

IBM MQ



Scenarios

Version 8 Release 0

Note

Before using this information and the product it supports, read the information in "Notices" on page 109.

This edition applies to version 8 release 0 of WebSphere MQ and to all subsequent releases and modifications until otherwise indicated in new editions.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

© **Copyright IBM Corporation 2007, 2018.**

US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

Contents

Figures v

Tables vii

Scenarios 1

Getting started with IBM MQ.	1
Planning the solution	1
Implementing the solution.	3
What to do next.	16
Other learning resources	17
Point-to-point scenario.	17
Planning the solution	18
Implementing the solution	19
Securing the point-to-point topology	26
Publish/subscribe scenarios	29
Publish/subscribe cluster scenario.	29
Publish/subscribe hierarchy scenarios	34
Transactional support scenarios.	44
Introducing units of work	45
Scenario 1: Queue manager performs the coordination	46

Scenario 2: Other software provides the coordination	71
Expiring global units of work	79
Unit of recovery disposition	79
Security scenarios	80
Security scenario: two queue managers on z/OS	80
Security scenario: queue-sharing group on z/OS	87
Connecting two queue managers using SSL or TLS	93
Connecting a client to a queue manager securely	100
File transfer scenario	106

Index 107

Notices 109

Programming interface information	110
Trademarks	111

Sending your comments to IBM . . . 113

Figures

1. Put message on LQ1, get message from LQ1.	2	10. Sample XAResourceManager entry for Sybase on UNIX and Linux platforms	62
2. QM1 sends a message to QM2	18	11. Sample XAResourceManager entries for multiple Db2 databases	63
3. Topology diagram showing the relationship between queue managers in a typical publisher/subscribe hierarchy.	35	12. Sample XAResourceManager entries for a Db2 and Oracle database	63
4. Topology diagram showing the relationship between queue managers in a typical publisher/subscribe hierarchy.	39	13. Sample dspmqtrn output	66
5. Topology diagram showing the relationship between queue managers that are using a cluster channel.	42	14. Commented- out XAResourceManager stanza on UNIX and Linux systems	68
6. Sample XAResourceManager entry for Db2 on UNIX platforms	55	15. Example security scenario.	80
7. Sample XAResourceManager entry for Oracle on UNIX and Linux platforms	58	16. Configuration resulting from this task.	94
8. Sample XAResourceManager entry for Informix on UNIX platforms	60	17. Configuration resulting from this task.	96
9. Example contents of \$SYBASE/ \$SYBASE_OCS/xa_config.	61	18. Queue managers allowing one-way authentication.	98
		19. Configuration resulting from this task	101
		20. Configuration resulting from this task	103
		21. Client and queue manager allowing anonymous connection	105

Tables

1. What happens when a database server fails	48	13. Sample security profiles for queue manager QM2 using TCP/IP and not SSL	85
2. What happens when an application program fails	48	14. Sample security profiles for queue manager QM2 using LU 6.2	86
3. XA switch function pointers	49	15. Sample security profiles for queue manager QM2 using TCP/IP and SSL	86
4. Summary of XA function calls	70	16. Sample security profiles for the CICS application on queue manager QM1	87
5. XA switch load file names	72	17. Security profiles for the example scenario	90
6. Alternative 64-bit XA switch load file names	72	18. Sample security profiles for the batch application on queue manager QM1	91
7. 64-bit transaction managers that require the alternative 64-bit switch load file	73	19. Sample security profiles for queue manager QM2 using TCP/IP	91
8. Essential code for CICS applications: XA initialization routine	75	20. Sample security profiles for queue manager QM2 using LU 6.2	92
9. Installation directories on Windows, UNIX and Linux operating systems	75		
10. CICS task termination exits	77		
11. Security profiles for the example scenario	84		
12. Sample security profiles for the batch application on queue manager QM1	85		

Scenarios

Each scenario walks you through a significant set of tasks, and helps you to configure a major product feature. The scenarios include useful links to other content to help you to gain a better understanding of the area in which you are interested.

The available IBM® MQ scenarios are described in the following subtopics. The *IBM Product Connectivity Scenarios and Patterns* product documentation provides worked examples of using several IBM products (for example, IBM MQ and WebSphere® Application Server) connected together.

Related information:

IBM Product Connectivity Scenarios and Patterns product documentation

Getting started with IBM MQ

This scenario explains how to get started with IBM MQ on a Windows platform. Use this scenario if you have never used IBM MQ and want to get started quickly.

This scenario describes the basic steps for installing, configuring and verifying IBM MQ on Windows if you do not already have it installed on your system. You can complete the steps of the scenario by using either the graphical user interface or command-line interface.

This scenario was tested using IBM MQ Version 8.0.0, Fix Pack 2 on a Windows 7 Professional 64-bit (SP 1) operating system.

Planning the solution

Choose a method for installing IBM MQ on Windows. Use the graphical user interface and wizards that take you through the installation and configuring process or use the command line to conduct a silent installation.

Overview: The delivered logical topology

The delivered logical topology after completing the scenario.

The installed IBM MQ server instance allows for creation of IBM MQ objects: queues and queue managers. You can use the MQ Explorer to put and get messages from the local queue through the queue manager. After this scenario is complete, the delivered topology will look like Figure 1.

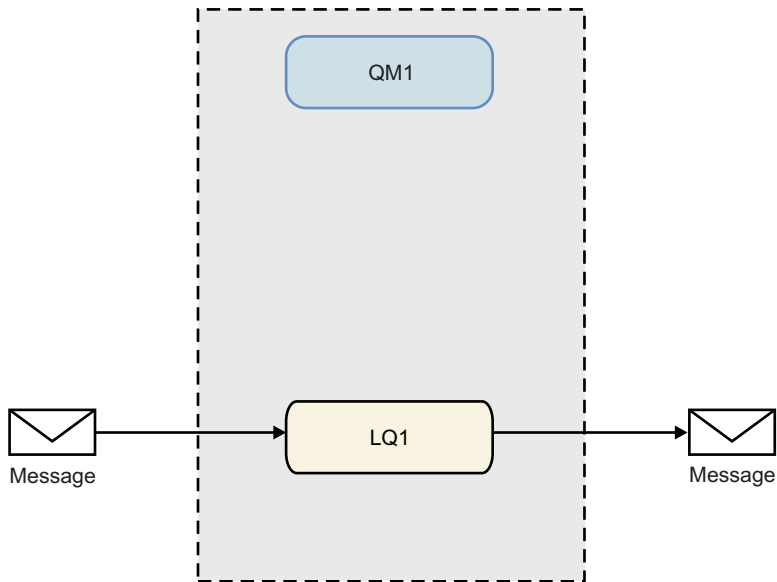


Figure 1. Put message on LQ1, get message from LQ1.

Basic concepts and key terms

Description of the basic concepts and key terms you must know about before using the Getting started with IBM MQ scenario.

Basic concepts

IBM MQ enables applications to read and write messages to a queue. The application that reads the message is independent of the application that writes the message. It is not a requirement to have the two applications running at the same time. If no application is available to read the message it is queued on the IBM MQ queue until an application reads it.

In this scenario you can choose to install and configure IBM MQ in one of the following ways:

“Installing and configuring using the graphical user interface” on page 3

During installation using the graphical user interface, you are guided through several wizards to help you apply the relevant options and settings:

Launchpad

Check software requirements, specify network information and start the IBM MQ installation wizard.

IBM MQ installation wizard

Install the software and start the Prepare IBM MQ wizard.

Prepare IBM MQ wizard

Start the IBM MQ service and MQ Explorer.

MQ Explorer

Manage queues and queue managers, access Default Configuration wizard and the Postcard application.

Default configuration wizard

Create IBM MQ objects and put and get messages to and from the queue, to test the installation was successful.

Postcard application

Exchange messages between two users to verify the installation.

“Installing and configuring using the command line interface” on page 9

The command line interface installation can be silent or interactive. The silent installation is fully accessible and is the one covered in this scenario. During installation using the command line, you are guided through several steps to help you apply relevant options and settings:

- Install IBM MQ
- Create and configure IBM MQ objects; queue managers and queues.
- Verify the installation by using `amqsput` to put and `amqsget` to get a message from the queue.

As well as using MQ Explorer and command line to create IBM MQ objects, it is possible to do so by using the programmable interface. This is not included in the current scenario.

Key terms

Here is a list of key terms about message queuing.

Term	Description
Queue managers	The queue manager is responsible for maintaining the queues it owns, and for storing all the messages it receives onto the appropriate queues.
Messages	A message is a string of bytes that is meaningful to the applications that use it. Messages are used to transfer information from one application program to another. The applications can be running on the same or on different computers.
Local queues	A local queue is a data structure used to store messages. The queue can be a normal queue or a transmission queue. A normal queue holds messages that are to be read by an application that is reading the message directly from the queue manager. A transmission queue holds messages that are in transit to another queue manager.

Implementing the solution

Implement the solution to the scenario. Install IBM MQ on Windows and create IBM MQ objects: queue managers and queues. Use sample applications to verify your installation by putting and getting messages to and from a queue.

Installing and configuring using the graphical user interface

Install IBM MQ on Windows by using the installation launchpad, and then use the Postcard application to verify the installation. After verifying your installation, create a queue manager and a queue and then try putting a message to the queue and getting a message from the queue.

This scenario was tested with IBM MQ Version 8.0.0.0 on a Windows 7 Professional 64-bit (SP 1) operating system.

Installing using the launchpad:

Install IBM MQ on Windows using the installation launchpad, and verify your installation by using the Postcard application.

Before you begin

Before completing this task, complete the following checks:

- You must have local administrator authority when you are installing. Define this authority through the Windows facilities.
- Ensure that the machine name does not contain any spaces.
- Ensure that you have sufficient disk space, up to 1005 MB, to fully install IBM MQ for Windows.
- Determine whether you need to define Windows domain user ID's for any IBM MQ users.

Before you install IBM MQ, check that your system meets the hardware and software requirements. For the latest details of hardware and software requirements on all supported platforms, see IBM MQ System Requirements.

About this task

The launchpad and subsequent wizards take you through the installation process and help you review the software requirements and IBM MQ settings. You are also taken through initial tasks for creating a default queue manager, a local queue and can verify the installation by using the Postcard application.

This task assumes that you are installing IBM MQ for the first time on your machine, and that you will be using the default locations. By default the location of the IBM MQ program files are C:\Program Files\IBM\WebSphere MQ, and the data and log file location is C:\ProgramData\IBM\MQ.

Note: If you are installing IBM MQ Version 8.0, and you have any previous installations of IBM MQ on your machine, the location of the program and data files will be different from the default. For further information, see Program and data directory locations. If you have already previously completed this scenario, and want to repeat it with a single, fresh installation using the default locations, remove your previous installation before starting the scenario again. To uninstall an existing instance of IBM MQ from your machine, see “Uninstalling IBM MQ” on page 15.

The installation programs contain links to further information if you require it during the installation process.

Procedure

1. Start the Launchpad, review, and if necessary, modify the software requirements and network configuration.
 - a. Navigate to the IBM MQ software directory, and double click the file Setup.exe to start the Launchpad.
 - b. Select the **Software Requirements** tab to display the **Software Requirements** settings.
 - c. Check that the software requirements have been met and that the entry for the requirement displays a green tick with the words OK. Make any indicated corrections.

Note:

For details of any requirement, click the check box to expand an information tab.

- d. Select the **Network Configuration** tab to display the **Network Configuration** settings.
- e. Select **No**.

Note: This scenario assumes that you do not need to configure a domain user ID for IBM MQ. For more information regarding configuring IBM MQ for Windows domain users, click **More information**.

- f. On the **IBM MQ Installation** tab of the Launchpad, select the installation language, and then click **Launch IBM MQ Installer** to start the IBM MQ installation wizard.

You have completed setting up IBM MQ by meeting or specifying your installation requirements, and have started the IBM MQ installation wizard.

2. Use the IBM MQ installation wizard to install the software, and start the Prepare IBM MQ wizard.
 - a. In the Prepare IBM MQ wizard, read the License Agreement and click the **I accept the terms in the license agreement** check box, and then click the **next** button.
 - b. Click **Typical**, and then click **Next**.
 - c. In the **Ready to Install IBM MQ** page, review the installation information and click the **Install** button.

Note: Note the following details:

- Installation Name
- Top-level folder for Program Files
- Top level folder for Data Files

The following features are installed:

- IBM MQ Server
- IBM MQ: a graphical interface for administering and monitoring IBM MQ resources
- Java™ and .NET Messaging and Web Services
- IBM MQ Development Toolkit

The installation process begins. Depending on your system the installation process can take several minutes.

At the end of the installation process, the IBM MQ Setup window displays the message **Installation Wizard Completed Successfully** .

- d. Click **Finish**.

You have successfully installed IBM MQ. The Prepare IBM MQ wizard starts automatically, displaying the **Welcome to the Prepare IBM MQ Wizard** page.

3. Use the Prepare MQ wizard to start the IBM MQ service and start the Default Configuration wizard.

Note:

You cannot create the default configuration if you have already created other queue managers; you must first delete the other queue managers then run the Default Configuration wizard. To delete a queue manager, see **Steps for deleting a queue manager**

- a. On the **Welcome to the Prepare IBM MQ Wizard**, select **Next**. The Prepare IBM MQ Wizard displays the message **Status: Checking IBM MQ Configuration** and a progress bar. When the process is complete the IBM MQ Network Configuration page is displayed.
- b. On the IBM MQ Network Configuration page of the Prepare IBM MQ Wizard, select **No**.
- c. Click **Next**. The Prepare IBM MQ Wizard displays a message **Status: starting the IBM MQ Service**, and a progress bar. When the process is complete the wizard displays the **Completing the Prepare IBM MQ Wizard**.
- d. Select **Launch IBM MQ Explorer** and choose whether to view the release notes, and then click the **Finish** button. IBM MQ Explorer starts.

You have installed IBM MQ. You have also started the MQ Explorer.

4. Optional: If you want to use the IBM MQ Postcard application to verify your installation, create the default configuration.
 - a. If the Content page is not already displayed, click **Window > Show View > MQ Explorer - Content** to display it.
 - b. Click **Create the Default Configuration**. The IBM MQ Default Configuration window is opened.
 - c. Click **Set up Default Configuration**. The Default Configuration Wizard is opened.
 - d. Click **Next** and **Next** again to move through the information pages.
 - e. On the Default Configuration page, clear both **Allow remote administration of the queue manager** and **Join the queue manager to the default cluster** options, and then click **Next**. Take a note of the queue manager name, as you will need this later, when using the Postcard application.
 - f. On the summary page, click **Finish**. The Default Configuration Wizard is closed, and a dialog box displaying the message Setting up the default default configuration. When this is complete, focus is returned to the IBM MQ Default Configuration dialog box, and the following message is displayed: Default configuration is partially complete .

Note: The IBM MQ Default Configuration may also display the message: Join the default cluster by clicking "Join default cluster" to complete the default configuration on this computer . This is only necessary if you want to join a cluster. This is not covered in this scenario, as it is beyond the scope of this discussion.

- g. Click **Close**.

The Default Configuration is now set up, and you are ready to verify your installation.

5. Optional: If you created a default configuration, verify your installation by using the Postcard application that is supplied with IBM MQ. You can start two instances of the Postcard application and exchange messages between them.

If you did not create a default configuration, you can still follow these steps to verify your installation, but you must first configure a queue manager as described in the previous task of this scenario.

