Add breakpoints as appropriate:

- ESQL code: add a breakpoint in the ESQL code.
- Java code: add a breakpoint in the Java code.
- Mappings: add a breakpoint to a map using the Map Script panel. Note that mapping routines are implemented in ESQL; you might choose to step through the ESQL code rather than the mappings.

To step through your source code:

1. Switch to the Debug perspective.
2. Step into the source code. In the Debug view:
   - either, click **Step into Source Code** on the toolbar.
   - or, right-click the flow stack frame, then click **Step Into** .
3. When message flow processing has paused at a breakpoint within ESQL code, Java code, or mappings, you can step through the source code, line by line. Repeat this step as often as necessary. In the Debug view:
   - either, click **Step Over** on the toolbar.
   - or, right-click the flow stack frame, then click **Step Over** .

   A single line of source code runs and the flow pauses at the next line of code. What you can do depends on what type of code is contained within the node. See:
   - "Debugging ESQL" on page 55
   - "Debugging Java" on page 56
   - "Debugging mappings" on page 57

If the debugger is paused before the last line of code when you step over, the last line of code runs and message flow processing continues until the next breakpoint in the logical processing of the current message. If there is no further enabled breakpoint at which the flow instance can pause, processing runs to completion and the flow instance is removed from the Debug view.

4. If you have finished looking at the code or mappings before the last breakpoint, you can continue processing the message flow. In the Debug view:

   - either, click **Step Return** on the toolbar.
   - or, right-click the flow stack frame, then click **Step Return**.

   The source code runs to completion from the current breakpoint and message flow processing continues until the next breakpoint that is set in the logical processing of the current message. If there is no further enabled breakpoint at which the flow instance can pause, processing runs to completion and the flow instance is removed from the Debug view.

Next:

If you have completed debugging this message flow, you can remove the breakpoints, or end the debugging session:

- "Removing breakpoints in the flow debugger" on page 49
- "Debug: ending a session" on page 60

Debugging data

You can view (and change) data in messages, ESQL code, Java code or mappings as debugging progresses.

When you have added one or more breakpoints to a deployed message flow, the debugger stops the message flow processing at each breakpoint. Depending on the context of the breakpoint, you can do one of the following tasks:

- "Debugging messages"
- "Debugging ESQL" on page 55
- "Debugging Java" on page 56
- "Debugging mappings" on page 57

When you have finished debugging a message flow, you can remove the breakpoints, or end the debugging session:

- "Removing breakpoints in the flow debugger" on page 49
- "Debug: ending a session" on page 60

Debugging messages

When message flow processing has paused at a breakpoint in your message flow, you can examine and modify the message content.

Before you start

Add one or more breakpoints to your message flow. See: "Adding breakpoints in the flow debugger" on page 46.

To examine and modify message data:

1. Switch to the Debug perspective.

2. View the messages in the Variables view.

   The Breakpoints view and the Variables view share the same pane. Click the tab at the bottom to select the view that you want.
3. To alter a message, right-click it and select an option from the menu. You cannot alter the content of exceptions within a message.

Message flow processing continues until the next breakpoint that is set in the logical processing of the current message. If there is no further enabled breakpoint at which the flow instance can pause, processing runs to completion and the flow instance is removed from the Debug view.

Next:

If you have finished debugging this message flow, you can remove the breakpoints, or end the debugging session:

- "Removing breakpoints in the flow debugger" on page 49
- "Debug: ending a session" on page 60

Debugging ESQL

When message flow processing has paused at a breakpoint that you have set in source code within a node that contains ESQL code, you can examine and modify the ESQL variables in the Flow Debugger.

Before you start

Complete the following tasks:

- "Adding breakpoints in the flow debugger" on page 46
- "Debug: stepping through source code" on page 53

You can browse ESQL variables in the Variables view in the Debug Perspective, and change their associated data values. You can also set breakpoints on lines in the ESQL code. See the following sections for further details:

- "Using breakpoints on ESQL code lines"
- "Working with ESQL variables"

Using breakpoints on ESQL code lines:

1. Switch to the Debug perspective.
2. Open the ESQL editor.
3. Right-click a line where you want to set a breakpoint.

   You cannot set a breakpoint on a comment line or a blank line.
4. Select from the menu to create, delete, or restrict the breakpoint, in a similar way to normal debugger breakpoints, as described in "Working with breakpoints in the flow debugger" on page 46.

Working with ESQL variables:

1. Switch to the Debug perspective.
2. Open the Variables view. Variables are shown in a tree, using the symbol  .
3. To work with a variable, right-click it and select an option from the pop-up menu.

   You cannot update message trees, or REFERENCE variables.

For example, if you have declared the following ESQL variables, you can change their values in the debugger:

```
DECLARE myInt INT 0;
DECLARE myFloat FLOAT 0.0e-1;
DECLARE myDecimal DECIMAL 0.1;
DECLARE myInterval INTERVAL DAY TO MONTH;
```

Message flow processing continues until the next breakpoint that is set in the logical processing of the current message. If there is no further enabled breakpoint at which the flow instance can pause, processing runs to completion and the flow instance is removed from the Debug view.

Next:

If you have finished debugging this message flow, you can remove the breakpoints, or end the debugging session:

- "Removing breakpoints in the flow debugger" on page 49
- "Debug: ending a session" on page 60

Debugging Java

When message flow processing has paused at a breakpoint that you have set in source code within a node that contains Java code, you can examine and modify the Java variables in the Flow Debugger.

1. Switch to the Broker Administration perspective.
2. Click **Start → Programs → IBM WebSphere Message Brokers 6.1 → Command Console** to open the Command Console.
3. Start the broker by running the mqsistart command in the Command Console.
4. Set the Java debug port by running the mqsichangeproperties command (all on one line) in the Command Console:

   ```
   mqsichangeproperties broker_name -e execution_group_name
   -o ComIbmJVMManager -n jvmDebugPort -v port_number
   ```

   For example:

   ```
   mqsichangeproperties TEST -e default
   -o ComIbmJVMManager -n jvmDebugPort -v 3920
   ```
5. Stop and restart the broker by running the mqsistop and mqsistart commands.
6. Open the message flow that you want to debug in the Message Flow editor by double-clicking its name in the Broker Development view.
7. Add a breakpoint where the Java method is called, by following the instructions in "Adding breakpoints in the flow debugger" on page 46.
8. To step directly into the Java code during the debugging process, add a breakpoint in the Java code.
9. Deploy the broker archive (BAR) file that includes the JAR file that contains the Java code, by following the instructions in "Deploying a broker archive file" on page 21.
10. Click **Run → Debug** to open the Debug wizard.
11. Right-click **Message Broker Debug** in the list of elements on the left and click **New**.
12. Set the **Java Debug Port** with the same value that you specified for the **-v** parameter on the mqsichangeproperties command, and click **Apply** to save your changes.
13. Click the **Source** tab, specify the source file location, and click **Apply** to save your changes.