Note:

Running the Postcard application on a non-default configuration automatically creates a queue called postcard on your queue manager. You can delete this queue after using the postcard application.

- a. If the Content page is not already displayed, click **Window > Show View > MQ Explorer - Content** to display it.
- b. Click **Launch Postcard** to open the Postcard - Sign On window.
- c. Enter a nickname for the first user, for example: Jim and click **OK**.
- d. On the IBM MQ Postcard Network window select **Continue on this computer only** and click **OK**. The Postcard application for 'jim' opens.
- e. Move Jim's Postcards to one side of your screen, then start a second Postcard.
- f. Click **Launch Postcard** to open the Postcard - Sign On window.
- g. Enter a nickname for the second user, for example: Sue and click **OK**.
- h. On the IBM MQ Postcard Network window select **Continue on this computer only** and click **OK**. The Postcard application for 'sue' opens.
- i. In the Postcard - 'jim' window enter sue in the **To:** textbox.
- j. In the Postcard - 'jim' window enter the name of the queue manager you defined in step 4e, or taken from the entry **On:** beneath the **Message:** textbox.
- k. In the Postcard - 'jim' window enter a message, for example Hi Sue! in the **Message:** textbox
- l. Click **Send** button to send the message to user Sue.
- m. Observe the message received by user Sue, in the Postcard - 'sue' application window. To view the received message double click on the entry in the Postcards sent and received grid.

Note: Click **Help** in this and other Postcard application windows to view further instructions about running the Postcard application.

You have verified your IBM MQ installation by using the Postcard application.

Results

IBM MQ is installed and verified, and you are ready to configure objects such as queue managers and queues.

What to do next

Follow the instructions in “Creating a queue manager called QM1.”

Related information:

Disc space requirements

Hardware and software requirements on Windows systems

Introduction to IBM MQ

Installing an IBM MQ server

Verifying the installation using the Postcard application

Post installation tasks

Creating a queue manager called QM1:

Create a queue manager, called QM1 by using the MQ Explorer. Queue managers are the main components in an IBM MQ messaging network.

Before you begin

You must have IBM MQ installed. If you do not, see “Installing using the launchpad” on page 4 for information about how to do so.

About this task

In this example, all names are typed in uppercase and because IBM MQ names are case-sensitive, you must type all names in uppercase too.

To create and start a queue manager by using the MQ Explorer, complete the following steps.

Procedure

1. Start MQ Explorer as an administrator.
2. In the Navigator view, right-click the Queue Managers folder, then click **New > Queue Manager**. The Create Queue Manager wizard starts.
3. In the **Queue Manager name** field, type QM1.
4. Select the Make this the default queue manager check box.
5. In the **Dead-letter queue** field type SYSTEM.DEAD.LETTER.QUEUE. This is the name of the dead-letter queue that is automatically created when you create the queue manager.
6. Leave the other fields empty and click **Finish**, or if that button is disabled, click **Next**. The **Finish** button is disabled if the port number conflicts with an existing queue manager, for example the queue manager that is created as part of the default configuration. You must continue through the wizard to change the default port number.
7. If you clicked **Next**, continue to accept the defaults and click **Next** on each page until you get to the final page of the wizard, when the **Finish** button becomes available. Change the specified port number, for example to 1415, and click **Finish**.

IBM MQ displays a Creating Queue Manager dialog window while the queue manager is created and started.

What to do next

To create a queue, see “Creating a queue called LQ1.”

Related information:

Creating and managing queue managers on distributed platforms

Creating a queue called LQ1:

Create a queue by using the MQ Explorer. Queues are data structures that are used to store messages and are IBM MQ queue manager objects.

About this task

In this task you can create IBM MQ objects using the MQ Explorer.

To create and start a queue by using the MQ Explorer, complete the following steps.

Procedure

1. In the Navigator view, expand the **Queue Managers** folder.
2. Expand queue manager **QM1**.
3. Right-click the **Queues** folder, then click **New > Local Queue...** The New Local Queue wizard starts.
4. In the **Name** field, type LQ1.
5. Click **Finish**.

The new queue LQ1, is displayed in the Content view. If the queue is not displayed in the Content view, click on the **Refresh** button, at the top of the Content view.

What to do next

You are ready to put a message on to your queue. To put a message in a queue, see “Putting a message to the queue LQ1.”

Putting a message to the queue LQ1:

Put a message on to the queue LQ1 by using the MQ Explorer.

About this task

This task assumes that you have already created a queue manager called QM1 as described in “Creating a queue manager called QM1” on page 12 and a queue called LQ1 as described in “Creating a queue called LQ1.”

To put a message to the queue by using the MQ Explorer, complete the following steps.

Procedure

1. In the Navigator view, expand the **Queue Managers** folder.
2. Expand queue manager QM1, which you created.
3. Click the **Queues** folder. The queue manager's queues are listed in the Content view.
4. In the Content view, right-click the local queue LQ1, then click **Put Test Message...** The **Put test message** dialog opens.

5. In the **Message data** field, type some text, for example Hello World, then click **Put message**. The **Message data** field is cleared and the message is put on the queue.
6. Click **Close**. In the Content view, notice that the LQ1 **Current queue depth** value is now 1. If the **Current queue depth** column is not visible, you might need to scroll to the right of the Content View.

What to do next

To get a message from the queue, see “Getting a message from the queue LQ1.”

Getting a message from the queue LQ1:

Get a message from the queue LQ1 by using the MQ Explorer.

About this task

This task assumes that you have already put a message QM1 as described in “Putting a message to the queue LQ1” on page 8.

To get a message from the queue by using the MQ Explorer, complete the following steps.

Procedure

1. In the Navigator view, expand the **Queue Managers** folder, then expand QM1.
2. Click the **Queues** folder.
3. In the Content view, right-click the local queue LQ1, then click **Browse Messages....** The Message browser opens to show the list of the messages that are currently on QM1.
4. Double-click the last message to open the properties dialog.
On the **Data** page of the properties dialog, the **Message data** field displays the content of the message in human-readable form.

What to do next

Follow the instructions in subsequent scenarios to explore further IBM MQ features.

To learn about writing queuing applications, connecting to and disconnecting from a queue manager, publish/subscribe, and opening and closing objects, see Writing a procedural application for queuing.

Installing and configuring using the command line interface

Install IBM MQ on Windows by using the command line to perform a silent installation and set up the environment variable. After verifying your installation, create a queue manager and a queue and then try putting a message to the queue and getting a message from the queue.

About this task

This scenario was tested with IBM MQ Version 8.0.0.2 on a Windows 7 Professional 64-bit (SP 1) operating system.

Installing using a silent installation:

Install IBM MQ on Windows by using the command line to perform a silent installation and confirm that the environment for your installation is set up correctly.

Before you begin

Before you start this task, complete the following checks:

- You must have local administrator authority when you are installing. Define this authority through the Windows facilities.
- Ensure that the machine name does not contain any spaces.
- Ensure that you have sufficient disk space. You need up to 1005 MB to fully install IBM MQ Version 8.0 for Windows.
- Determine whether you need to define Windows domain user IDs for any IBM MQ users.

Before you install IBM MQ, check that your system meets the hardware and software requirements. For the latest details of hardware and software requirements on all supported platforms, see IBM MQ System Requirements.

About this task

This scenario assumes that you are installing IBM MQ for the first time on your machine, and that you are using the default locations. By default the location of the IBM MQ Version 8.0 program files are C:\Program Files\IBM\WebSphere MQ, and the data and log file location is C:\ProgramData\IBM\MQ.

Note: If you have any previous installations of IBM MQ on your machine the default locations of the program and data files might change. For further information, see Program and data directory locations. If you have already previously completed this scenario, and want to repeat it with a single, fresh installation using the default locations, remove your previous installation before starting the scenario again. To uninstall an existing instance of IBM MQ from your machine, see “Uninstalling IBM MQ” on page 15.

IBM MQ on Windows uses the MSI technology to install software. For more information on installing using the MSI technology, see Advanced installation using msixec.

To install IBM MQ using the command line, you must specify the following parameters:

- **/i** "**<WMQ_INSTALLATION_MEDIA>\MSI\IBM WebSphere MQ.msi**" where **<WMQ_INSTALLATION_MEDIA>** is the location of the IBM WebSphere MQ.msi file. This argument specifies the location of the .msi file.
- **/l*v** **<USER_LOGFILE_LOCATION>\install.log** where **<USER_LOGFILE_LOCATION>** is where you want the installation logs to be written to.
- **/q** this parameter must be used to perform the silent installation.
- **USEINI="<RESPONSE_FILE>"** where **<RESPONSE_FILE>** is the name and location of the response file to be used by the silent installation. This scenario uses the sample Response.ini file, which is included in the IBM MQ installation media.
- **TRANSFORMS="<TRANSFORM_FILE>"** where **<TRANSFORM_FILE>** is the name of the transform file to be applied to the installation. This scenario uses the American English transform, 1033.mst.
- **AGREETOLICENSE="YES"** this parameter must be included, or the installation can not complete.
- **ADDLOCAL="Server"** this parameter lists which components to install.

Procedure

1. Use the command line to conduct a silent installation.

- a. To invoke the silent installation from an elevated command prompt, click the **Start button** on your **Windows taskbar** and type `cmd` in **search programs and files** field. Right click the `cmd.exe` program and select **Run as administrator**.
- b. In the Windows command prompt, enter the following command:

Note: The command is presented on multiple lines here, but it must be typed out on one line.

```
msiexec /i "<MQ_INSTALLATION_MEDIA>\MSI\IBM WebSphere MQ.msi"  
/l*v c:\wmqinslogs\install.log  
/q USEINI="<MQ_INSTALLATION_MEDIA>\Response.ini"  
TRANSFORMS="1033.mst"  
AGREETOLICENSE="yes"  
ADDLOCAL="Server"
```

Where `<MQ_INSTALLATION_MEDIA>` is the path to your IBM MQ installation media.

Note:

After you input the command, the command line will return the prompt.

- c. To view the installations progress, open the log file that you specified. If the installation completed successfully, you see the message `Product: IBM MQ (Installation1) -- Installation operation completed successfully.` two paragraphs up from the bottom of the log file.
 - d. When the installation is complete, the service starts and the IBM MQ icon appears in the system tray. You have installed IBM MQ, and you have started the IBM MQ service.
2. Set up environment variables for your installation by using the `setmqenv` command.
 - a. Enter the following command in the command line:

Note: If you used the default location, the path to your installation will be `C:\Program Files\IBM\WebSphere MQ`.

```
"<MQ_INSTALLATION_PATH>/bin/setmqenv" -s
```

where `<MQ_INSTALLATION_PATH>` refers to the location where IBM MQ is installed. Ensure you enclose the path to `setmqenv` in the `bin` folder, in quotation marks, to prevent the prompt returning an error.

- b. Check that the environment is set up correctly by entering the following command:

```
dspmqver
```

If the command completes successfully, and the expected version number and installation name are returned, the environment is correctly set up. For this scenario the message should include the line:

```
Version: 8.0.0.2
```

and if you did not specify a non-default installation name, the line:

```
InstName: Installation1
```

You have successfully installed IBM MQ using a silent installation.

Results

You have conducted an IBM MQ silent installation and confirmed that your environment is set up correctly.

What to do next

- You can run the Prepare WebSphere MQ wizard. For more information, see Prepare IBM MQ Wizard.
- Follow the instructions in “Creating a queue manager called QM1” on page 12.

If you encounter any issues during the installation, check the installation log, at the location that you specified in the **msiexec** command, in this scenario the location of the log file is: `c:\wmqins\logs\install.log`. Take any action that is specified in the log and rerun the installation again. You can also check the parameters that you passed with the command, making sure you are including all the required parameters.

Related information:

Advanced installation using msiexec

Using transforms with msiexec

Installing IBM MQ

Creating a queue manager called QM1:

Create a queue manager, called QM1 by using the command-line interface. Queue managers are the main components in an IBM MQ messaging network.

Before you begin

You must have IBM MQ installed. If you do not, see “Installing using a silent installation” on page 10 for information about how to do so.

About this task

In this example, all names are typed in uppercase and because IBM MQ names are case-sensitive, you must type all names in uppercase too.

Procedure

1. Open a command prompt as an administrator.
2. Create a queue manager with the name QM1 by typing the following command:

```
crtmqm QM1
```

When the system creates the queue manager, the following output is displayed:

```
C:\>crtmqm QM1
IBM MQ queue manager created.
Creating or replacing default objects for QM1.
Default objects statistics : 61 created. 0 replaced. 0 failed.
Completing setup.
Setup completed.
```

The queue manager is created, and is stopped. You must start the queue manager before you can administer it, and before you can read and write messages from its queues.

3. Start the queue manager by entering the following command:

```
strmqm QM1
```

When the queue manager successfully starts, the following output is displayed:

```
C:\>strmqm QM1
IBM MQ queue manager 'QM1' starting.
5 log records accessed on queue manager 'QM1' during the log replay phase.
Log replay for queue manager 'QM1' complete.
Transaction manager state recovered for queue manager 'QM1'.
IBM MQ queue manager 'QM1' started.
```

The queue manager is started.

What to do next

To create a queue, see “Creating a queue called LQ1” on page 13.

Related information:

Creating and managing queue managers on distributed platforms

Creating a queue called LQ1:

Create a queue by using the command-line interface. Queues are data structures that are used to store messages and are IBM MQ queue manager objects.

About this task

There are three ways to create IBM MQ objects:

- Command-line.
- MQ Explorer.
- Using a programmable interface.

In this task you can create IBM MQ objects using the command-line.

The command-line interface has a scripting language called IBM MQ Script Commands (MQSC). The scripting tool, **runmqsc**, is used to run the script against a queue manager. To create and start a queue by using the command-line interface, complete the following steps.

Procedure

1. Start the scripting tool by typing the following command:

```
runmqsc QM1
```

When the scripting tool starts, the following output is displayed:

```
C:\>runmqsc QM1
5724-H72 (C) Copyright IBM Corp. 1994, 2008. ALL RIGHTS RESERVED.
Starting MQSC for queue manager QM1.
```

The tool is ready to accept MQSC commands.

2. Create a local queue called LQ1 by typing the following MQSC command:

```
define qlocal(LQ1)
```

When the queue is created, the following output is displayed:

```
define qlocal(LQ1)
2 : define qlocal(LQ1)
AMQ8006: IBM MQ queue created.
```

3. Stop the scripting tool by typing the following MQSC command:

```
end
```

When the scripting tool ends, the following output is displayed:

```
One MQSC command read.
No commands have a syntax error.
All valid MQSC commands were processed.
```

```
C:\>
```

What to do next

You are ready to put a message on to your queue. To put a message in a queue, see “Putting a message to the queue LQ1” on page 14.

Putting a message to the queue LQ1:

Put a message on to the queue LQ1 by using the command-line interface.

About this task

IBM MQ comes with a sample application called **amqsput** . This application puts a message to a predefined queue.

To put a message to the queue by using the command-line interface, complete the following steps.

Procedure

1. Use the **amqsput** sample application to put a message to queue LQ1, by typing the following command:

```
amqsput LQ1 QM1
```

When the sample application starts, the following output is displayed:

```
C:\>amqsput LQ1 QM1
Sample AMQSPUT0 start
target queue is LQ1
```

2. Type Hello World and press Enter. You placed a message that contains the text "Hello World" on the queue LQ1 managed by the queue manager called QM1.
3. To end **amqsput** , press Enter. The following output is displayed:

```
C:\>amqsput LQ1 QM1
Sample AMQSPUT0 start
target queue is LQ1
Hello World
```

```
Sample AMQSPUT0 end
```

What to do next

To get a message from the queue, see "Getting a message from the queue LQ1."

Getting a message from the queue LQ1:

Get a message from the queue LQ1 by using the command-line interface.

About this task

IBM MQ comes with a sample application called **amqsget** . This application reads messages from a queue.

To get a message from the queue by using the command-line interface, complete the following steps.

Procedure

Use the **amqsget** sample application to read a message on the queue LQ1, by typing the following command:

```
amqsget LQ1 QM1
```

When the sample application starts, the following output is displayed:

```
C:\>amqsget LQ1 QM1
Sample AMQSGET0 start
message <Hello World>
no more messages
Sample AMQSGET0 end
```

The **amqsget** application ends 30 seconds after reading the message.

What to do next

Follow the instructions in subsequent scenarios to explore further IBM MQ features.

To learn about writing queuing applications, connecting to and disconnecting from a queue manager, publish/subscribe, and opening and closing objects, see Writing a procedural application for queuing.

Uninstalling IBM MQ

Stop, and then uninstall IBM MQ, including removing any queue managers and their objects. At the end of this task, you are ready to reinstall IBM MQ.

About this task

This task describes the steps for uninstalling IBM MQ on the Windows 7 operating system by using the installation media.

The Getting started scenario takes you through options for installing IBM MQ by using the launchpad or command line. Although you can have more than one installation of IBM MQ, this scenario is based on a new installation on a single server. Therefore, if you want to repeat the scenario, or try out a different installation method, you must first uninstall the existing IBM MQ components, including any existing queue managers and their objects, so that you can start again with a fresh installation.

You might also need to uninstall so that you can carry out a fresh installation for some of the other scenarios in this section.

Procedure

1. Stop the IBM MQ service.
 - a. Right-click on the **Websphere MQ** icon in the system tray, then click **Stop Websphere MQ** to stop the IBM MQ service. A dialog with the following message is displayed:
Shutting down WebSphere MQ installation "Installation1" terminates all running queue managers and WebSphere MQ processes for that installation, except those under Microsoft Failover Cluster control. Are you sure you want to continue?
 - b. Click **Yes** and then wait for the IBM MQ to stop.
 - c. When the IBM MQ stops, right-click the **Websphere MQ** icon in the system tray, then click **Exit**
2. Begin the uninstalling process in one of the two following ways:
 - a. In Windows Explorer, navigate to the temporary folder with the installation image and double click setup.exe.
 - b. Insert the IBM MQ for Windows Server DVD into the DVD drive. If autorun is enabled, the installation process starts. Otherwise, double-click the Setup icon in the root folder of the DVD to start the uninstalling process. The IBM MQ Installation Launchpad window opens.
3. Remove IBM MQ.
 - a. Click **IBM MQ Installation**.
 - b. Click **Launch IBM MQ Installer** and click **Next** until the IBM MQ Program Maintenance pane is displayed with a welcome message. If this pane is not displayed, IBM MQ for Windows is not currently installed.

- c. Click **Maintain or upgrade an existing instance**. Select **Installation1** to remove it. Click **Next** and in the Program Maintenance pane, click **Remove**, then **Next**. The Removing Server feature pane is shown.
- d. Select **Remove**: remove existing queue managers and their objects. Click **Next**. The Remove IBM MQ pane is displayed, with a summary of the installation to be removed.
- e. Click **Remove** to continue. If a message appears stating that locked files are found, ensure that no IBM MQ programs are running; see Uninstalling IBM MQ on Windows systems. When IBM MQ is uninstalled, a message indicates completion.
- f. Click **Finish**.

You have successfully uninstalled IBM MQ.

Related information:

Uninstalling IBM MQ on Windows systems

What to do next

What to do next on completion of the Getting started with IBM MQ scenario.

There are additional topics for you to view in the IBM MQ product documentation. You might want to look at the following sections:

- Administering IBM MQ

IBM MQ provides control commands that you can use. You use two of these commands in this scenario: **crtmqm** and **strmqm**. This section also provides a good overview about message queuing.

- MQSC reference

In this scenario, you use the `define qlocal('LQ1')` command to define a local queue called LQ1 ; this command is an MQSC command. IBM MQ System Administrators use these commands to manage their queue managers. This section introduces the commands and shows you how to use them, before describing the commands in detail, in alphabetical order.

- Configuring a queue manager cluster

This section describes how to organize, use, and manage queue managers in virtual groups known as clusters. Clustering ensures that each queue manager within a cluster knows about all the other queue managers in the same cluster. Clustering also makes the management of complex queue manager networks simpler.

The Product Connectivity Scenarios and Patterns product documentation provides information that leads you through the key tasks required to connect WebSphere Application Server to IBM MQ in a variety of scenarios. Each scenario contains the instructions for implementing a solution in a business context, allowing you to learn as you go without needing to make use of other information resources.

Other learning resources

IBM MQ provides role-based training paths to assist you by defining a path to acquiring skills for specific WebSphere product offerings.

There are two training paths for IBM MQ:

- Application Developer

These users are responsible for creating the applications that use the queue manager. In this scenario, they write the applications **amqsput** and **amqsget**.

- System Administrator

These users are responsible for creating the queue manager and its objects, they typically carry out similar tasks to that you covered in this scenario.

For more information about IBM MQ training paths, see: <http://www.ibm.com/software/websphere/education/paths/>.

A certification program is available which demonstrates you achieve a skill level in IBM MQ. For more information, see: http://www.ibm.com/certify/certs/ws_index.shtml.

Conferences are available for you to attend, for a list, see: <http://www-304.ibm.com/jct03001c/services/learning/ites.wss?pagetype=page&c=a0015064>.

You can collaborate with other users, for example, for:

- E-mail based community of IBM MQ professionals, see <https://listserv.meduniwien.ac.at/archives/mqser-l.html>.
- Discussion forums focusing on the IBM MQ family of products, see <http://www.mqseries.net/>.
- A developerWorks® blog by the developers of the various IBM messaging products, see <http://www.ibm.com/developerworks/community/blogs/page/messaging/>.
- IBM MQ tagged questions and answers on [stackoverflow.com](http://stackoverflow.com/questions/tagged/websphere-mq), see <http://stackoverflow.com/questions/tagged/websphere-mq>.

Related information:

IBM MQ Version 8.0 PDF documentation

Point-to-point scenario

Connect two IBM MQ queue managers in a point-to-point topology to enable distributed queuing.

About this task

Create two queue managers and the appropriate queues and channels to create a one-way, point-to-point messaging infrastructure. Create the queue managers on separate hosts to enable communication over a network. As an extension to the scenario, add Transport Layer Security to the channel to enable secure communication of data.

Planning the solution

Point-to-point messaging is the simplest form of messaging in IBM MQ. In point-to-point messaging, the sending application must know certain information about the receiving application before messages can be sent. The sending application will require a way to address the remote queue. Use point-to-point messaging to send a message to a remote queue manager with a sample application.

Overview: The delivered logical topology

The delivered logical topology after completing the scenario.

The point-to-point infrastructure allows one directional messaging between queue managers on different host machines. Queue manager one, on host one sends messages to queue manager two, on host two. After this scenario is complete, the delivered topology will look like Figure 1.

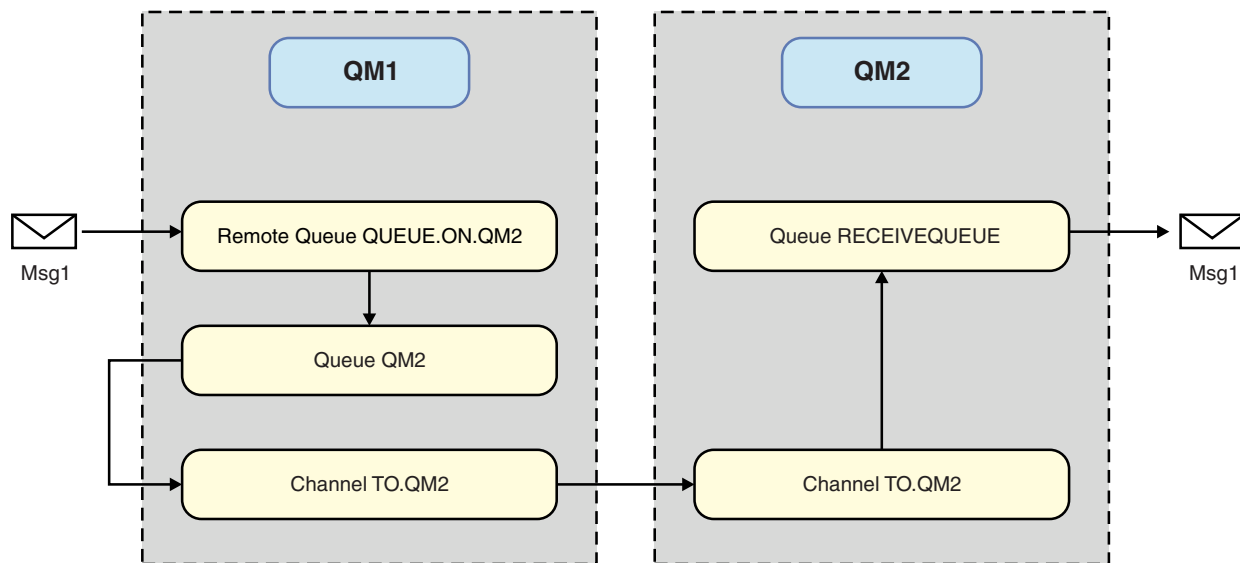


Figure 2. QM1 sends a message to QM2

Basic concepts and key terms

Descriptions of the basic concepts and key terms you must know to complete the point to point scenario.

Basic concepts

IBM MQ enables applications to read and write messages to a queue. The application that reads the message is independent of the application that writes the message. It is not a requirement to have the two applications running at the same time. If no application is available to read the message it is queued on the IBM MQ queue until an application reads it.

Key terms

Here is a list of key terms about message queuing.

Term	Description
Queue managers	The queue manager is responsible for maintaining the queues it owns, and for storing all the messages it receives onto the appropriate queues.

Term	Description
Messages	A message is a string of bytes that is meaningful to the applications that use it. Messages are used to transfer information from one application program to another. The applications can be running on the same or on different computers.
Local queues	A local queue is a data structure used to store messages. The queue can be a normal queue or a transmission queue. A normal queue holds messages that are to be read by an application that is reading the message directly from the queue manager. A transmission queue holds messages that are in transit to another queue manager.
Remote queues	A remote queue is used to address a message to another queue manager.
Channels	Channels are use to send and receive messages between queue managers.
Listeners	Listeners are processes that accept network requests from other queue managers, or client applications, and start associated channels.

Implementing the solution

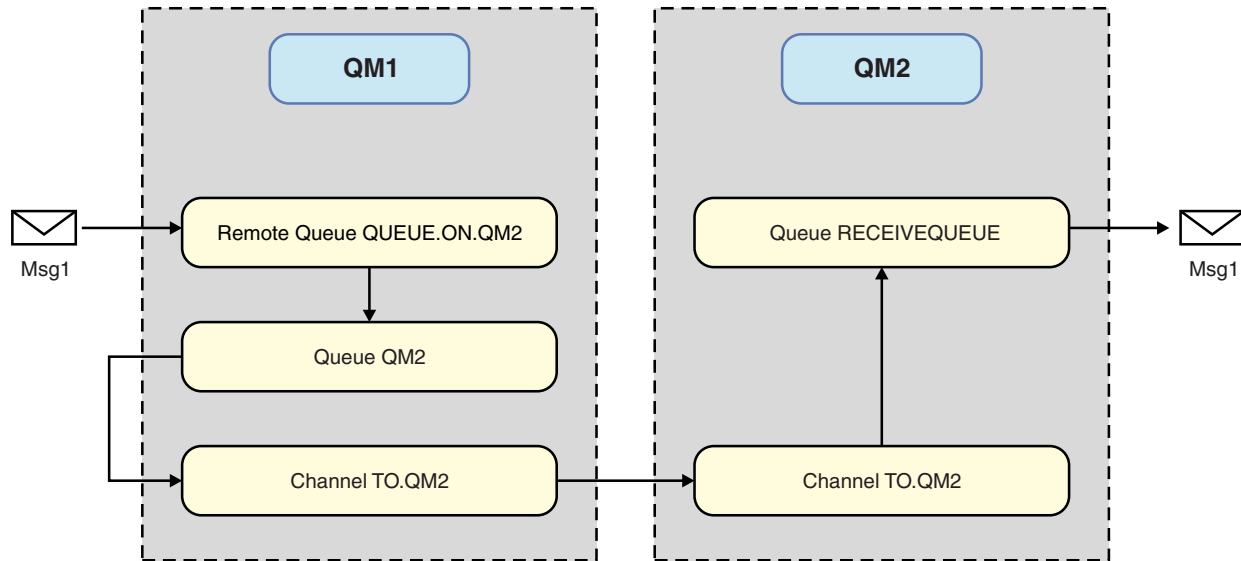
Implement the solution to the scenario. Create two IBM MQ queue managers on two separate hosts, the source queue manager to send messages, and the target queue manager to receive messages.

Before you begin

The starting point for this scenario is an existing, verified IBM MQ installation. For instructions to install IBM MQ, follow the steps in *Installing an IBM MQ server*.

About this task

Create two queue managers by using the command-line interface, define the required listeners, queues, and channels. The delivered logical topology shows the functions added by implementing the solution.



Creating the queue manager

Create an IBM MQ queue manager to send messages to the target queue manager.

Before you begin

- You must have IBM MQ installed. For more information about installing IBM MQ, see [Installing and uninstalling](#).

About this task

Create the IBM MQ queue manager by using the command-line interface.

Procedure

- Create a queue manager with the name QM1. On the command-line, type:
`crtmqm QM1`

The following messages are displayed to confirm that the queue manager is created:

```

IBM MQ queue manager created.
Creating or replacing default objects for QM1.
Default objects statistics : 61 created. 0 replaced. 0 failed.
Completing setup.
Setup completed.
  
```

- Start the queue manager. On the command-line, type:
`strmqm QM1`

The following messages are displayed to confirm that the queue manager is started:

```

IBM MQ queue manager 'QM1' starting.
5 log records accessed on queue manager 'QM1' during the log replay phase.
Log replay for queue manager 'QM1' complete.
Transaction manager state recovered for queue manager 'QM1'.
IBM MQ queue manager 'QM1' started.
  
```

Results

The IBM MQ queue manager QM1 is created and started.

What to do next

To create the queues to use with QM1, follow the instructions in “Creating the queues.”

Creating the queues

Create IBM MQ queues that are managed by the IBM MQ queue manager.

Before you begin

You must have an IBM MQ queue manager that is set up as described in “Creating the queue manager” on page 20.

About this task

Start the **MQSC** interface to administer objects that are connected to the queue manager. Create a transmission queue, and a remote queue definition. Exit the **MQSC** interface.

Procedure

1. On the command-line, type:

```
runmqsc QM1
```

After a confirmation message, the tool is ready to accept commands.

2. Create a transmission queue called QM2. It is good practice to give the transmission queue the same name as the remote queue manager. In the MQSC interface, type:

```
DEFINE QLOCAL(QM2) DESCR('Transmission queue to QM2') USAGE(XMITQ)
```

The transmission queue is created.