14. Click **Debug** to start the debug process.

Working with Java variables:

When message flow processing has paused at a breakpoint in the source code within a node that contains Java code (a user-defined node or a JavaCompute node), you can browse Java variables in the Variables view on the Debug perspective, and change their associated data values.

1. Switch to the Debug perspective.
2. Click the **Variables** tab to open the Variables view if it is not already open.

   Variables are shown in a tree, using the symbol  .
3. To work with a variable, right-click it and select an option from the menu.

Message flow processing continues until the next breakpoint that is set in the logical processing of the current message. If there is no further enabled breakpoint at which the flow instance can pause, processing runs to completion and the flow instance is removed from the Debug view.

Next:

When you have completed debugging the message flow, you can remove the breakpoints or end the debugging session:

- "Removing breakpoints in the flow debugger" on page 49
- "Debug: ending a session" on page 60

Debugging mappings

When message flow processing has paused at a breakpoint that you have set in source code within a node that contains mappings, you can view the mapping routines and modify user-defined variables in the Flow Debugger.

Before you start

To complete this task, you must have completed the following tasks:

- "Adding breakpoints in the flow debugger" on page 46
- "Debug: stepping through source code" on page 53

Mapping routines are implemented in ESQL. If you step into the code, you can choose either to step through the ESQL code, or to step through the mappings.

1. Switch to the Debug perspective.
2. In the Debug view:

   - either, click **Step into Source Code**  on the toolbar.
   - or, right-click the flow stack frame, then click **Step into** .

   The Message Mapping editor opens with the mapping routine highlighted in both the Mapping editor and the Outline view.
3. To use breakpoints on mapping lines:
   a. In the Message Mapping Editor, select the line for the mapping command that you want to use, right-click the space beside it and select from the menu to add or disable a breakpoint. (Alternatively, double-click the same space to add or remove a breakpoint.)

b. Select from the menu to create, delete, or restrict the breakpoint, in a similar way to normal debugger breakpoints, as described in: "Working with breakpoints in the flow debugger" on page 46.

You cannot set a breakpoint on a comment line or a blank line.

4. Check the mapping routines by stepping through the mappings.

   In the Debug view, the stack frame shows the list of mapping commands and the current command. The Variables view shows your user-defined mapping variables and the current message. You can change the values of user-defined variables.

Message flow processing continues until the next breakpoint that is set in the logical processing of the current message. If there is no further enabled breakpoint at which the flow instance can pause, processing runs to completion and the flow instance is removed from the Debug view.

Next:

If you have finished debugging, you can remove the breakpoints, or end the debugging session:

- "Removing breakpoints in the flow debugger" on page 49
- "Debug: ending a session" on page 60

Managing flows and flow instances during debugging

During a debugging session, there are various administrative tasks that you might need to do, which include detaching the debugger from the execution group when you have finished.

When you have started a session for message flow debugging, you might want to complete one or more of the following associated tasks:

- "Debug: querying a broker to find deployed flows"
- "Debug: stopping a message flow instance" on page 59
- "Debug: redeploying a message flow" on page 59
- "Debug: ending a session" on page 60

Debug: querying a broker to find deployed flows

You can find the message flow that you want to work with in the Flow Debugger by refreshing the list of available flows.

During an active debugging session, you can query an execution group on a broker to find out what flows are currently deployed to it. The displayed list of message flows that are available in that execution group is updated. The updated list might include message flows that were not previously deployed, or that were not accessible because the flow was already being accessed by another developer.

To query an execution group for deployed flows:

1. Switch to the Debug perspective.
2. In the Debug view, select the execution group that you want to query, then:

   - either, click **Refresh Selected Flow Engine to Get More Flow Types**  on the toolbar.
   - or, right-click the execution group, then click **Refresh**  .

The Debug view is refreshed with the names of the flows that are currently deployed to the execution group and are available.

Next:

You can continue your debugging session and debug one of the listed message flows, or end your debugging session:

- "Working with breakpoints in the flow debugger" on page 46
- "Debug: ending a session" on page 60

Debug: stopping a message flow instance

While debugging, a message flow cannot be redeployed until it has been stopped.

While you are debugging, you might need to stop a message flow instance. For example, you might want to correct an error in your flow or source code. To do this, you must stop the flow and then redeploy it. See "Debug: redeploying a message flow."

To stop message flow processing, run it to completion:

1. Switch to the Debug perspective.
2. In the Debug view:
   - either, click **Run to completion**  on the toolbar.
   - or, right-click the flow stack frame, then click **Run to completion**  .

The flow instance ignores all breakpoints and processing continues to the end. The flow instance is automatically removed from the Debug view.

Next:

After stopping a flow instance, you can start to debug another message flow, or end your debugging session:

- "Attaching the flow debugger to an execution group for debugging" on page 42
- "Debug: ending a session" on page 60

Debug: redeploying a message flow

If you want to change your message flow while you are debugging it, you must redeploy it to the execution group and then reattach the flow debugger.

Before you start

Stop the message flow before you redeploy it. See: "Debug: stopping a message flow instance"

During your debugging session, you might find a problem in a message flow that you want to correct or see a behavior that you want to change. You can alter the flow to resolve the situation and redeploy the flow to the broker:

1. Switch to the Debug perspective.
2. Detach the debugger from the execution group by clicking **Detach from the Selected Flow Engines**  on the toolbar.
3. Switch to the Broker Application Development perspective.
4. Edit the flow in the Message Flow editor and save your changes.

5. Switch to the Broker Administration perspective.
6. Double-click the broker archive (BAR) file that contains your flow. Remove the flow, then add your edited version and save your changes.

   See “Adding files to a broker archive” on page 16.
7. Deploy your BAR file.

   Drag your BAR file from the Broker Administration Navigator view to the execution group in the Domains view. Check the Event log to make sure that the deployment was successful.

   See: “Deploying a broker archive file” on page 21.
8. Switch to the Debug perspective.
9. Reattach the debugger to the execution group.

   Click the down-arrow on the **Debug** icon  on the toolbar, and select **Debug** to invoke the **Debug (Create, manage, and run configurations)** wizard, and attach the flow engine again, following the instructions in “Attaching the flow debugger to an execution group for debugging” on page 42.

The modified message flow is now deployed to the broker, and the debugging session is ready for you to debug the new flow logic.

Next: Continue to use these tasks to debug your message flow:

- “Working with breakpoints in the flow debugger” on page 46
- “Stepping through message flow instances in the debugger” on page 50

Debug: ending a session

Finish debugging by detaching the flow debugger from the execution group to which your message flows are deployed.

When you have finished debugging a flow, detach the flow debugger from the execution group. Other developers are then able to attach the debugger to the execution group. Detaching the flow debugger also restores the performance of your workbench environment, which might have been reduced by having the debugger attached.

To detach the flow debugger from an execution group:

1. Switch to the Debug perspective.
2. In the Debug view select the name of the execution group from which you want to detach the flow debugger and then:

   - either click **Detach from the Selected Flow Engines**  on the toolbar.
   - or right-click the execution group, then click  **Detach**).

All existing flow instances are automatically run to completion and the flow debugger is detached from the execution group. Your debugging session is now finished. You can start a new debugging session at any time.

Debugging message flows that contain WebSphere Adapters nodes

You can use various methods to monitor message flows that include WebSphere Adapters nodes.

Before you use any of the methods listed below, ensure that the appropriate JAR files and shared libraries are available to the WebSphere Adapters nodes. For more information, see Preparing the environment for WebSphere Adapters nodes.

Also, check for the latest information about WebSphere Adapters at WebSphere Adapters technotes.

- **User and service trace:** You can use user and service trace to trace a message flow that contains WebSphere Adapters nodes. For more information, see Using trace.
- **Flow debugger:** Use the flow debugger in the normal way to debug a message flow that contains WebSphere Adapters nodes. For more information, see "Debugging a message flow" on page 40.
- **Adapter event table:** The WebSphere Adapters nodes use an event table to communicate the outcome of operations asynchronously to a calling application. For more information, see Creating a custom event project in PeopleTools.

Handling exceptions that are raised by a WebSphere Adapters request node

The WebSphere Adapters request nodes raise exceptions that indicate the following Enterprise Information System (EIS) failures.

Message number	Exception type	Explanation
BIP3511	`RecordNotFound`	The requested record could not be found in the EIS.
BIP3512	`DuplicateRecord`	An attempt was made to create a record that already exists in the EIS.
BIP3513	`MultipleMatchingRecords`	A retrieve request matched more than one record. To retrieve multiple records, perform a `retrieveall` operation.
BIP3515	`MatchesExceededLimit`	A `retrieveall` exception returned more entries than the maximum allowed number.
BIP3516	`MissingData`	The message tree that was sent to the adapter request node does not have all the required fields set.

If an exception occurs that does not fit into the categories in the table, the node raises a general BIP3450 message that describes the problem.

You can use these exceptions to perform special processing when you do not want the exceptions to be treated as errors. For example:

- If a `create` operation fails because the record already exists, you could modify the request to an `update`.
- If a `retrieve` operation fails because the request matches more than one record, you could try a `retrieveall` operation instead.
- If a `retrieve` operation fails because the record could not be found, an empty record could be returned.

To handle these exceptions, you can connect a message routing node, Compute node, or JavaCompute node to the Failure terminal of the WebSphere Adapters request node, and route the exception to other processing nodes based on the exception message number.

Debugging by using trace

You can use trace in several ways to analyze message flow behavior.

Enabling user trace shows the history of processing that is carried out in a particular message flow, but it shows only those parts of the messages that were accessed. You can use Trace nodes to write out your own debugging information at specific points in the message flow, including the full message tree at that point, provided that you have coded the flow to include it.

You can use both of these methods to review behavior only after a message has been processed.

- "Debugging with user trace"
- "Debugging by adding Trace nodes to a message flow" on page 63

Debugging with user trace

Built-in nodes write messages to user trace when they are processing work. You can use these messages offline to review the activity in a message flow and show information such as which nodes were invoked, what code they ran, and through which terminals the messages was sent.

Before you start:

Before you start to trace a broker, or any of its execution groups or messages flows, the broker must be running, and you must have deployed the message flows by using the workbench.

If part of the message is parsed by the nodes, user trace shows the fields that are being navigated.

If an error occurs while a message is being processed, the exception is written to user trace. If the error is not caught in the message flow, it is also be written to the system log. Each entry in user trace is prefixed by ″BIP″. You can look up BIP messages in the information center in Diagnostic messages. For information about the location of user trace log files on different operating systems, see User trace.

When you start user tracing, you cause additional processing for every activity in the component that you are tracing. Large quantities of data are generated by the components. Expect to see some impact on performance while trace is active. You can limit this additional processing by being selective about what you trace, and by restricting the time during which trace is active.

- Trace is inactive by default. Turn it on by following the instructions in Starting user trace.
- If you need to check what tracing options are currently active for your brokers, use the mqsireporttrace command, as described in Checking user trace options.
- To change user trace options, use the mqsichangetrace command, as described in Changing user trace options.
- To retrieve user trace, use the mqsireadlog command, as described in Retrieving user trace.
- To format the information that is generated by the mqsireadlog command, use the mqsiformatlog command, as described in Formatting trace.
- For information about how to interpret the contents of user trace, see Interpreting trace.

- To stop user trace, use the mqsichangetrace command, as described in Stopping user trace.
- To clear old information from trace files, use the mqsichangetrace command, as described in Clearing old information from trace files.
- Alternatively, you can include a Trace node in your message flows when you design them. Use a Trace node when you want to specify an alternative location for the trace contents. For more details, see "Debugging by adding Trace nodes to a message flow."

Debugging by adding Trace nodes to a message flow

By adding a Trace node to a message flow, you can write debugging messages to a file, to user trace, or to the system log, and review those messages after the message flow has processed some data.

You must add a Trace node when the message flow is designed. Viewing the logical message tree in trace output explains how to view the structure of the logical message tree at any point in the message flow, and contains an example of the message content. You can turn the Trace node off when a message flow is promoted to production to improve performance, but you can turn the node on when required. Performance can be affected when Trace nodes are active. The extent of the impact depends on the destination that you choose for the debugging messages; for example, writing to user trace is typically faster than writing to a file or to the system log.

The debug messages can include writing part or all of the logical message tree, but they can also include hard-coded strings to identify a particular point in the message flow (such as `PRINTF` in a C program). If you write the entire message tree in a Trace node, the behavior of the message flow might be changed. Typically, only the parts of the message that are referenced are parsed, rather than the entire message.

- Add a Trace node to your message flow, then set the following properties on the node (as described in detail in Trace node):
  - Set the destination of the trace record that is written by the node to User Trace, Local Error Log, or File.
  - If you choose File, set the file path of the file to which to write records.
  - Use the Pattern property to create an ESQL pattern that specifies the data to be included in the trace record.
  - Specify the message catalog from which the error text for the error number of the exception is extracted.
  - Specify the error number of the message that is written.
  -
- After you have added a Trace node to your message flow, you can turn it on or off, as described in Switching Trace nodes on and off.
- To view the structure of the logical message tree at any point in the message flow, include a Trace node and write some or all of the message (including headers and all four message trees) to the trace output destination. The following topics describes how to view that output: Viewing the logical message tree in trace output
- If you write debugging messages to user trace, the following topic describes how to retrieve user trace: Retrieving user trace

Testing message flows by using the Test Client

You can test message flows in a safe environment before they are used on a production system by using the Test Client.