3. Create a remote queue definition called QUEUE.ON.QM2. The remote queue definition must refer to the name given to the local queue on the remote host. In the MQSC interface, type:

```
DEFINE QREMOTE(QUEUE.ON.QM2) DESCR('Remote queue for QM2') XMITQ(QM2) RNAME(RECEIVEQUEUE) RQMNAME(QM2)
```

The remote queue definition is created.

4. Type end to exit the MQSC interface.

What to do next

To create the sender channel that is used to connect to the target queue manager, follow the instructions in “Creating the sender channel” on page 22.

Creating the sender channel

Create the sender channel on the source queue manager, the channel is used to connect to the target queue manager.

Before you begin

To create a channel that uses TLS, follow the instructions in “Creating the channels to use TLS” on page 28. This can be done afterward if you want to test the solution without TLS security.

About this task

Start the **MQSC** interface to administer objects that are connected to the queue manager and create the sender channel. This channel is used to connect to the target queue manager called QM2.

Procedure

1. On the command-line, type:

```
runmqsc QM1
```

After a confirmation message, the tool is ready to accept commands.

2. Create the sender channel, called T0.QM2. In the MQSC interface, type:

```
DEFINE CHANNEL(T0.QM2) CHLTYPE(SDR) CONNAME(' remoteHost ') TRPTYPE(TCP) XMITQ(QM2)
```

Note: The variable *remoteHost* is the hostname or IP address of the target queue manager. The sender channel is created.

What to do next

To create the distributed queue manager topology, follow the instructions in “Creating the distributed queue manager topology.”

Creating the distributed queue manager topology

Point-to-point messaging is the simplest form of messaging in IBM MQ. In point-to-point messaging, the sending application must know certain information about the receiving application before messages can be sent. The sending application will require a way to address the remote queue. Use point-to-point messaging to send a message to a second queue manager with a sample application.

Before you begin

You must have set up the source queue manager as described in “Creating the queue manager” on page 20.

About this task

Create the target queue manager on a remote host. Use the sample applications to verify communication between the source and target queue managers.

Creating the queue manager:

Create an IBM MQ queue manager to receive messages from the remote queue manager.

Before you begin

You must have IBM MQ installed. For more information about installing IBM MQ, see [Installing an IBM MQ server](#).

About this task

Create the IBM MQ queue manager by using the command-line interface.

Procedure

1. Create a queue manager with the name QM2. On the command-line, type:

```
crtmqm QM2
```

The following messages are displayed:

```
IBM MQ queue manager created.  
Creating or replacing default objects for QM2.  
Default objects statistics : 61 created. 0 replaced. 0 failed.  
Completing setup.  
Setup completed.
```

2. Start the queue manager. On the command-line, type:

```
strmqm QM2
```

The following messages are displayed to confirm that the queue manager is started:

```
IBM MQ queue manager 'QM2' starting.  
5 log records accessed on queue manager 'QM2' during the log replay phase.  
Log replay for queue manager 'QM2' complete.  
Transaction manager state recovered for queue manager 'QM2'.  
IBM MQ queue manager 'QM2' started.
```

Results

The IBM MQ queue manager QM2 is created and started.

What to do next

To create the queue to use with QM2, follow the instructions in [“Creating the queue.”](#)

Creating the queue:

Create the local queue that is used to receive messages on the target queue manager, and the listener that accepts the inbound channel connection.

About this task

After you have started the **runmqsc** scripting tool, you can use MQSC commands to create a local queue and listener.

Procedure

1. Start the scripting tool by typing the following command:

```
runmqsc QM2
```

A message is displayed to confirm that the tool has started.

2. Create a local queue called RECEIVEQUEUE. The queue must have the same name as referred to in the remote queue definition on the source queue manager. In the MQSC interface, type:

```
DEFINE QLOCAL(RECEIVEQUEUE) DESCR('Receiving queue')
```

The local queue is created.

3. Create a listener called LISTENER1. In the MQSC interface, type:

```
DEFINE LISTENER(LISTENER1) TRPTYPE(TCP) PORT(1414) CONTROL(QMGR)
```

Note: Port 1414 is the default port for IBM MQ. If you chose a different port number, you must add it to the CONNAME of the sender channel on the sending queue manager.

4. Start the listener so it is ready to accept inbound connections. In the MQSC interface, type:

```
START LISTENER(LISTENER1)
```

Note: Since the listener was created with the option CONTROL(QMGR), next time the queue manager is started, the listener will also be automatically started.

5. Type end to exit the **MQSC** interface.

What to do next

To create the receiver channel to create the connection between the source and target queue managers, follow the instructions in “Creating the receiver channel.”

Creating the receiver channel:

Create the receiver channel for the target queue manager to enable communication between the source and target queue managers.

Before you begin

To create a channel that uses TLS, follow the instructions in “Creating the channels to use TLS” on page 28. This can be done afterward if you want to test the solution without TLS security.

About this task

Use the **MQSC** interface to create a receiver channel that is managed by QM2.

Procedure

1. On the command-line, type:

```
runmqsc QM2
```

After a confirmation message, the tool is ready to accept commands.

2. Create a receiver channel called TO.QM2. The channel must have the same name as the sender channel on the source queue manager. In the MQSC interface, type:

```
DEFINE CHANNEL(TO.QM2) CHLTYPE(RCVR) TRPTYPE(TCP)
```

The receiver channel is created.

What to do next

To start the sender channel on the source queue manager, that in turn initiates the receiver channel on the target queue manager, follow the instructions in “Starting the sender channel” on page 25.

Starting the sender channel:

Start the sender channel on the source queue manager, the receiver channel on the target queue manager is also started. Messages can be sent from the source queue manager to the target queue manager.

About this task

Start the **MQSC** interface to administer objects that are connected to the queue manager. Start the sender channel to connect to the target queue manager, enabling communication. The receiver channel starts automatically when the source channel is started.

Procedure

1. On the command-line, type:

```
runmqsc QM1
```

After a confirmation message, the tool is ready to accept commands.

2. Start the sender channel on the source queue manager. In the MQSC interface, type:

```
START CHANNEL(TO.QM2)
```

The sender channel starts, the receiver channel on the target queue manager is also started.

3. Check that the channel is running. In the MQSC interface, type:

```
DISPLAY CHSTATUS(TO.QM2)
```

If the channel is running, you will see that it reports **STATUS(RUNNING)**. If it reports any other value in **STATUS** then check the error log.

What to do next

To verify that the source queue manager can send messages to the target queue manager, follow the instructions in “Verifying the solution.”

Verifying the solution

Verify that the source queue manager can put a message onto the remote queue. Verify that the target queue manager can get the message from the queue.

About this task

Use the sample applications, **amqsput** and **amqsget** to verify the solution.

Procedure

1. Send a message to the target queue manager, QM2 from the source queue manager.

- a. In the command-line interface, type:

```
amqsput QUEUE.ON.QM2 QM1
```

You must use the name of the remote queue definition to send the message to the target queue manager. The following message is displayed:

```
Sample AMQSPUT0 start  
target queue is QUEUE.ON.QM2
```

- b. Type **Hello world.**, press **Enter** twice.
2. Get the message on the target queue manager.

- a. In the command-line interface, type:

```
amqsget RECEIVEQUEUE QM2
```

The following message is displayed:

```
Sample AMQSGETO start
message <Hello world.>
no more messages
Sample AMQSGETO end
```

Results

The target queue manager received the message from the source queue manager, verifying that point to point communication is achieved.

What to do next

If you want to add security to the solution, follow the instructions in “Securing the point-to-point topology.”

Securing the point-to-point topology

Secure the point-to-point topology so that messages can be transmitted in a production environment.

About this task

Secure the source and target queue manager objects so that the correct level of access is granted. Define which user groups have access to the queues and queue managers. Secure the network connection by using digitally signed certificates to connect using Transport Layer Security (TLS).

Securing the source queue manager objects

Set the authorization values for the objects on the source queue manager.

About this task

Use the **setmqaut** command to grant authorities to the user group running the application.

Procedure

1. To grant the specified user group with *connect* authorization to the queue manager, on the command-line interface, type:

```
setmqaut -m QM1 -t qmgr -g userGroup +connect
```
2. To grant the specified user group with *put* authorization on the remote queue definition, on the command-line interface, type:

```
setmqaut -m QM1 -t q -n "QUEUE.ON.QM2" -g userGroup +put
```

Securing the target queue manager objects

Set the authorization values for the objects on the target queue manager.

About this task

Use the **setmqaut** command to grant authorities to the user group running the application.

Procedure

1. To grant the specified user group with *connect* authorization to the queue manager, on the command-line interface, type:
2. To grant the specified user group with *get* authorization on the remote queue definition, in the command-line interface, type:

```
setmqaut -m QM2 -t qmgr -g userGroup +connect
```

```
setmqaut -m QM2 -t q -n "RECEIVEQUEUE" -g userGroup +get
```

Securing the network

Secure the network connections between the source and remote queue managers.

About this task

Use signed certificates to verify the authenticity of the source and remote queue managers. Transfer messages using an SSL or TLS network to encrypt messages.

Preparing the queue managers to use TLS:

The IBM MQ queue manager's key repository is used to store the queue manager's personal certificate and the public Certificate Authority (CA) certificate. The personal certificate request from the IBM MQ queue manager must be signed by a CA, the public certificate is used by the other entities to authenticate the IBM MQ queue manager.

Before you begin

You must have the public Certificate Authority certificate in a file.

About this task

Create the IBM MQ queue manager's key repository, import the certificate authority's signer certificate and create the queue manager's personal certificate request.

Procedure

1. Create a CMS key repository file for the queue manager called `key.kdb`. Navigate to the `Qmgrs\QM1\ssl` directory, and on the command line, type:

```
runmqckm -keydb -create -db key.kdb -pw passw0rd -type cms -stash
```

Note: For this simple example we have used a password of `passw0rd`. You may wish to choose a different password and change each of the following commands to use your own password instead.

2. Add the CA certificate, which you have in a file, to the key repository, on the command line, type:

```
runmqckm -cert -add -file CA-certificate-file -db key.kdb -pw passw0rd -label TrustedCA
```

3. Request a personal certificate that will be written to a request file called `QM1req.req`. On the command line, enter:

```
runmqckm -certreq -create -db key.kdb -pw passw0rd -label ibmwebspheremqm1  
-dn CN="QM1" -size 1024 -file QM1req.req  
-sig_alg SHA1WithRSA
```

The default certificate label name is shown in this example. You can set your own name if you prefer. For details, see Digital certificate labels.

4. Send the certificate request file to your CA, they will issue a digitally signed certificate. Put the received, signed certificate file in a suitable location to be received into the queue manager's key repository.
5. Receive the signed personal certificate into the queue manager's key repository.
`runmqckm -cert -receive -file Signed-certificate-file -db key.kdb -pw password -format ascii`
6. Complete these steps for each queue manager, changing the queue manager name accordingly.

What to do next

To enable secure communication over the sender and receiver channels, follow the instructions in “Creating the channels to use TLS.”

Creating the channels to use TLS:

Create a new channel that uses TLS to create a connection.

Before you begin

To communicate over a channel that uses TLS, first you must have the required certificates for each end of the connection. To create the required certificates, follow the instructions in “Preparing the queue managers to use TLS” on page 27.

About this task

Use the MQSC interface to define channels with SSL/TLS attributes set. This task can be done even if you defined your channels without SSL/TLS in a prior step through the use of the REPLACE keyword.

Procedure

1. On the command-line, type:
`runmqsc QM1`
2. Create the sender channel on QM1, called T0.QM2, in the MQSC interface, type:

```
DEFINE CHANNEL(T0.QM2) CHLTYPE(SDR) TRPTYPE(TCP)
CONNNAME('remoteHost') XMITQ(QM2)
SSLCIPH(TLS_RSA_WITH_AES_128_CBC_SHA256)
DESCR('Sender channel using TLS from QM1 to QM2')
REPLACE
```

Note: The variable *remoteHost* is the hostname or IP address of the target queue manager.

You can specify a CERTLABL attribute for the channel. If you do, it must match the value on the **-label** parameter of the **runmqckm** command that you previously ran in step 3 of “Preparing the queue managers to use TLS” on page 27. For more information on certificate labels, see Digital certificate labels, understanding the requirements.

3. Type end to exit the MQSC interface.
4. On the command-line, type:
`runmqsc QM2`
5. Create a receiver channel on QM2, called T0.QM2, in the MQSC interface, type:

```
DEFINE CHANNEL(T0.QM2) CHLTYPE(RCVR) TRPTYPE(TCP)
SSLCIPH(TLS_RSA_WITH_AES_128_CBC_SHA256) SSLCAUTH(REQUIRED)
DESCR('Receiver channel using TLS from QM1 to QM2')
REPLACE
```
6. Type end to exit the MQSC interface.

What to do next

To verify that the source queue manager can send messages to the target queue manager using TLS, follow the instructions in “Verifying the solution” on page 25.

Publish/subscribe scenarios

Two sets of scenarios that demonstrate use of publish/subscribe clusters and publish/subscribe hierarchies.

The available publish/subscribe scenarios are described in the following subtopics:

Publish/subscribe cluster scenario

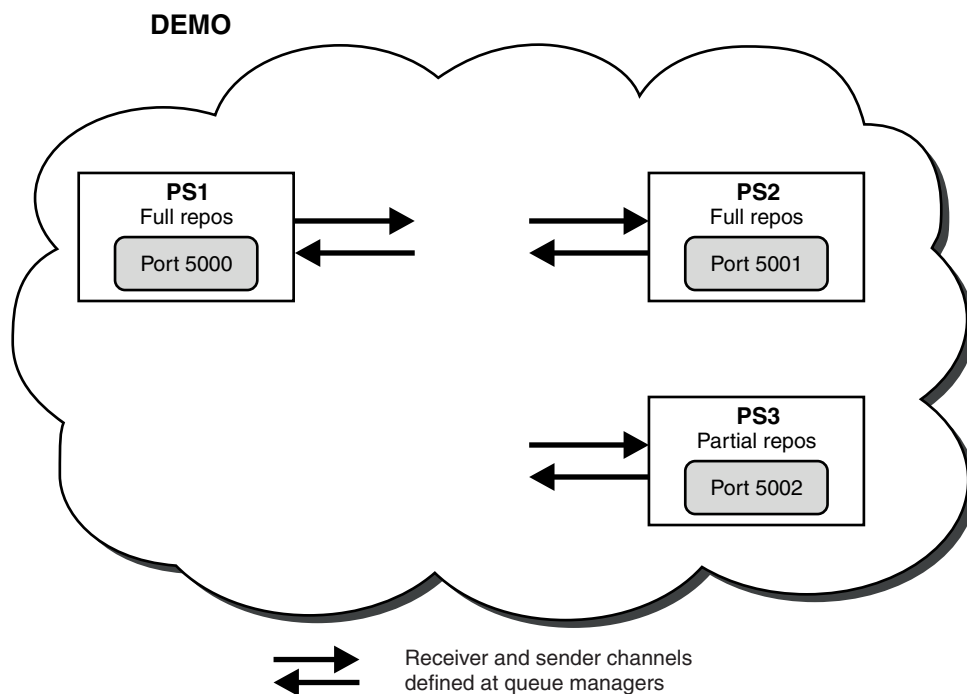
In this scenario you create a simple three queue manager cluster and configure it to allow subscriptions created on one queue manager to receive messages published by an application connected to another queue manager.