You can use the Test Client to send test messages to message flows that use WebSphere MQ, JMS, SOAP, or HTTP input nodes. The Test Client monitors the output nodes in the message flow, and can provide information about the path that a test message takes through a message flow. The Test Client can also provide information about errors that are generated by the message flow.

You can perform the following tasks using the Test Client:

- "Testing a message flow" on page 65
- "Configuring the test settings" on page 67
- "Creating and editing a test message" on page 68
- "Using the Test Client in trace and debug mode" on page 71

Test Client overview

Use the Test Client to test message flows in a safe environment before they are used in a production system.

You can use the Test Client to send test messages to message flows that use any of the following input nodes:

- WebSphere MQ
- JMS
- HTTP
- SOAP

Configuring the input message

You can use the Test Client to change the content of test messages that are sent to a message flow, to help you determine if the message flow is working as expected.

WebSphere MQ queues
If your message flow uses WebSphere MQ queues, the Test Client clears the queues before your test messages are sent to the message flow.

XML messages
If the input node in the message flow that you select expects an XML message from an associated message set, the message structure is provided, and it can be edited to produce the appropriate test message. Alternatively, you can create a new test message, or import an existing message from your file system.

WebSphere MQ and JMS messages
If the message format is WebSphere MQ or JMS, you can also configure an appropriate header for the test message.

Monitoring a flow with the Test Client

The Test Client monitors output nodes in the message flow so that you can see which nodes output messages are received on. When an error message is produced as the message passes along the flow, or when a message is received on an output node, a test event is recorded in the Test Client.

You can view the content of the output message, and view error messages. The details of the test configuration and the test events can be saved as a `.mbtest` file. You can use this file to repeat the test or to review the results later.

Deploying message flows when you use the Test Client

If you change your message flow, you can use the same test configuration to test the changes. The default behavior of the Test Client is to deploy the message flow that you want to test automatically to an execution group, whenever a change is made to the message flow. You can therefore change a message flow, and quickly test the result using the Test Client, without the need to manually deploy your message flows.

The first time that you send a test message to an input node, you configure the execution group to deploy the message flow by using the Deployment location wizard. You can configure the deployment options to override the default behavior of the Test Client to deploy the message flow manually, or to deploy the message flow every time that you pass a test message to the message flow.

Stopping the Test Client

The default behavior of the Test Client is to stop the test when the first output message is received. You can configure the Test Client to wait for multiple output messages to be received. In this case, you stop the test manually. Stopping the test disconnects the monitors that are running, but does not stop the message flow.

Synchronous tests
: A synchronous test, such as when the message flow is invoked from an HTTPInput node, is stopped automatically when a reply message is received.

Asynchronous tests
: You can stop an asynchronous test, such as when the message flow is invoked from an MQInput node, manually depending on the monitor setting in the configuration panel.

All test events are stopped when the Test Client is closed, and all test monitors removed.

Using The flow debugger with the Test Client

You can run the Test Client using the trace and debug mode to view more information about the path that the message takes through the message flow. A test event is produced when the message passes from one node to the next node in the message flow. The structure of the message is recorded as it leaves each node in the message flow. The flow debugger is launched in the trace and debug mode so that the test message stops at breakpoints that are configured in the message flow.

Testing a message flow

You can test your message flows using the Test Client.

Before you start:

You must have a broker domain configured and running. If you do not have an existing broker domain, create one using the Default Configuration wizard; see

Using the Default Configuration wizard. If the broker domain is not already configured in the Message Broker Toolkit, use the Deployment Location wizard to connect to the broker domain.

To test a message flow, complete the following tasks:

1. "Opening the Test Client editor"
2. "Configuring the test settings" on page 67
3. "Creating and editing a test message" on page 68
4. "Selecting the deployment location for the message flow" on page 70.

The test message is put to the selected input node. The Test Client monitors the output nodes in the message flow and events are generated as the message passes through the message flow. You might need to stop the test manually, depending on the nodes in the message flow and the settings that you have configured in the Test Client.

Next:

To test the message flow again, right-click **Invoke Message Flow** in the **Message Flow Test Events** pane and click **Invoke** to start a new test; or click **Duplicate** or **Re-run** to re-run the test using the same message.

Opening the Test Client editor

Open the Test Client editor and select a message flow to test.

Before you start:

Ensure that there are no errors in your message flow before you open it with the Test Client. Errors are displayed in the Problems view, see Problems view in Broker Application Development perspective for more details.

To open the Test Client editor and ensure that the message flow that you want to test is selected, use one of the following methods:

- From a message flow file:
  1. Switch to the Broker Application Development perspective.
  2. In the Broker Development pane, right-click the message flow you want to test and click **Test Message Flow**.

  The Test Client editor is opened with settings from the message flow.
- From an input node:
  1. Open the message flow that you want to test.
  2. Right-click an input node in the message flow and click **Test**.

  The Test Client editor is opened with settings from the message flow, and the input node is selected.
- From a message flow:
  1. Open the message flow that you want to test.
  2. Right-click anywhere in the message flow and click **Test**. This menu option is disabled if there are no input nodes that can be tested.

  The Test Client editor is opened with settings from the message flow.
- From the **Flow** menu:
  1. Open the message flow that you want to test.

2. Click **Flow → Test**. This menu option is disabled if there are no input nodes that can be tested.

The Test Client editor is opened with settings from the message flow.

You can now configure a test message to send to the message flow, or configure the Test Client settings.

Next:

You can save the Test Client configuration in a `.mbtest` file:

1. Click **File → Save**. The Save Execution Trace window is displayed.
2. Enter a name for the file, and select a project in which to save the file.
3. Click **Finish** to save the file.

Configuring the test settings

You can configure the settings in the Test Client to control how your tests are run.

Before you start:

You must complete the following tasks before you can configure test settings:

- Creating a message flow
- "Opening the Test Client editor" on page 66

Use the following topics to help you to configure the settings on the Test Client:

- "Testing a message flow that has WebSphere MQ nodes"
- "Testing a message flow that uses JMS nodes" on page 68
- "Test Client Configuration tab" on page 82

You can modify settings that relate to all your test configurations using the Test Client preferences; see "Test Client preferences" on page 84.

Testing a message flow that has WebSphere MQ nodes:

You can configure settings in the Test Client for testing message flows that have WebSphere MQ nodes.

To test a message flow that uses WebSphere MQ nodes:

1. Right-click on your message flow and click **Test Message Flow**. The Test Client opens with the settings from the selected message flow.
2. In the Test Client, click the **Configuration** tab to display the Test Client configuration settings.
3. Click **MQ Settings** and select the appropriate options for your test.
4. Click **MQ Message Header** "**Default Header**" to view the settings for the message header that is used for the test message. You can edit the options for the default header, or alternatively, you can create a new header to edit:
   a. Click **MQ Message Headers**.
   b. Click **Add** and enter a unique name for the header.
   c. Edit the header settings.
   d. Click the **Events** tab, and select the appropriate header for your message from the **Header** list.

5. You can use the Test Client to create WebSphere MQ queues that are used in nodes in your message flow. To configure the Test Client to create the queues:
   a. Click **Window** → **Preferences**.
   b. Expand **Broker Development** and click **Message Broker Test Client**.
   c. Ensure **Create queues of input and output nodes of message flows when host name is localhost** is selected and click **OK**.
6. Create a test message to test your message flow, see "Creating and editing a test message."
7. When you have created your test message, you must select the execution group to deploy the message flow to, see "Selecting the deployment location for the message flow" on page 70.

Testing a message flow that uses JMS nodes:

You can configure settings in the Test Client for testing message flows that uses JMS nodes.

To test a message flow that uses JMS nodes:

1. Right-click on your message flow and click **Test Message Flow**. The Test Client opens with the settings from the selected message flow.
2. In the Test Client, click the **Configuration** tab to display the Test Client configuration settings.
3. Click **JMS Settings** and select the appropriate options for your test. You can add references to the client JAR files used to create the JMS connection.

   To add a reference to these JAR files into your test configurations:
   a. Click **Configure preference settings**. The Test Client preferences are displayed.
   b. Click **Add** and locate the JAR files in your file system.
   c. Click **OK** to add the reference to the JAR files.
   d. Ensure that **Use preference settings** is selected in the **Configuration** tab.
4. Click **JMS Message Headers** to create a JMS header:
   a. Click **Add** and enter a unique name for the header.
   b. Edit the header settings.
   c. Click the **Events** tab, and select the appropriate header for your message from the **Header** list.
5. Create a test message to test your message flow, see "Creating and editing a test message."
6. Select the execution group to which you want to deploy the message flow to, see "Selecting the deployment location for the message flow" on page 70.

Creating and editing a test message

To use the Test Client, you must create or edit a test message to send to your message flow.

Before you start:

Complete these tasks before you edit your test message:

1. "Opening the Test Client editor" on page 66
2. "Configuring the test settings" on page 67

A number of editors are available in the Test Client for creating a test message. The most appropriate editor to use depends upon the type of test message you want to send to your message flow.

- If the input node that you want to send the message to for the test expects an XML message, and the message flow is associated with a message definition, the **Edit as XML structure** editor is available.
- If you want to send an XML message, but do not have a message definition defined, or you want to create a test message that is not in XML format, then you can use the **Edit as text** editor.
- If you want to use an existing test message from a workspace resource or file system file, then you can use the **Import from external file** editor.
- Alternatively you can import an existing test message into the **Edit as text** editor, or take the generated source from the **Edit as XML structure** editor and paste it into the **Edit as text** editor.

Select from the following options to create and edit a test message:

- Edit as XML structure:
  1. In the Events tab of the Test Client, select **Edit as XML structure** from the **Body** list.
  2. Edit the entries in the **Value** column for each field to change the content of the test message.
  3. Right-click the fields in the **Edit as XML structure** editor to see additional options for defining the content of the test message. These options include adding message parts and elements, for example if your message has repeating fields.
  4. You can save your file with the updated test message by clicking **File → Save**.
  5. To view and copy the generated test message, click **Show Generated Source**.
- Edit as text:
  1. In the Events tab of the Test Client, select **Edit as text** from the **Body** list.
  2. Enter the text content for your test message. You can copy content into the editor from another source by right-clicking in the editor and selecting **Paste**, or import content from an existing test message by clicking **Import Source**.
  3. You can save your file with the updated test message by clicking **File → Save**.
- Import from external file:
  1. In the Events tab of the Test Client:
     - If the input node expects an XML message, select **Import from external file** from the **Body** list.
     - If the input node does not expect an XML message, select **Import from external file (Binary and Text)** from the **Body** list.
  2. Select from the following options to locate the file containing your test message:
     - Workspace resource
       a. Click **Workspace**. You can then locate your existing test message from your workspace.
       b. Click **OK** to use the selected resource.
       c. Click **Edit** to open the default editor associated with the resource.
     - File system file
       a. Click **File system**. You can then locate your existing test message from your file system.

b. Click **Open** to use the selected file.

3. You can save your file with the updated test message file details by clicking **File** → **Save**.

Next:

Ensure that you have selected the correct input node to send the test message to in your message flow. Click **Send Message** to send your test message to the selected input node. If this is the first time you have sent a message using this Test Client file, the Deployment Location wizard opens. See "Selecting the deployment location for the message flow."

Selecting the deployment location for the message flow

You can specify the execution group to which to deploy a message flow by using the Test Client from within the Deployment Location wizard.

Before you start:

Before you can test your message flow, you must have configured a broker domain with a broker. The components in the broker domain must all be running. If you do not have an existing broker domain, you can create one using the Default Configuration wizard; see Using the Default Configuration wizard. If the broker domain is not already configured in the Message Broker Toolkit, you can use the Deployment Location wizard to connect to the broker domain by selecting **New Domain Connection**.

When you first send a test message to a message flow using the Test Client, the **Deployment Location** wizard is opened. You can use the wizard to select an execution group to which to deploy the message flow.

1. In the Test Client, click **Send Message**, to open the **Deployment Location** wizard.
2. If your domain is not connected, click **Connect**.
3. From the list in the wizard, select the execution group to which to deploy your message flow. You can also create a new execution group from the **Deployment Location** wizard by clicking **New Exec Group**.
4. Select the **Mode** to run the test in.
5. Select **Run** to monitor the output nodes only.
6. Select **Trace** to receive information about each node that the message passes through in the message flow.
7. Click **Next**.
8. Modify the test settings as required.
9. Click **Finish** to save the settings and deploy the message flow.

Next:

You can change the deployment location settings from the **Configuration** tab.

1. Click **Configuration** in the Test Client.
2. Click **Deployment** to display the deployment settings.
3. Click **Change** to open the **Deployment Location** wizard.

Using the Test Client in trace and debug mode

You can run the Test Client in trace and debug mode to trace the path of a test message through a message flow.

Before you start:

Before you can test your message flow, you must have configured a broker domain with a broker. The components in the broker domain must all be running. You must also have created an execution group to deploy your message flows to.

You can use the trace and debug mode to:

- Stop the test message at breakpoints in the message flow by using the flow debugger.
- Trace the message nodes and terminals that the test message passes through.
- See the test message change as it passes through the message flow.
- View a message node, where an exception occurs, and the associated exception message and trace details.