Before you begin

The starting point for this scenario is an existing IBM MQ installation. For instructions to install IBM MQ, follow the steps in Installing an IBM MQ server.

About this task

By completing the steps in this scenario, you first create the following cluster:



This cluster consists of three queue managers, two of which are defined as full repository queue managers.

You then define a cluster topic on queue manager PS3. By creating the cluster topic, you have made the cluster into a publish/subscribe cluster. To test the publish/subscribe cluster, you subscribe to the topic on any queue manager, then publish a message to the topic from another queue manager and check that

your subscription receives the message.

Related information:

Designing publish/subscribe clusters

Configuring a queue manager cluster

Creating and starting the queue managers

Create and start three queue managers, called PS1, PS2 and PS3.

Procedure

1. Create and start queue manager PS1.
 - a. Create the queue manager.

In the command line, enter the following command:

```
crtmqm PS1
```
 - b. Start the queue manager.

In the command line, enter the following command:

```
strmqm PS1
```
2. Repeat step 1 to create and start queue manager PS2.
3. Repeat step 1 to create and start queue manager PS3.

What to do next

You are now ready to configure the first queue manager.

Configuring the first queue manager

Use the MQSC interface to define a listener and a receiver channel for PS1, to set the queue manager as a full repository for the cluster, and to define a sender channel from PS1 to PS2 so the two full repositories can exchange information.

Before you begin

This task assumes that you have completed the steps in “Creating and starting the queue managers.”

Procedure

1. Define and start a listener for PS1.
 - a. Launch the MQSC interface.

In the command line, enter the following command:

```
runmqsc PS1
```
 - b. Define a listener.

Enter the following MQSC command:

```
DEFINE LISTENER(PS1_LS) TRPTYPE(TCP) CONTROL(QMGR) PORT(5000)
```
 - c. Start the listener.

Enter the following MQSC command:

```
START LISTENER(PS1_LS)
```
2. Set the queue manager as a full repository for the cluster.

Enter the following MQSC command:

```
ALTER QMGR REPOS(DEMO)
```
3. Define a receiver channel for PS1, to allow other queue managers in the cluster to communicate with it.

Enter the following MQSC command:

```
DEFINE CHANNEL(DEMO.PS1) CHLTYPE(CLUSRCVR) TRPTYPE(TCP) CONNAME('$HOSTNAME(5000)') CLUSTER(DEMO)
DESCR('TCP Cluster-receiver channel for queue manager PS1')
```

4. Define a sender channel from PS1 to PS2, to allow the two full repositories to exchange information.
Enter the following MQSC command:

```
DEFINE CHANNEL(DEMO.PS2) CHLTYPE(CLUSSDR) TRPTYPE(TCP) CONNAME('$HOSTNAME(5001)') CLUSTER(DEMO)
DESCR('TCP Cluster-sender channel from PS1 to queue manager PS2')
```

What to do next

You are now ready to configure the second queue manager.

Configuring the second queue manager

Use the MQSC interface to define a listener and a receiver channel for PS2, to set the queue manager as a full repository for the cluster, and to define a sender channel from PS2 to PS1 so the two full repositories can exchange information.

Before you begin

This task assumes that you have completed the steps in “Configuring the first queue manager” on page 30.

Procedure

1. Define and start a listener for PS2.
 - a. Launch the MQSC interface.
In the command line, enter the following command:

```
runmqsc PS2
```
 - b. Define a listener.
Enter the following MQSC command:

```
DEFINE LISTENER(PS2_LS) TRPTYPE(TCP) CONTROL(QMGR) PORT(5001)
```
 - c. Start the listener.
Enter the following MQSC command:

```
START LISTENER(PS2_LS)
```
2. Set the queue manager as a full repository for the cluster.
Enter the following MQSC command:

```
ALTER QMGR REPOS(DEMO)
```
3. Define a receiver channel for PS2, to allow other queue managers in the cluster to communicate with it.
Enter the following MQSC command:

```
DEFINE CHANNEL(DEMO.PS2) CHLTYPE(CLUSRCVR) TRPTYPE(TCP) CONNAME('$HOSTNAME(5001)') CLUSTER(DEMO)
DESCR('TCP Cluster-receiver channel for queue manager PS2')
```
4. Define a sender channel from PS2 to PS1, to allow the two full repositories to exchange information.
Enter the following MQSC command:

```
DEFINE CHANNEL(DEMO.PS1) CHLTYPE(CLUSSDR) TRPTYPE(TCP) CONNAME('$HOSTNAME(5000)') CLUSTER(DEMO)
DESCR('TCP Cluster-sender channel from PS2 to PS1')
```

What to do next

You are now ready to configure the third queue manager.

Configuring the third queue manager

Use the MQSC interface to define a listener and a receiver channel for PS3. Join PS3 into the cluster by defining a sender channel from PS3 to one of the full repository queue managers.

Before you begin

This task assumes that you have completed the steps in “Configuring the second queue manager” on page 31.

Procedure

1. Define and start a listener for PS3.
 - a. Launch the MQSC interface.

In the command line, enter the following command:

```
runmqsc PS3
```
 - b. Define a listener.

Enter the following MQSC command:

```
DEFINE LISTENER(PS3_LS) TRPTYPE(TCP) CONTROL(QMGR) PORT(5002)
```
 - c. Start the listener.

Enter the following MQSC command:

```
START LISTENER(PS3_LS)
```
2. Define a receiver channel for PS3, to allow other queue managers in the cluster to communicate with it.

Enter the following MQSC command:

```
DEFINE CHANNEL(DEMO.PS3) CHLTYPE(CLUSRCVR) TRPTYPE(TCP) CONNAME('$HOSTNAME(5002)') CLUSTER(DEMO) DESCR('TCP Cluster-receiver channel for queue manager PS3')
```
3. Define a sender channel from PS3 to one of the full repository queue managers (for example, PS1). This joins PS3 into the cluster.

Enter the following MQSC command:

```
DEFINE CHANNEL(DEMO.PS1) CHLTYPE(CLUSSDR) TRPTYPE(TCP) CONNAME('$HOSTNAME(5000)') CLUSTER(DEMO) DESCR('TCP Cluster-sender channel from PS3 to PS1')
```
4. Validate that PS3 has successfully joined the cluster.

Enter the following MQSC command:

```
DISPLAY CLUSQMGR(*) QMTYPE
```

This command returns three entries, one each for QM1, QM2 and QM3. QM1 and QM2 should have a **QMTYPE** of REPOS, and QM3 should have a **QMTYPE** of NORMAL.

What to do next

You are now ready to define a cluster topic.

Defining cluster topics

Publishing and subscribing applications can publish to any topic string, with no need for an administered topic object to be defined. However, if the publishing applications are connected to a cluster queue manager that is different to the queue managers where subscriptions are created, an administered topic object must be defined and added to the cluster. To make a topic a cluster topic, you specify the name of the cluster in its definition.

Before you begin

This task assumes that you have completed the steps in “Configuring the third queue manager” on page 32.

About this task

The administered topic object identifies the point in the topic tree that is clustered through its topic string. Publishing and subscribing applications can use any topic string at or below that point, and their messages are automatically transmitted between queue managers.

When you define a cluster topic, you also choose its routing model. For more information about publication routing in clusters, see *Designing publish/subscribe clusters*.

For this scenario we use the default routing of *DIRECT*. This means that messages are sent direct from a publishing queue manager to the subscribing queue managers.

Procedure

1. Define the cluster topic SCORES on PS3.

To make the topic a cluster topic, specify the name of the cluster, and set the cluster routing (**CLROUTE**) that you want to use for publications and subscriptions for this topic.

- a. Launch the MQSC interface.

In the command line, enter the following command:

```
runmqsc PS3
```

- b. Define the cluster topic SCORES.

Enter the following MQSC command:

```
DEFINE TOPIC(SCORES) TOPICSTR('/Sport/Scores') CLUSTER(DEMO) CLROUTE(DIRECT)
```

- c. Enter end to exit the MQSC interface for PS3.

2. Verify the topic definition on PS1.

- a. Launch the MQSC interface for PS1.

In the command line, enter the following command:

```
runmqsc PS1
```

- b. Display the cluster state for cluster topic SCORES.

Enter the following MQSC command:

```
DISPLAY TCLUSTER(SCORES) CLSTATE
```

The **CLSTATE** for cluster topic SCORES is shown as ACTIVE.

What to do next

For a more detailed exploration of this task, see *Configuring a publish/subscribe cluster*.

You are now ready to verify the solution. See “Testing the publish/subscribe cluster” on page 34.

Testing the publish/subscribe cluster

Test the publish/subscribe cluster by publishing and subscribing to a topic string from different queue managers in the cluster.

Before you begin

This task assumes that you have completed the steps in “Defining cluster topics” on page 33.

About this task

Using the command line, and the `amqspub` and `amqssub` sample applications that are included with IBM MQ, you can publish a topic from one queue manager and subscribe to the topic with the other queue managers. When a message is published to the topic, it is received by the subscribing queue managers.

Procedure

1. In the command line, enter the following command:
`amqspub /Sport/Scores/Football PS1`
2. Concurrently, in separate command lines, enter the following commands:
`amqssub /Sport/Scores/Football PS2`
`amqssub /Sport/Scores/Football PS3`
3. In the first command line, enter a message. The message is displayed in both the subscribing command lines.

Note: The `amqssub` application will time out if a publication is not received for ten seconds.

Results

The publish/subscribe cluster set up is complete.

What to do next

Try defining different topic objects for different branches of the topic tree, and with different routing models.

Publish/subscribe hierarchy scenarios

Three scenarios that demonstrate use of publish/subscribe hierarchies. Each of the three scenarios sets up the same simple publish/subscribe topology. In each scenario, the queue managers rely on a different method for connecting to their neighboring queue managers in the hierarchy.

The available publish/subscribe hierarchy scenarios are described in the following subtopics:

Related information:

Publish/subscribe hierarchies

Publish/subscribe hierarchy scenario 1: Using point-to-point channels with queue manager name alias

This is the first in a set of three scenarios that set up a publish/subscribe hierarchy in different ways to establish the connection between queue managers. This scenario sets up a publish/subscribe hierarchy that uses point-to-point channels with queue manager name alias.

About this task

This set of scenarios all use a parent queue manager called QM1, and two child queue managers called QM2, and QM3.

Scenario 1 is split into smaller sections to make the process easier to follow.

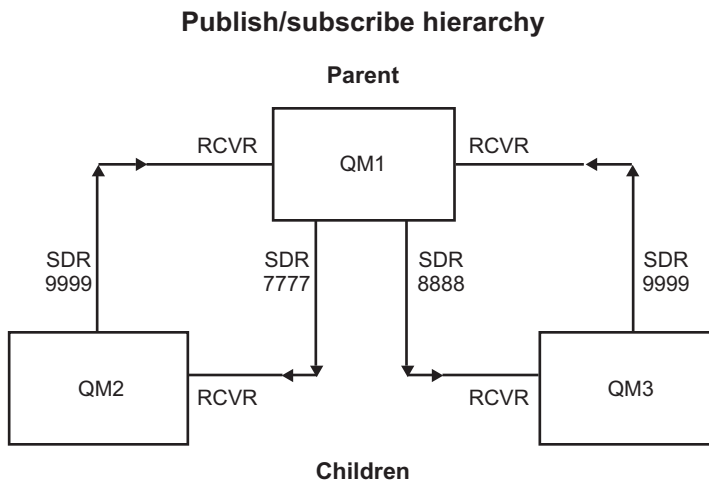


Figure 3. Topology diagram showing the relationship between queue managers in a typical publisher/subscribe hierarchy.

Scenario 1 part 1: Create the queue managers:

Procedure

1. Create and start three queue managers called QM1, QM2, and QM3 using the following commands:

```
crtmqm -u SYSTEM.DEAD.LETTER.QUEUE QM1
strmqm QM1
```

```
crtmqm -u SYSTEM.DEAD.LETTER.QUEUE QM2
strmqm QM2
```

```
crtmqm -u SYSTEM.DEAD.LETTER.QUEUE QM3
strmqm QM3
```

2. Enable the queue manager publish/subscribe mode by using the following command on all three queue managers:

```
ALTER QMGR PSMODE(ENABLED)
```

Scenario 1 part 2: Point-to-point channel connections:

About this task

Establish point-to-point channel connections between queue managers using a queue manager alias with the same name as the parent queue manager.

Procedure

1. Define a transmission queue and queue manager alias on QM2 to QM1. Define a sender channel to QM1 and a receiver channel for the sender channel created on QM1 for QM2:

```
DEFINE QLOCAL(QM1.XMITQ) USAGE(XMITQ)
```

```
DEFINE QREMOTE (QM1) RNAME('') RQMNAME(QM1) XMITQ(QM1.XMITQ)
```

```
DEFINE CHANNEL('QM2.TO.QM1') CHLTYPE(SDR) CONNAME('localhost(9999)') XMITQ(QM1.XMITQ) TRPTYPE(TCP)
```

```
DEFINE CHANNEL('QM1.TO.QM2') CHLTYPE(RCVR) TRPTYPE(TCP)
```

2. Define a transmission queue and queue manager alias on QM3 to QM1. Define sender channel to QM1 and a receiver channel for the sender channel created on QM1 for QM3:

```
DEFINE QLOCAL(QM1.XMITQ) USAGE(XMITQ)
```

```
DEFINE QREMOTE (QM1) RNAME('') RQMNAME(QM1) XMITQ(QM1.XMITQ)
```

```
DEFINE CHANNEL('QM3.TO.QM1') CHLTYPE(SDR) CONNAME('localhost(9999)') XMITQ(QM1.XMITQ) TRPTYPE(TCP)
```

```
DEFINE CHANNEL('QM1.TO.QM3') CHLTYPE(RCVR) TRPTYPE(TCP)
```

3. Define a transmission queue and queue manager alias on QM1 to QM2 and QM3. Define sender channel to QM2 and QM3, and a receiver channel for the sender channels created on QM2 and QM3 for QM1:

```
DEFINE QLOCAL(QM2.XMITQ) USAGE(XMITQ)
```

```
DEFINE QREMOTE (QM2) RNAME('') RQMNAME(QM2) XMITQ(QM2.XMITQ)
```

```
DEFINE CHANNEL('QM1.TO.QM2') CHLTYPE(SDR) CONNAME('localhost(7777)') XMITQ(QM2.XMITQ) TRPTYPE(TCP)
```

```
DEFINE CHANNEL('QM2.TO.QM1') CHLTYPE(RCVR) TRPTYPE(TCP)
```

```
DEFINE QLOCAL(QM3.XMITQ) USAGE(XMITQ)
```

```
DEFINE QREMOTE (QM3) RNAME('') RQMNAME(QM3) XMITQ(QM3.XMITQ)
```

```
DEFINE CHANNEL('QM1.TO.QM3') CHLTYPE(SDR) CONNAME('localhost(8888)') XMITQ(QM3.XMITQ) TRPTYPE(TCP)
```

```
DEFINE CHANNEL('QM3.TO.QM1') CHLTYPE(RCVR) TRPTYPE(TCP)
```

4. Start the appropriate listeners on the queue managers:

```
runmq1sr -m QM1 -t TCP -p 9999 &
runmq1sr -m QM2 -t TCP -p 7777 &
runmq1sr -m QM3 -t TCP -p 8888 &
```

5. Start the following channels:

a. On QM1:

```
START CHANNEL('QM1.TO.QM2')
```

```
START CHANNEL('QM1.TO.QM3')
```

b. On QM2:

```
START CHANNEL('QM2.TO.QM1')
```

c. On QM3:

```
START CHANNEL('QM3.TO.QM1')
```

6. Check that all the channels have started:

```
DISPLAY CHSTATUS('QM1.TO.QM2')
```

```
DISPLAY CHSTATUS('QM1.TO.QM3')
```

```
DISPLAY CHSTATUS('QM2.TO.QM1')
```

```
DISPLAY CHSTATUS('QM3.TO.QM1')
```

Scenario 1 part 3: Connect queue managers and define a topic:

About this task

Connect the child queue managers QM2 and QM3 to the parent queue manager QM1.