To use the Test Client in trace and debug mode:

1. Set the Java Debug Port for the execution group by using the mqsichangeproperties command:
   a. Open a command window that is configured for your environment.
   b. Enter the following command:

      ```
      mqsichangeproperties broker -e exec_group -o ComIbmJVMManager
                                          -n jvmDebugPort -v port_number
      ```

      where *broker* is the name of your broker, *exec_group* is the name of your execution group, and *port_number* is an unused port number
   c. Stop and restart your broker using the mqsistop and mqsistart commands.
2. In the Test Client click **Send Message** to open the Deployment Location wizard.
3. If your domain is not connected, click **Connect**. From the list in the wizard, select the execution group to which you want to deploy your message flow.
4. Click **Trace and debug**.
5. Optional: If you want the test message to stop at a breakpoint after the input node, click **Stop at the beginning of the flow during debug**.
6. Click **Next** and modify the test settings as required.
7. Click **Finish** to save the settings and deploy the message flow.

Next:

You can modify the deployment location settings from run mode to trace and debug mode by using the Deployment Location wizard:

1. Click **Configuration** in the Test Client.
2. Click **Deployment** to display the deployment settings.
3. Click **Change** to open the Deployment Location wizard.

Part 3. Reference

Flow application debugger

The flow debugger is a visual interface that supports the debugging of message flow applications in the workbench.

The following topics provide reference information to help you use the debugger effectively:

- "Flow debugger shortcuts"
- "Flow debugger icons and symbols" on page 76

You can also use the "Java Debugger" on page 78 provided by the Java Development tools to debug Java code within the workbench.

Flow debugger shortcuts

You can use function keys and shortcut keys to complete actions in the flow debugger views and windows.

Shortcut keys are shown as a pair that you press together, followed by a subsequent key, for example `Shift-F10, C` means hold the Shift key down and press F10, then release both and press key C.

The tables below describe the main shortcuts that are available in the debug session:

- "Debug view"
- "Breakpoints view" on page 76
- "Flow Breakpoint Properties dialog" on page 76
- "Variables view" on page 76

To see a complete list of all the shortcuts that are available, press `Shift-F10` and release; the contextual menu is displayed.

Debug view

Key combination	Function
Shift-F10, C	Run to completion
Shift-F10, E	Disconnect
F5 or Shift-F10, I	Step into
F8 or Shift-F10, M	Resume
Shift-F10, N	Terminate All
F6 or Shift-F10, O	Step over
Shift-F10, T	Terminate
F7 or Shift-F10, U	Step return
Shift-F10, V	Terminate and Remove

Breakpoints view

Key combination	Function
Shift-F10, A	Select All
Shift-F10, B	Add breakpoint
Shift-F10, D	Disable the selected breakpoints
Shift-F10, E	Enable the selected breakpoints
Shift-F10, L	Remove all breakpoints
Shift-F10, O	Remove the selected breakpoints

Flow Breakpoint Properties dialog

Key combination	Function
E	Enable the breakpoint
Alt-R, <space>	Restrict the breakpoint to the selected flow instances

Variables view

Key combination	Function
Shift-F10, A	Select All
Shift-F10, C	Change the value of the selected flow variable
Shift-F10, V	Copy variables

Flow debugger icons and symbols

This topic describes the icons and symbols used in the Debug perspective and its views:

- "Debug perspective"
- "Debug view" on page 77
- "Message Flow editor" on page 77
- "Breakpoints view" on page 78
- "Variables view" on page 78

Debug perspective

These icons and symbols are used in the Debug perspective *outside* any individual view.

Icon or Symbol	Description
	Debug perspective (symbol)
	Attach to Flow runtime (icon)

Debug view

Icon or Symbol	Description
	Debug view (symbol)
	Flow engine (symbol)
	Flow (symbol)
	Flow instance paused (symbol)
	Flow instance running (symbol)
	Flow instance terminated (symbol)
	Stack frame (symbol)
	Detach from the selected flow engine (icon)
	Resume flow execution (icon)
	Run the flow to completion (icon)
	Step into subflow (icon)
	Step over node (icon)
	Step out of subflow (icon)
	Step into source code (icon)

Message Flow editor

These icons and symbols in the message flow editor are specific to the flow debugger.

Icon or Symbol	Description
	Enabled breakpoint (symbol)
	Disabled breakpoint (symbol)
	Paused at breakpoint (symbol)
	Source code available (symbol)
	Error or exception (symbol)

Breakpoints view

Icon or Symbol	Description
	Breakpoints view (symbol)
	Enabled breakpoint (symbol)
	Disabled breakpoint (symbol)
	Remove selected breakpoints (icon)
	Remove all breakpoints (icon)

Variables view

These icons and symbols in the Variables view are specific to ESQL.

Icon or Symbol	Description
	Variable view (symbol)
	Tree reference variable (symbol)
	Message (symbol)
	ESQL reference variable (symbol)
	ESQL constant (symbol)
	ESQL scalar variable (symbol)
	ESQL schema variable (symbol)
	ESQL module variable (symbol)

Java Debugger

The Java Development Tools include a debugger that enables you to detect and diagnose errors in your programs running on local or remote systems. You can control the execution of your program by setting breakpoints, suspending launched programs, stepping through your code, and examining the contents of variables.

For further information about the Java debugger, refer to the Java Development User Guide plug-in - Debugger.

Test Client

Use the Test Client to test message flow applications.

The following topics provide reference information to help you to use the Test Client effectively:

Test Client Events tab

You can use the **Events** tab in the Test Client to edit the properties and content of your test messages. You also view test events in the **Events** tab when you run the test.

Purpose

Use the **Events** tab to edit and send the test messages, and to monitor the results of the test.

Message Flow Test Events

The **Message Flow Test Events** section of the **Events** tab displays the status and history of the test execution:

Invoke Message Flow
The Invoke Message Flow event starts the selected message flow. The message flow selection is defined by the selected message flow input node. Invoke Message Flow can also be the start of a new test execution with an empty test message. You can enter the message content and start the execution.

Starting
The Starting event indicates the beginning of the test execution.

Sending Message
The Sending Message event indicates that the message is being sent.

Enqueue
The Enqueue event indicates that the current message has been queued on an existing WebSphere MQ queue. Put a message to the specified queue manager, queue, port, and host name as defined in the Detailed Properties section. For more information, see "Enqueue" on page 82.

Dequeue
The Dequeue event indicates that the current message has been dequeued from an existing WebSphere MQ queue. Click **Get Message** from the

specified queue manager, queue, port and host name as defined in the Detailed Properties section. For more information, see “Dequeue” on page 82.

Monitor
The Monitor event indicates the message was received on an output node monitor.

Stopped
The Stopped event indicates that the test execution has been stopped. The test execution can be stopped either by the Test Client or by the user.

Exception
The Exception event indicates when errors are encountered during the test. A message is displayed in the Exception message box on the right of the page and the message flow execution errors are logged in the Windows Event Viewer application log. To view the Windows Event Viewer application log, click **Event Viewer**.

Exception Trace
An Exception Trace event occurs when errors are encountered during test and Trace has been selected. Trace details are displayed in the Exception Trace panel on the right of the page.

Events tab actions

The following actions can be initiated on the **Events** tab, by right-clicking in **Message Flow Test Events**:

Re-run
Reruns the current message. To clone and rerun a previously started test message, right-click the message flow and click **Re-run**.

Duplicate
Duplicates the current message. To duplicate a previously started test message, right-click the message flow and click **Duplicate**.

Invoke
Restarts the current message. To restart a previously started test message, right-click the message flow and click **Invoke**.

Buttons on the Events tab

Several buttons are provided on the top-right of the **Events** tab:

Invoke
Starts a new Invoke Message Flow event where you can enter a request message and start the test execution.

Enqueue
Puts the message to the specified queue manager, queue, port, and host name as defined in the Detailed Properties section on the right of the page.

Dequeue
Gets the message from the specified queue manager, queue, port, and host name as defined in the Detailed Properties section on the right of the page.

Saved Messages
Displays the Data Pool editor in which you can select values that you have used in a previous test session. You can use the Data Pool editor to save or retrieve values for use during tests. When you use the Data Pool editor to

save or retrieve values, the values are saved to, or retrieved from, a global data pool in your workspace. By selecting **Saved Messages**, you ensure that you are always working with the same set of values regardless of how many Test Client instances or test configurations you are using in your tests.

Stop Stops the current test.

Show Event Viewer
Displays the Event Viewer if the operating system is Windows.

Show Console
The Message Broker Runtime Console view opens. This view shows more details of the test run.

Detailed Properties

The Detailed Properties section varies according to the different events that you have selected in the Message Flow Test Events pane. The Detailed Properties section displays details of the current event type. The default details are:

- **Message Flow**. The name of the message flow that is being tested.
- **Input node**. The input node to which the test message is being sent.

Message
The Message section either displays the test message or the output message from a test event. If you are creating a new test message, you can select either the XML Structure editor or the Source editor to edit the test message. The XML Structure editor is available only if the input node that is selected on the message flow is expecting an XML message, and an existing message definition is associated with the message flow.

Header
Select the header to use for your test message if your message flow uses a WebSphere MQ or JMS input node.

SOAP operation
Select the SOAP operation to use for your test message if your message flow uses a SOAP input node.

Viewer
Select the appropriate editor to view or edit your test message or output message from the following editors:

XML Structure editor

Use the XML Structure editor to view and edit an XML test message derived from an associated message definition. You can change the content of the message by editing the entries in the Values column. Right-click to display a menu with options to change the content of the XML structure, including adding and removing elements. To view the generated source code, click **Show Generated Source**. Click **Saved messages** to display a list of saved error messages.

Source editor
Use the Source editor to compose and send test messages if you want to import a test message, or if the test message is not in XML format. To import a test message from a file, click **Import Source**.

Source tab
Edit the test message as plain text using the **Source** tab.

XML Source tab
Edit the test message in an XML editor using the **XML Source** tab.

Hexadecimal (read only) tab
View the test message in hexadecimal format by using the **Hexadecimal (Read Only)** tab.

Enqueue

Enqueue is the term that is used to describe the process of putting a message on to a WebSphere MQ queue.

The following properties are entered on the Events page:

Host The name of the host computer.

Port The number of the port that is used.

Queue Manager
The name of WebSphere MQ Queue manager.

Queue
The name of the queue on the Queue manager.

Dequeue

Dequeue is the term that is used to describe the process of removing a message from a WebSphere MQ queue.

The following properties are entered on the Events page:

Host The name of the host computer.

Port The number of the port that is used.

Queue Manager
The name of WebSphere MQ Queue manager.

Queue
The name of the queue on the Queue manager.

Test Client Configuration tab

Configure your test environment in the **Configuration** tab in the Test Client.

Purpose

You can use the Test Client **Configuration** tab to alter the settings that are used when you test your message flow.

You can set the following settings in the **Configuration** tab:

- Message Flows
- Deployment
- MQ Settings
- JMS Settings
- MQ Message Headers

- JMS Message Headers

Select the relevant options in the left pane to display the properties in the right pane of the **Configuration** tab. Details of the properties on the **Configuration** tab are given in the following sections.

Message Flows

Add or remove Message Flows to be tested
In this section of the **Configuration** tab, the message flow that you have selected to test is listed. You can add more message flows to the test configuration so that you can test multiple message flows at the same time; for example, if you have an output message from one message flow that triggers another message flow, or if you are using subflows. Click **Add** to add message flows to your test configuration. Click **Remove** to remove message flows from the configuration.

Deployment

I will deploy the specified Broker Archive manually
You can select this option to prevent the Test Client from deploying the message flow before you send a message to test the message flow. Specify the broker archive (`.bar`) file that you want to deploy manually in the **Specify Broker Archive file** property on the **Deployment** section of the **Configuration** tab. Deploy the broker archive file to the execution group that is specified in the Deployment Location section.

Always rebuild and deploy a Broker Archive automatically
The Test Client always builds and deploys the file irrespective of whether there is a change to the broker archive file or its dependents, including message flows.

You can use this option to force the Test Client to deploy when the Test Client cannot detect changes in the message flow.

Only rebuild and deploy Broker Archive when changes occur
The Test Client rebuilds and deploys the broker archive file only when there is a change in the content of the message flow. This action is the default option.

Override configurable properties when rebuilding Broker Archive file
You can define the configurable properties in the Broker Archive editor. Use this option to specify whether the user-defined value is overridden when the Test Client rebuilds the broker archive file.

Click **Change** to select a deployment location, see the "Deployment Location wizard" on page 85.

MQ Settings

Stop when first MQ message is received
Use this option if you want the Test Client to stop receiving events when the first WebSphere MQ Output queue contains a message.

Add one or more conditions. This option is ignored if no MQ queues are being monitored.

The Test Client monitors MQ queues that are defined in the MQOutput nodes in the message flows that are being tested. If you check the option,

it instructs the Test Client to stop testing when the message reaches any of the WebSphere MQ queues that are being monitored.

Select Purge or Browse option
Use this option to either purge a message from the queue or to browse the messages on the output queue.

Queue manager connection parameters
Enter the character set ID to use for connection to your queue manager.

JMS Settings

Stop when first JMS message is received
Use this option to stop the Test Client receiving events when the first JMS Output destination contains a message.

Specify JMS Client JARs
Use this option to add and remove JAR files that are used to create JMS connections.

Select **Use preference settings** to configure preference settings. This option is a global setting that can be applied to all Test Clients in the same workbench.