Procedure

1. On QM2 and QM3, set the parent queue manager to QM1:

```
ALTER QMGR PARENT (QM1)
```

2. Run the following command on all queue managers to check that the child queue managers are connected to the parent queue manager:

```
DISPLAY PUBSUB TYPE(ALL)
```

Command output is displayed. For example, here is output for QM1, with the key details highlighted:

```
DISPLAY PUBSUB ALL
1 : DISPLAY PUBSUB ALL
AMQ8723: Display pub/sub status details.
QMNAME(QM1)          TYPE(LOCAL)
STATUS(ACTIVE)       SUBCOUNT(6)
TPCOUNT(9)
AMQ8723: Display pub/sub status details.
QMNAME(QM2) TYPE(CHILD)
STATUS(ACTIVE) SUBCOUNT(NONE)
TPCOUNT(NONE)
AMQ8723: Display pub/sub status details.
QMNAME(QM3) TYPE(CHILD)
STATUS(ACTIVE) SUBCOUNT(NONE)
TPCOUNT(NONE)
```

Scenario 1 part 4: Publish and subscribe the topic:

About this task

Use the amqspub.exe and amqssub.exe applications to publish and subscribe the topic.

Procedure

1. Run this command in the first command window:

```
amqspub Sport/Soccer QM2
```

2. Run this command in the second command window:
amqssub Sport/Soccer QM1
3. Run this command in the third command window:
amqssub Sport/Soccer QM3

Results

The amqssub.exe applications in the second and third command windows receive the messages published in the first command window.

Related tasks:

“Publish/subscribe hierarchy scenario 2: Using point-to-point channels with same name for transmission queue and remote queue manager”

This is the second in a set of three scenarios that set up a publish/subscribe hierarchy in different ways to establish the connection between queue managers. This scenario sets up a publish/subscribe hierarchy that uses point-to-point channels with the transmission queue name the same as the remote queue manager.

“Publish/subscribe hierarchy scenario 3: Using a cluster channel to add a queue manager” on page 41
This is the third in a set of three scenarios that set up a publish/subscribe hierarchy in different ways to establish the connection between queue managers. This scenario uses a cluster channel to add a queue manager to a hierarchy.

Related information:

Connecting a queue manager to a publish/subscribe hierarchy

Publish/subscribe hierarchy scenario 2: Using point-to-point channels with same name for transmission queue and remote queue manager

This is the second in a set of three scenarios that set up a publish/subscribe hierarchy in different ways to establish the connection between queue managers. This scenario sets up a publish/subscribe hierarchy that uses point-to-point channels with the transmission queue name the same as the remote queue manager.

About this task

This set of scenarios all use a parent queue manager called QM1, and two child queue managers called QM2, and QM3.

Scenario 2 is split into smaller sections to make the process easier to follow. This scenario reuses Scenario 1 part 1, Scenario 1 part 3, and Scenario 1 part 4 from “Publish/subscribe hierarchy scenario 1: Using point-to-point channels with queue manager name alias” on page 35.

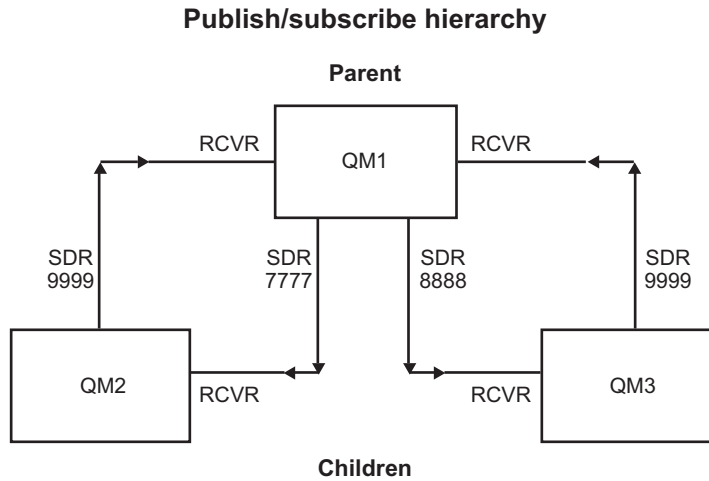


Figure 4. Topology diagram showing the relationship between queue managers in a typical publisher/subscribe hierarchy.

Scenario 2 part 1: Create queue manager and set PSMODE:

Procedure

1. Create and start three queue managers called QM1, QM2, and QM3 using the following commands:

```
crtmqm -u SYSTEM.DEAD.LETTER.QUEUE QM1
strmqm QM1
```

```
crtmqm -u SYSTEM.DEAD.LETTER.QUEUE QM2
strmqm QM2
```

```
crtmqm -u SYSTEM.DEAD.LETTER.QUEUE QM3
strmqm QM3
```

2. Enable the queue manager publish/subscribe mode by using the following command on all three queue managers:

```
ALTER QMGR PSMODE(ENABLED)
```

Scenario 2 part 2: Point-to-point channel connections:

About this task

Establish point-to-point channel connections between a queue manager using a transmission queue with the same name as the parent queue manager.

Procedure

1. Define a transmission queue on QM2 to QM1. Define a sender channel to QM1 and a receiver channel for the sender channel for QM2 created on QM1:

```
DEFINE QLOCAL(QM1) USAGE(XMITQ)
```

```
DEFINE CHANNEL('QM2.TO.QM1') CHLTYPE(SDR) CONNAME('localhost(9999)') XMITQ(QM1) TRPTYPE(TCP)
```

```
DEFINE CHANNEL('QM1.TO.QM2') CHLTYPE(RCVR) TRPTYPE(TCP)
```

2. Define a transmission queue on QM3 to QM1. Define sender channel to QM1 and a receiver channel for the sender channel created on QM1 for QM3:

- ```

DEFINE QLOCAL(QM1) USAGE(XMITQ)

DEFINE CHANNEL('QM3.TO.QM1') CHLTYPE(SDR) CONNAME('localhost(9999)') XMITQ(QM1) TRPTYPE(TCP)

DEFINE CHANNEL('QM1.TO.QM3') CHLTYPE(RCVR) TRPTYPE(TCP)

```
3. Define transmission queues on QM1 to QM2 and QM3. Define sender channels to QM2 and QM3, and a receiver channel for the sender channels created on QM2 and QM3 for QM1:

```

DEFINE QLOCAL(QM2) USAGE(XMITQ)

DEFINE CHANNEL('QM1.TO.QM2') CHLTYPE(SDR) CONNAME('localhost(7777)') XMITQ(QM2) TRPTYPE(TCP)

DEFINE CHANNEL('QM2.TO.QM1') CHLTYPE(RCVR) TRPTYPE(TCP)

DEFINE QLOCAL(QM3) USAGE(XMITQ)

DEFINE CHANNEL('QM1.TO.QM3') CHLTYPE(SDR) CONNAME('localhost(8888)') XMITQ(QM3) TRPTYPE(TCP)

DEFINE CHANNEL('QM3.TO.QM1') CHLTYPE(RCVR) TRPTYPE(TCP)

```
  4. Start the appropriate listeners on the queue managers:

```

runmqlsr -m QM1 -t TCP -p 9999 &
runmqlsr -m QM2 -t TCP -p 7777 &
runmqlsr -m QM3 -t TCP -p 8888 &

```
  5. Start the following channels:
    - a. On QM1:

```

START CHANNEL('QM1.TO.QM2')

START CHANNEL('QM1.TO.QM3')

```
    - b. On QM2:

```

START CHANNEL('QM2.TO.QM1')

```
    - c. On QM3:

```

START CHANNEL('QM3.TO.QM1')

```
  6. Check that all the channels have started:

```

DISPLAY CHSTATUS('QM1.TO.QM2')

DISPLAY CHSTATUS('QM1.TO.QM3')

DISPLAY CHSTATUS('QM2.TO.QM1')

DISPLAY CHSTATUS('QM3.TO.QM1')

```

### Scenario 2 part 3: Connect queue managers and define a topic: About this task

Connect the child queue managers QM2 and QM3 to the parent queue manager QM1.

#### Procedure

1. On QM2 and QM3, set the parent queue manager to QM1:

```

ALTER QMGR PARENT (QM1)

```
2. Run the following command on all queue managers to check that the child queue managers are connected to the parent queue manager:

```

DISPLAY PUBSUB TYPE(ALL)

```

Command output is displayed. For example, here is output for QM1, with the key details highlighted:

```

DISPLAY PUBSUB ALL
1 : DISPLAY PUBSUB ALL
AMQ8723: Display pub/sub status details.
QMNAME(QM1) TYPE(LOCAL)
STATUS(ACTIVE) SUBCOUNT(6)

```



```
TPCOUNT(9)
AMQ8723: Display pub/sub status details.
QMNAME(QM2) TYPE(CHILD)
STATUS(ACTIVE) SUBCOUNT(NONE)
TPCOUNT(NONE)
AMQ8723: Display pub/sub status details.
QMNAME(QM3) TYPE(CHILD)
STATUS(ACTIVE) SUBCOUNT(NONE)
TPCOUNT(NONE)
```

## Scenario 2 part 4: Publish and subscribe the topic:

### About this task

Use the amqspub.exe and amqssub.exe applications to publish and subscribe the topic.

### Procedure

1. Run this command in the first command window:  
amqspub Sport/Soccer QM2
2. Run this command in the second command window:  
amqssub Sport/Soccer QM1
3. Run this command in the third command window:  
amqssub Sport/Soccer QM3

### Results

The amqssub.exe applications in the second and third command windows receive the messages published in the first command window.

### Related tasks:

“Publish/subscribe hierarchy scenario 1: Using point-to-point channels with queue manager name alias” on page 35

This is the first in a set of three scenarios that set up a publish/subscribe hierarchy in different ways to establish the connection between queue managers. This scenario sets up a publish/subscribe hierarchy that uses point-to-point channels with queue manager name alias.

“Publish/subscribe hierarchy scenario 3: Using a cluster channel to add a queue manager”

This is the third in a set of three scenarios that set up a publish/subscribe hierarchy in different ways to establish the connection between queue managers. This scenario uses a cluster channel to add a queue manager to a hierarchy.

### Related information:

Connecting a queue manager to a publish/subscribe hierarchy

## Publish/subscribe hierarchy scenario 3: Using a cluster channel to add a queue manager

This is the third in a set of three scenarios that set up a publish/subscribe hierarchy in different ways to establish the connection between queue managers. This scenario uses a cluster channel to add a queue manager to a hierarchy.

### About this task

This set of scenarios all use a parent queue manager called QM1, and two child queue managers called QM2, and QM3.

**Note:** This scenario is only using the cluster configuration to connect queue managers together, not to propagate publish/subscribe traffic through clustering topics. When defining child/parent hierarchy relationships between queue managers in the same cluster, propagation of publications between queue managers will occur based on the publication and subscription scope settings of the topics in the topic

tree. It is important not to use the cluster name setting of a topic to add the topics into the cluster. If using the cluster name, the topology becomes a publish/subscribe cluster and does not require the child/parent hierarchy relationships defined. See “Publish/subscribe cluster scenario” on page 29 and Planning your distributed publish/subscribe network.

Scenario 3 is split into smaller sections to make the process easier to follow. This scenario reuses Scenario 1 part 1, Scenario 1 part 3, and Scenario 1 part 4 from “Publish/subscribe hierarchy scenario 1: Using point-to-point channels with queue manager name alias” on page 35.

This scenario creates a cluster called DEMO where QM1 and QM2 are full repositories, and QM3 is a partial repository. Queue manager QM1 is the parent of queue managers QM2 and QM3.

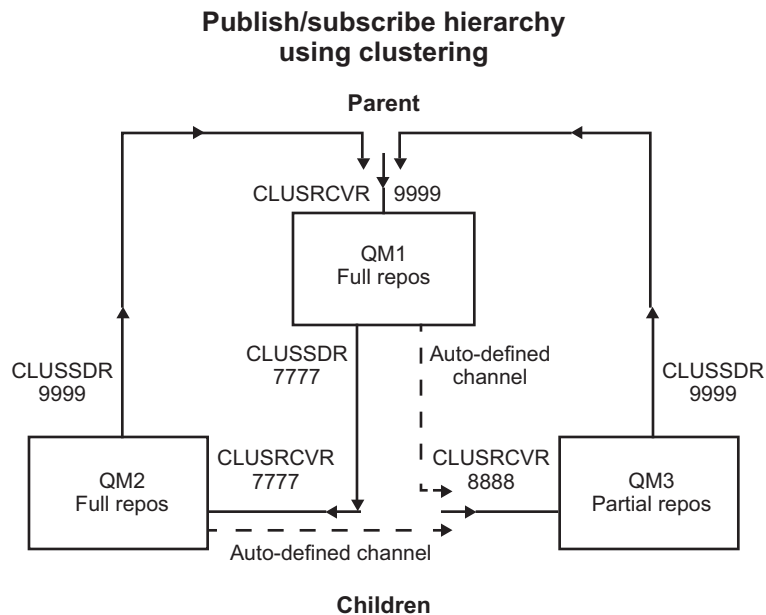


Figure 5. Topology diagram showing the relationship between queue managers that are using a cluster channel.

### Scenario 3 part 1: Create queue manager and set PSMODE: Procedure

1. Create and start three queue managers called QM1, QM2, and QM3 using the following commands:

```
crtmqm -u SYSTEM.DEAD.LETTER.QUEUE QM1
strmqm QM1
```

```
crtmqm -u SYSTEM.DEAD.LETTER.QUEUE QM2
strmqm QM2
```

```
crtmqm -u SYSTEM.DEAD.LETTER.QUEUE QM3
strmqm QM3
```

2. Enable the queue manager publish/subscribe mode by using the following command on all three queue managers:

```
ALTER QMGR PSMODE(ENABLED)
```

### Scenario 3 part 2: Point-to-point channel connections: About this task

Establish point-to-point channel connections between queue managers a cluster.