WMQ Message Headers

WMQ Message Headers
Use this option to build multiple MQMD header definitions. When you send a message to a message flow that contains an MQInput node, you have the option to select an MQMD with suitable values for your test. Click **Add** to enter additional MQMD headers. Each new header is listed under 'MQ Message Headers' in the left navigator column. Each MQMD definition name must be unique within the Test Client configuration file.

MQ Message Header "Default Header"
This options specifies the default MQ Message Header definition. You can edit the values in this definition for your test configuration, or create a new WebSphere MQ Message Header definition.

JMS Message Headers

JMS Message Headers
Use this option to enter multiple JMS Message Headers. Click **Add** to enter additional headers. Each new header is listed under 'JMS Headers' in the left navigator column.

Select each JMS Message Header to view and change the settings.

JMS Header
Enter the values for your JMS configuration in the JMS Header page.

Test Client preferences

You can set test settings for the local test environment that are used by the Websphere Message Broker Test Client.

You can configure the settings for the test in the Deployment Location wizard, or on the Test Client preferences. If you configure the settings on the Test Client preferences, these settings are applied to all tests. You can access the Test Client preferences by using the following instructions:

1. Click **Window** → **Preference**. The Preferences dialog is displayed.

2. Expand the item for Broker Development on the left and click Message Broker Test Client.

Use the following fields to define the settings for the local test engine:

Seconds to wait for deployment completion
The amount of time in seconds to allow for deployment. The default value is 20.

Seconds to wait after Test Client complete the deployment
The amount of time in seconds to wait after the Test Client completes deployment. The default value is 0.

Seconds to wait on launching the debugger for tracing purposes
The amount of time in seconds to wait before launching the debugger. The default value is 20.

Show information dialog before disconnecting the debugger
Select this option to display a dialog when you use the Test Client in Run mode if the flow debugger is already connected.

Seconds to wait for test client to stop
The amount of time in seconds to wait before ending the test. When the number of seconds has elapsed, monitoring of output nodes is stopped, and a Timeout event and a Stop event are displayed in the Test Client. The default value is 120.

Also select the required options:

Create queues of input and output nodes for message flows when host name is localhost
Use this option to create queues for the local host.

Add or modify (but not clear) what has already been deployed on the execution group Use this option to add or change, but not delete, what has previously been deployed on the current execution group.

Deployment Location wizard

Use the Deployment Location wizard to set the execution group to which the test message flow is deployed. You can also use the wizard to create a domain connection, a broker, and a new execution group.

The Deployment Location wizard is displayed when you click **Send Message** on the **Events** tab of the Test Client the first time that you run a test.

You can run the Test Client in one of the following modes:

- Run mode: test events are produced when the test message has been sent successfully from the message flow, or when an error occurs. Although you can also use the debugger, and the Test Client execution halts at the breakpoints, the Test Client does not receive and record message node-level trace information. When you use the Test Client in this mode, you are given the option to disconnect the flow debugger if it is already running. To use the Test Client in Run mode, ensure that **Trace and Debug** is not selected on the Deployment location wizard.
- Trace and debug mode: test trace events are produced when the test message leaves each node so that you can see the path that the message takes through the message flow. The results are stored in the broker. If you save the message flow test file, you can view the results at a later time. The flow debugger is

launched in this mode, if it is not already connected. The test message stops at any breakpoints that you have configured in the message flow. You must configure a Java Debug port to run the test in the trace and debug mode.

If you use the Test Client in trace and debug mode, you can select the option **Stop at the beginning of the flow during debug**. This option suspends the test at the beginning of the message flow by setting a breakpoint on every connection after the selected input node. When you use the Test Client in the trace and debug mode, you can choose to use the flow debugger in the Debug perspective.

You can also use the Deployment Location wizard to specify settings for the test. For more information about the test settings, see "Test Client preferences" on page 84.

After you run a test for the first time, you can access the settings for the deployment location by clicking **Change** in the **Deployment** section of the **Configuration** tab.

You can create broker domain components and associated resources by selecting the options that are described in the following table.

Resource	Option	Action
Domain connection	New Domain	Launches the Create a Domain Connection wizard.
Configuration Manager	New Domain New Broker	Launches the Create a Domain Connection wizard. Launches the Create a Broker Reference wizard and connects to the domain if the domain is not already connected.
Broker topology	New Domain Connection New Exec group	Launches the Create a Domain Connection wizard. Launches the Create an Execution Group wizard.

JMS events in the Test Client

Use the information in this topic to help you to understand JMS events in the Test Client.

When you test a message flow that contains JMS nodes, you might see some of the following events in the Message Broker Test Events section of the Test Client.

JMS Information event

The event details are displayed in the following properties.

Property name	Description
Initial Context Factory	The initial context factory that is used for JNDI look up
Location JNDI binding	The JNDI URL that is used for JNDI look up
Connection Factory	The JNDI name of the Connection Factory
Message Properties	The properties of the JMS message

Property name	Description
Message Header	The header of the JMS message

The message content is displayed in the XML Structure or Source view.

JMS Response event

The event details are displayed in the following properties.

Property name	Description
Initial Context Factory	The initial context factory that is used for JNDI look up
Location JNDI binding	The JNDI URL that is used for JNDI look up
Connection Factory	The JNDI name of the Connection Factory
Message Properties	The properties of the JMS message
Message Header	The header of the JMS message

The message content is displayed in the XML Structure or Source view.

JMS Exception event

The event details are displayed in the following properties.

Property name	Description
Initial Context Factory	The initial context factory that is used for JNDI look up
Location JNDI binding	The JNDI URL that is used for JNDI lookup
Connection Factory	The JNDI name of the Connection Factory

The detailed trace message is displayed as well as the Exception trace details.

Part 4. Appendixes

Appendix. Notices for WebSphere Message Broker

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this information in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this information. The furnishing of this information does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

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.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,
Hampshire,
England
SO21 2JN

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction or intent are subject to change or withdrawal without notice, and represent goals and objectives only.

This information includes examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not

been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Each copy or any portion of these sample programs or any derivative work, must include a copyright notice as follows:

(C) (*your company name*) (*year*). Portions of this code are derived from IBM Corp. Sample Programs. © Copyright IBM Corp. *_enter the year or years_*. All rights reserved.

Trademarks in the WebSphere Message Broker information center

IBM, the IBM logo, and ibm.com are trademarks or registered trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at Copyright and trademark information at `www.ibm.com/legal/copytrade.shtml`.

Adobe is a registered trademark of Adobe Systems Incorporated in the United States, and/or other countries.

Intel, Itanium, and Pentium are trademarks of Intel Corporation in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft and Windows are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Other company, product, or service names may be trademarks or service marks of others.

Index

P

R

S

T

W

Printed in USA