## Procedure

1. On QM1 and QM2, set the **REPOS** parameter to the name of the cluster DEMO:  
`ALTER QMGR REPOS(DEMO)`
2. Start the appropriate listeners on the queue managers:  
`runmq1sr -m QM1 -t TCP -p 9999 &`  
`runmq1sr -m QM2 -t TCP -p 7777 &`  
`runmq1sr -m QM3 -t TCP -p 8888 &`
3. Define the cluster receiver channel on each queue manager:
  - a. On QM1:  
`DEFINE CHANNEL(TO.QM1) CHLTYPE(CLUSRCVR) TRPTYPE(TCP) CONNAME('localhost(9999)') CLUSTER(DEMO)`
  - b. On QM2:  
`DEFINE CHANNEL(TO.QM2) CHLTYPE(CLUSRCVR) TRPTYPE(TCP) CONNAME('localhost(7777)') CLUSTER(DEMO)`
  - c. On QM3:  
`DEFINE CHANNEL(TO.QM3) CHLTYPE(CLUSRCVR) TRPTYPE(TCP) CONNAME('localhost(8888)') CLUSTER(DEMO)`
4. Define a cluster sender channel to a full repository on each queue manager in the cluster:
  - a. On QM1:  
`DEFINE CHANNEL(TO.QM2) CHLTYPE(CLUSSDR) TRPTYPE(TCP) CONNAME('localhost(7777)') CLUSTER(DEMO)`
  - b. On QM2:  
`DEFINE CHANNEL(TO.QM1) CHLTYPE(CLUSSDR) TRPTYPE(TCP) CONNAME('localhost(9999)') CLUSTER(DEMO)`
  - c. QM3 can have a cluster sender channel to either full repository on QM1 or QM2. This example defines the channel to QM1:  
`DEFINE CHANNEL(TO.QM1) CHLTYPE(CLUSSDR) TRPTYPE(TCP) CONNAME('localhost(9999)') CLUSTER(DEMO)`

## Scenario 3 part 3: Connect queue managers and define a topic:

### About this task

Connect the child queue managers QM2 and QM3 to the parent queue manager QM1.

## Procedure

1. On QM2 and QM3, set the parent queue manager to QM1:  
`ALTER QMGR PARENT (QM1)`
2. Run the following command on all queue managers to check that the child queue managers are connected to the parent queue manager:  
`DISPLAY PUBSUB TYPE(ALL)`  
Command output is displayed. For example, here is output for QM1, with the key details highlighted:  
`DISPLAY PUBSUB ALL`  
`1 : DISPLAY PUBSUB ALL`  
`AMQ8723: Display pub/sub status details.`  
`QMNAME(QM1) TYPE(LOCAL)`  
`STATUS(ACTIVE) SUBCOUNT(6)`  
`TPCOUNT(9)`  
`AMQ8723: Display pub/sub status details.`  
**`QMNAME(QM2) TYPE(CHILD)`**  
**`STATUS(ACTIVE) SUBCOUNT(NONE)`**  
`TPCOUNT(NONE)`  
`AMQ8723: Display pub/sub status details.`  
**`QMNAME(QM3) TYPE(CHILD)`**  
**`STATUS(ACTIVE) SUBCOUNT(NONE)`**  
`TPCOUNT(NONE)`

## Scenario 3 part 4: Publish and subscribe the topic:

### About this task

Use the `amqspub.exe` and `amqssub.exe` applications to publish and subscribe the topic.

## Procedure

1. Run this command in the first command window:  
amqspub Sport/Soccer QM2
2. Run this command in the second command window:  
amqssub Sport/Soccer QM1
3. Run this command in the third command window:  
amqssub Sport/Soccer QM3

## Results

The amqssub.exe applications in the second and third command windows receive the messages published in the first command window.

### Related tasks:

“Publish/subscribe hierarchy scenario 1: Using point-to-point channels with queue manager name alias” on page 35

This is the first in a set of three scenarios that set up a publish/subscribe hierarchy in different ways to establish the connection between queue managers. This scenario sets up a publish/subscribe hierarchy that uses point-to-point channels with queue manager name alias.

“Publish/subscribe hierarchy scenario 2: Using point-to-point channels with same name for transmission queue and remote queue manager” on page 38

This is the second in a set of three scenarios that set up a publish/subscribe hierarchy in different ways to establish the connection between queue managers. This scenario sets up a publish/subscribe hierarchy that uses point-to-point channels with the transmission queue name the same as the remote queue manager.

### Related information:

Connecting a queue manager to a publish/subscribe hierarchy

---

## Transactional support scenarios

Using transactional support you can enable your applications to work reliably with databases.

This section introduces transactional support. The work required to enable your applications to use IBM MQ with a database product spans the areas of application programming and system administration. Use the information here together with Committing and backing out units of work.

We start by introducing the units of work that form transactions, then describe the ways in which you enable IBM MQ to coordinate transactions with databases.

## Related concepts:

“Introducing units of work”

This topic introduces and defines the general concepts of unit of work, commit, backout and sync point. It also contains two scenarios that illustrate global units of work.

## Related information:

IBM MQ and HP NonStop TMF

## Introducing units of work

This topic introduces and defines the general concepts of unit of work, commit, backout and sync point. It also contains two scenarios that illustrate global units of work.

When a program puts messages on queues within a unit of work, those messages are made visible to other programs only when the program *commits* the unit of work. To commit a unit of work, all updates must be successful to preserve data integrity.

If the program detects an error and decides not to make the put operation permanent, it can *back out* the unit of work. When a program performs a backout, IBM MQ restores the queues by removing the messages that were put on the queues by that unit of work.

Similarly, when a program gets messages from one or more queues within a unit of work, those messages remain on the queues until the program commits the unit of work, but the messages are not available to be retrieved by other programs. The messages are permanently deleted from the queues when the program commits the unit of work. If the program backs out the unit of work, IBM MQ restores the queues by making the messages available to be retrieved by other programs.

The decision to commit or back out the changes is taken, in the simplest case, at the end of a task. However, it can be more useful for an application to synchronize data changes at other logical points within a task. These logical points are called sync points (or synchronization points) and the period of processing a set of updates between two sync points is called a *unit of work*. Several MQGET calls and MQPUT calls can be part of a single unit of work.

With IBM MQ, we need to distinguish between *local* and *global* units of work:

### Local units of work

Are those in which the only actions are puts to, and gets from, IBM MQ queues, and the coordination of each unit of work is provided within the queue manager using a *single-phase commit* process.

Use local units of work when the only resources to be updated are the queues that are managed by a single IBM MQ queue manager. Updates are committed by using the MQCMIT verb or backed out using MQBACK.

There are no system administration tasks, other than log management, which is involved in using local units of work. In your applications, where you use the MQPUT and MQGET calls with MQCMIT and MQBACK, try using the MQPMO\_SYNCPOINT and MQGMO\_SYNCPOINT options. (For information about log management, see Managing log files.)

### Global units of work

Are those in which other resources, such as tables in a relational database, are also updated. When more than one *resource manager* is involved, there is a need for *transaction manager* software that uses a *two-phase commit* process to coordinate the global unit of work.

Use global units of work when you also need to include updates to relational database manager software, such as DB2<sup>®</sup>, Oracle, Sybase, and Informix<sup>®</sup>.

There are several possible scenarios for using global units of work. Documented here are two scenarios:

1. In the first, the queue manager itself acts as the transaction manager. In this scenario, MQI verbs control the global units of work; they are started in applications using the MQBEGIN verb and then committed using MQCMIT or backed out using MQBACK.
2. In the second, the transaction manager role is performed by other software, such as TXSeries®, Encina, or Tuxedo. In this scenario, an API provided by the transaction manager software is used to control the unit of work (for example, EXEC CICS® SYNCPOINT for TXSeries ).

The following sections describe all the steps necessary to use global units of work, organized by the two scenarios:

- Scenario 1: Queue manager performs the coordination
- “Scenario 2: Other software provides the coordination” on page 71

## Scenario 1: Queue manager performs the coordination

### distributed

In scenario 1, the queue manager acts as the transaction manager. In this scenario, MQI verbs control the global units of work; they are started in applications using the MQBEGIN verb and then committed using MQCMIT or backed out using MQBACK.

### Isolation level

In IBM MQ, a message on a queue might be visible before a database update, depending on the transaction isolation design implemented within the database.

When an IBM MQ queue manager is working as an XA transaction manager, to coordinate updates to XA resource managers, the following commit protocol is followed:

1. Prepare all XA resource managers.
2. Commit the IBM MQ queue manager resource manager.
3. Commit other resource managers.

Between step 2 and 3, an application might see a message that is committed to the queue but the corresponding row in the database does not reflect this message.

This is not a problem if the database is configured such that the application's database API calls wait for pending updates to be completed.

You can resolve this by configuring the database differently. The type of configuration needed is referred to as the "isolation level". For more information on isolation levels, refer to the database documentation. You can, alternatively, configure the queue manager to commit the resource managers in the following reverse order:

1. Prepare all XA resource managers.
2. Commit other resource managers.
3. Commit the IBM MQ queue manager resource manager.

When you change the protocol the IBM MQ queue manager is committed last, so applications that read messages from the queues see a message only after the corresponding database update has been completed.

To configure the queue manager to use this changed protocol, set the **AMQ\_REVERSE\_COMMIT\_ORDER** environment variable.

Set this environment variable in the environment from which the **strmqm** is run to start the queue manager. For example, run the following in the shell just before starting the queue manager:

```
export AMQ_REVERSE_COMMIT_ORDER=1
```

**Note:** Setting this environment variable might cause an extra log entry per transaction, so this will have a small impact on the performance of each transaction.

## Database coordination

When the queue manager coordinates global units of work itself, it becomes possible to integrate database updates within the units of work. That is, a mixed MQI and SQL application can be written, and the MQCMIT and MQBACK verbs can be used to commit or roll back the changes to the queues and databases together.

The queue manager achieves this using the two-phase commit protocol described in *X/Open Distributed Transaction Processing: The XA Specification*. When a unit of work is to be committed, the queue manager first asks each participating database manager whether it is prepared to commit its updates. Only if all the participants, including the queue manager itself, are prepared to commit, are all the queue and database updates committed. If any participant cannot prepare its updates, the unit of work is backed out instead.

In general, a global unit of work is implemented in an application by the following method (in pseudocode):

```
MQBEGIN
MQGET (include the flag MQGMO_SYNCPOINT in the message options)
MQPUT (include the flag MQPMO_SYNCPOINT in the message options)
SQL INSERT
MQCMIT
```

The purpose of MQBEGIN is to denote the beginning of a global unit of work. The purpose of MQCMIT is to denote the end of the global unit of work, and to complete it with all participating resource managers, using the two-phase commit protocol.

When the unit of work (also known as a *transaction*) is completed successfully using MQCMIT, all actions taken within that unit of work are made permanent or irreversible. If, for any reason, the unit of work fails, all actions are instead backed out. It is not possible for one action in a unit of work to be made permanent while another is backed out. This is the principle of a unit of work: either all actions within the unit of work are made permanent or none of them are.

### Note:

1. The application programmer can force a unit of work to be backed out by calling MQBACK. The unit of work is also backed out by the queue manager if the application or database *fails* before MQCMIT is called.
2. If an application calls MQDISC without calling MQCMIT, the queue manager behaves as if MQCMIT had been called, and commits the unit of work.

In between MQBEGIN and MQCMIT, the queue manager does not make any calls to the database to update its resources. That is, the only way a database's tables are changed is by your code (for example, the SQL INSERT in the pseudocode).

Full recovery support is provided if the queue manager loses contact with any of the database managers during the commit protocol. If a database manager becomes unavailable while it is in doubt, that is, it has successfully prepared to commit, but has yet to receive a commit or backout decision, the queue manager remembers the outcome of the unit of work until that outcome has been successfully delivered to the database. Similarly, if the queue manager terminates with incomplete commit operations outstanding, these are remembered over queue manager restart. If an application terminates unexpectedly, the integrity of the unit of work is not compromised, but the outcome depends on where in the process the application terminated, as described in Table 2 on page 48.

What happens when the database or application program fails is summarized in the following tables:

Table 1. What happens when a database server fails

| Failure occurrence                                                                                                                                                                          | Outcome                                                                                                         |
|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|-----------------------------------------------------------------------------------------------------------------|
| Before the application call to MQCMIT.                                                                                                                                                      | The unit of work is backed out.                                                                                 |
| During the application call to MQCMIT, <b>before</b> all databases have indicated that they have successfully prepared.                                                                     | The unit of work is backed out with a reason code of MQRC_BACKED_OUT.                                           |
| During the application call to MQCMIT, <b>after</b> all databases have indicated that they have successfully prepared, but before all have indicated that they have successfully committed. | The unit of work is held in recoverable state by the queue manager, with a reason code of MQRC_OUTCOME_PENDING. |
| During the application call to MQCMIT, <b>after</b> all databases have indicated that they have successfully committed.                                                                     | The unit of work is committed with a reason code of MQRC_NONE.                                                  |
| After the application call to MQCMIT.                                                                                                                                                       | The unit of work is committed with a reason code of MQRC_NONE.                                                  |

Table 2. What happens when an application program fails

| Failure occurrence                                                                                                    | Outcome                                                                                                                                                            |
|-----------------------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Before the application call to MQCMIT.                                                                                | The unit of work is backed out.                                                                                                                                    |
| During the application call to MQCMIT, <b>before</b> the queue manager has received the application's MQCMIT request. | The unit of work is backed out.                                                                                                                                    |
| During the application call to MQCMIT, <b>after</b> the queue manager has received the application's MQCMIT request.  | The queue manager tries to commit using two-phase commit (subject to the database products successfully executing and committing their parts of the unit of work). |

In the case where the reason code on return from MQCMIT is MQRC\_OUTCOME\_PENDING, the unit of work is remembered by the queue manager until it has been able to reestablish contact with the database server, and tell it to commit its part of the unit of work. Refer to “Considerations when contact is lost with the XA resource manager” on page 64 for information on how and when recovery is done.

The queue manager communicates with database managers using the XA interface as described in *X/Open Distributed Transaction Processing: The XA Specification*. Examples of these function calls are `xa_open`, `xa_start`, `xa_end`, `xa_prepare`, and `xa_commit`. We use the terms *transaction manager* and *resource manager* in the same sense as they are used in the XA specification.

### Restrictions:

There are restrictions to the database coordination support.

The following restrictions apply:

- The ability to coordinate database updates within IBM MQ units of work is **not** supported in an MQI client application. The use of MQBEGIN in a client application fails. A program that calls MQBEGIN must run as a *server* application on the same machine as the queue manager.

**Note:** A *server* application is a program that has been linked with the necessary IBM MQ server libraries; a *client* application is a program that has been linked with the necessary IBM MQ client libraries. See Building applications for IBM MQ MQI clients and Building a procedural application for details on compiling and linking programs that you write in a procedural language.



- The database server can reside on a different machine from the queue manager server, as long as the database client is installed on the same machine as the queue manager, and it supports this function. Consult the database product's documentation to determine whether their client software can be used for two-phase commit systems.
- Although the queue manager behaves as a resource manager (for the purposes of being involved in Scenario 2 global units of work), it is not possible to make one queue manager coordinate another queue manager within its Scenario 1 global units of work.

### Switch load files:

The switch load file is a shared library (a DLL on Windows systems) that is loaded by the code in your IBM MQ application and the queue manager. Its purpose is to simplify the loading of the database's client shared library, and to return the pointers to the XA functions.

The details of the switch load file must be specified before the queue manager is started. The details are placed in the qm.ini file on Windows, UNIX and Linux systems.

- On Windows and Linux (x86 and x86-64 platforms) systems, use the MQ Explorer to update the qm.ini file.
- On all other systems edit the file, qm.ini, directly.

The C source for the switch load file is supplied with the IBM MQ installation if it supports Scenario 1 global units of work. The source contains a function called MQStart. When the switch load file is loaded, the queue manager calls this function, which returns the address of a structure called an *XA switch*.

The XA switch structure exists in the database client shared library, and contains a number of function pointers, as described in Table 3:

*Table 3. XA switch function pointers*

| Function pointer name | XA function | Purpose                                                            |
|-----------------------|-------------|--------------------------------------------------------------------|
| xa_open_entry         | xa_open     | Connect to database                                                |
| xa_close_entry        | xa_close    | Disconnect from database                                           |
| xa_start_entry        | xa_start    | Start a branch of a global unit of work                            |
| xa_end_entry          | xa_end      | Suspend a branch of a global unit of work                          |
| xa_rollback_entry     | xa_rollback | Roll back a branch of a global unit of work                        |
| xa_prepare_entry      | xa_prepare  | Prepare to commit a branch of a global unit of work                |
| xa_commit_entry       | xa_commit   | Commit a branch of a global unit of work                           |
| xa_recover_entry      | xa_recover  | Discover from the database whether it has an in-doubt unit of work |
| xa_forget_entry       | xa_forget   | Allow a database to forget a branch of a global unit of work       |
| xa_complete_entry     | xa_complete | Complete a branch of a global unit of work                         |

During the first MQBEGIN call in your application, the IBM MQ code that executes as part of MQBEGIN loads the switch load file, and calls the xa\_open function in the database shared library. Similarly, during queue manager startup, and on other subsequent occasions, some queue manager processes load the switch load file and call xa\_open.

You can reduce the number of `xa_*` calls by using *dynamic registration*. For a complete description of this optimization technique, see “XA dynamic registration” on page 69.

### Configuring your system for database coordination:

There are several tasks that you must perform before a database manager can participate in global units of works coordinated by the queue manager. These are described here as follows:

- “Installing and configuring the database product”
- “Creating switch load files” on page 51
- “Adding configuration information to the queue manager” on page 52
- “Writing and modifying your applications” on page 53
- “Testing the system” on page 54

*Installing and configuring the database product:*

To install and configure your database product, see the product's own documentation. This topics in this section describe general configuration issues and how they relate to interoperation between IBM MQ and the database.

### Database connections

An application that establishes a standard connection to the queue manager is associated with a thread in a separate local queue manager agent process. (A connection that is not a *fastpath* connection is a *standard* connection in this context. See Connecting to a queue manager using the MQCONNX call.)

When the application issues **MQBEGIN** , both it and the agent process call the `xa_open` function in the database client library. In response to this, the database client library code *connects* to the database that is to be involved in the unit of work *from both the application and queue manager processes*. These database connections are maintained as long as the application remains connected to the queue manager.

This is an important consideration if the database supports only a limited number of users or connections, because two connections are being made to the database to support the one application program.

### Client/server configuration

The database client library that is loaded into the IBM MQ queue manager and application processes **must** be able to send to and receive from its server. Ensure that:

- The database's client/server configuration files have the correct details
- The relevant environment variables are set in the environment of the queue manager **and** the application processes

*Creating switch load files:*

IBM MQ comes with a sample makefile, used to build switch load files for the supported database managers.

`MQ_INSTALLATION_PATH` represents the high-level directory in which IBM MQ is installed.

The sample makefile, together with all the associated C source files required to build the switch load files, is installed in the following directories:

- For IBM MQ for Windows, in the `MQ_INSTALLATION_PATH\tools\c\samples\xatm\` directory
- For IBM MQ for UNIX and Linux systems, in the `MQ_INSTALLATION_PATH/samp/xatm/` directory

The sample source modules used to build the switch load files are:

- For Db2, `db2swit.c`
- For Oracle, `oraswit.c`
- For Informix, `infswit.c`
- For Sybase, `sybswit.c`

When you generate switch load files, install 32-bit switch load files in `/var/mqm/exits` and install 64-bit switch load files in `/var/mqm/exits64`.

**Note:** This switch load file is used by IBM MQ C applications. For Java applications, see JTA/JDBC coordination using IBM MQ classes for Java.

If you have 32-bit queue managers then the sample make file, `xaswit.mak`, installs a 32-bit switch load file in `/var/mqm/exits`.

If you have 64-bit queue managers then the sample make file, `xaswit.mak`, installs a 32-bit switch load file in `/var/mqm/exits`, and a 64-bit switch load file in `/var/mqm/exits64`.

## **File security**

It is possible that your operating system might fail the loading of the switch load file by IBM MQ, for reasons outside the control of IBM MQ. If this occurs, error messages are written to the IBM MQ error logs, and potentially the `MQBEGIN` call can fail. To help to ensure that your operating system does not fail the loading of the switch load file, you must fulfil the following requirements:

1. The switch load file must be available in the location that is given in the `qm.ini` file.
2. The switch load file must be accessible to all processes that need to load it, including the queue manager processes and application processes.
3. All of the libraries upon which the switch load file depends, including the libraries that are provided by the database product, must be present and accessible.

*Adding configuration information to the queue manager:*

When you have created a switch load file for your database manager, and placed it in a safe location, you must specify that location to your queue manager.

To specify the location, perform the following steps:

- On Windows and Linux (x86 and x86-64 platforms) systems use the IBM MQ Explorer. Specify the details of the switch load file in the queue manager properties panel, under XA resource manager.
- On all other systems specify the details of the switch load file in the XAResourceManager stanza in the queue manager's `qm.ini` file.

Add an XAResourceManager stanza for the database that your queue manager is going to coordinate. The most common case is for there to be only one database, and therefore only one XAResourceManager stanza. For details of more complicated configurations involving multiple databases, see "Multiple database configurations" on page 63. The attributes of the XAResourceManager stanza are as follows:

**Name=name**

User-chosen string that identifies the resource manager. In effect, it gives a name to the XAResourceManager stanza. The name is mandatory and can be up to 31 characters in length.

The name you choose must be unique; there must be only one XAResourceManager stanza with this name in this `qm.ini` file. The name should also be meaningful, because the queue manager uses it to refer to this resource manager both in queue manager error log messages and in output when the `dspmqtrn` command is used. (See "Displaying outstanding units of work with the `dspmqtrn` command" on page 65 for more information.)

Once you have chosen a name, and have started the queue manager, do not change the Name attribute. For more details about changing configuration information, see "Changing configuration information" on page 68.

**SwitchFile=name**

This is the name of the XA switch load file you built earlier. This is a mandatory attribute. The code in the queue manager and IBM MQ application processes tries to load the switch load file on two occasions:

1. At queue manager startup
2. When you make the first call to MQBEGIN in your IBM MQ application process

The security and permissions attributes of your switch load file must allow these processes to perform this action.

**XAOpenString=string**

This is a string of data that IBM MQ code passes in its calls to the database manager's `xa_open` function. This is an optional attribute; if it is omitted a zero-length string is assumed.

The code in the queue manager and IBM MQ application processes call the `xa_open` function on two occasions:

1. At queue manager startup
2. When you make the first call to MQBEGIN in your IBM MQ application process

The format for this string is particular to each database product, and will be described in the documentation for that product. In general, the `xa_open` string contains authentication information (user name and password) to allow a connection to the database in both the queue manager and the application processes.

**V8.0.0.4** From IBM MQ Version 8.0.0, Fix Pack 4, when the XAOpenString contains a password, you can get IBM MQ to protect this information, rather than having the password visible in plain text in the `qm.ini` file. IBM MQ stores the user name and the password (in an encrypted form) in a different file, and uses these credentials to connect to the database. For details, see Protection of database authentication details.

**XACloseString=string**

This is a string of data that IBM MQ code passes in its calls to the database manager's `xa_close` function. This is an optional attribute; if it is omitted a zero-length string is assumed.

The code in the queue manager and IBM MQ application processes call the `xa_close` function on two occasions:

1. At queue manager startup
2. When you make a call to `MQDISC` in your IBM MQ application process, having earlier made a call to `MQBEGIN`

The format for this string is particular to each database product, and will be described in the documentation for that product. In general, the string is empty, and it is common to omit the `XACloseString` attribute from the `XAResourceManager` stanza.

**ThreadOfControl=THREAD| PROCESS**

The `ThreadOfControl` value can be `THREAD` or `PROCESS`. The queue manager uses it for serialization purposes. This is an optional attribute; if it is omitted, the value `PROCESS` is assumed.

If the database client code allows threads to call the XA functions without serialization, the value for `ThreadOfControl` can be `THREAD`. The queue manager assumes that it can call the XA functions in the database client shared library from multiple threads at the same time, if necessary.

If the database client code does not allow threads to call its XA functions in this way, the value for `ThreadOfControl` must be `PROCESS`. In this case, the queue manager serializes all calls to the database client shared library so that only one call at a time is made from within a particular process. You probably also need to ensure that your application performs similar serialization if it runs with multiple threads.

Note that this issue, of the database product's ability to cope with multi-threaded processes in this way, is an issue for that product's vendor. Consult the database product's documentation for details on whether you can set the `ThreadOfControl` attribute to `THREAD` or `PROCESS`. We recommend that, if you can, you set `ThreadOfControl` to `THREAD`. If in doubt, the *safer* option is to set it to `PROCESS`, although you will lose the potential performance benefits of using `THREAD`.

*Writing and modifying your applications:*

How to implement a global unit of work.

The sample application programs for Scenario 1 global units of work that are supplied with an IBM MQ installation are described in the "Introducing units of work" on page 45.

In general, a global unit of work is implemented in an application by the following method (in pseudocode):

```
MQBEGIN
MQGET
MQPUT
SQL INSERT
MQCMIT
```

The purpose of `MQBEGIN` is to denote the beginning of a global unit of work. The purpose of `MQCMIT` is to denote the end of the global unit of work, and to complete it with all participating resource managers, using the two-phase commit protocol.

In between `MQBEGIN` and `MQCMIT`, the queue manager does not make any calls to the database to update its resources. That is, the only way a database's tables are changed is by your code (for example, the `SQL INSERT` in the pseudocode).

The role of the queue manager, as far as the database is concerned, is to tell it when a global unit of work has started, when it has ended, and whether the global unit of work should be committed or rolled-back.

As far as your application is concerned, the queue manager performs two roles: a resource manager (where the resources are messages on queues) and the transaction manager for the global unit of work.

Start with the supplied sample programs, and work through the various IBM MQ and database API calls that are being made in those programs. The API calls concerned are fully documented in *Sample IBM MQ procedural programs, Data types used in the MQI*, and (in the case of the database's own API) the database's own documentation.

#### *Testing the system:*

You know whether your application and system are correctly configured only by running them during testing. You can test the system's configuration (the successful communication between queue manager and database) by building and running one of the supplied sample programs.

## **Configuring Db2**

Db2 support and configuration information.

The supported levels of Db2 are defined at the WebSphere MQ detailed system requirements page.

**Note:** 32-bit instances of Db2 are not supported on platforms where the queue manager is 64-bit.

Do the following:

1. Check the environment variable settings.
2. Create the Db2 switch load file.
3. Add resource manager configuration information.
4. Change Db2 configuration parameters if necessary.

Read this information in conjunction with the general information provided in "Configuring your system for database coordination" on page 50.

**Warning:** If you run `db2profile` on UNIX and Linux platforms, the environment variable `LIBPATH` and `LD_LIBRARY_PATH` are set. It is advisable to unset these environment variables. See `crtmqenv` or `setmqenv` for more information.

## **Checking the Db2 environment variable settings**

Ensure that your Db2 environment variables are set for queue manager processes *as well as in* your application processes. In particular, you must always set the `DB2INSTANCE` environment variable *before* you start the queue manager. The `DB2INSTANCE` environment variable identifies the Db2 instance containing the Db2 databases that are being updated. For example:

- On UNIX and Linux systems, use:  
`export DB2INSTANCE=db2inst1`
- On Windows systems, use:  
`set DB2INSTANCE=DB2`

On Windows with a Db2 database, you must add the user `MUSR_MQADMIN` to the `DB2USERS` group, to enable the queue manager to start.

## **Creating the Db2 switch load file**

The easiest way to create the Db2 switch load file is to use the sample file `xaswit.mak`, which IBM MQ provides to build the switch load files for a variety of database products.

On Windows systems, you can find `xaswit.mak` in the directory `MQ_INSTALLATION_PATH\tools\c\samples\xatm`. `MQ_INSTALLATION_PATH` represents the high-level directory in which IBM MQ is installed. To create the Db2 switch load file with Microsoft Visual C++, use:

```
nmake /f xaswit.mak db2swit.dll
```

The generated switch file is placed in `C:\Program Files\IBM\WebSphere MQ\exits`.

You can find `xaswit.mak` in the directory `MQ_INSTALLATION_PATH/samp/xatm`. `MQ_INSTALLATION_PATH` represents the high-level directory in which IBM MQ is installed.

Edit `xaswit.mak` to *uncomment* the lines appropriate to the version of Db2 you are using. Then execute the makefile using the command:

```
make -f xaswit.mak db2swit
```

The generated 32-bit switch load file is placed in `/var/mqm/exits`.

The generated 64-bit switch load file is placed in `/var/mqm/exits64`.

## Adding resource manager configuration information for Db2

You must modify the configuration information for the queue manager to declare Db2 as a participant in global units of work. Modifying configuration information in this way is described in more details in “Adding configuration information to the queue manager” on page 52.

- On Windows and Linux (x86 and x86-64 platforms) systems, use the IBM MQ Explorer. Specify the details of the switch load file in the queue manager properties panel, under XA resource manager.
- On all other systems specify the details of the switch load file in the `XAResourceManager` stanza in the queue manager's `qm.ini` file.

Figure 6 is a UNIX sample, showing an `XAResourceManager` entry where the database to be coordinated is called `mydbname`, this name being specified in the `XAOpenString`:

```
XAResourceManager:
 Name=mydb2
 SwitchFile=db2swit
 XAOpenString=mydbname,myuser,mypasswd,toc=t
 ThreadOfControl=THREAD
```

Figure 6. Sample `XAResourceManager` entry for Db2 on UNIX platforms

### Note:

1. `ThreadOfControl=THREAD` cannot be used with Db2 versions earlier than version 8. Set `ThreadOfControl` and the `XAOpenString` parameter `toc` to one of the following combinations:

- `ThreadOfControl=THREAD` and `toc=t`
- `ThreadOfControl=PROCESS` and `toc=p`

If you are using the `jdbcdb2` XA switch load file to enable JDBC/JTA coordination, you must use `ThreadOfControl=PROCESS` and `toc=p`.

## Changing Db2 configuration parameters

For each Db2 database that the queue manager is coordinating, you must set database privileges, change the `tp_mon_name` parameter, and reset the `maxappls` parameter. To do this, perform the following steps:

## Set database privileges

The queue manager processes run with effective user and group mqm on UNIX and Linux systems. On Windows systems, they run as the user that started the queue manager. This can be one of:

1. The user who issued the **strmqm** command, or
2. The user under which the IBM MQ Service COM server runs

By default, this user is called MUSR\_MQADMIN.

If you have not specified a user name and password on the xa\_open string, **the user under which the queue manager is running** is used by Db2 to authenticate the xa\_open call. If this user (for example, user mqm on UNIX and Linux systems) does not have minimal privileges in the database, the database refuses to authenticate the xa\_open call.

The same considerations apply to your application process. If you have not specified a user name and password on the xa\_open string, the user under which your application is running is used by Db2 to authenticate the xa\_open call that is made during the first MQBEGIN. Again, this user must have minimal privileges in the database for this to work.

For example, give the mqm user connect authority in the mydbname database by issuing the following Db2 commands:

```
db2 connect to mydbname
db2 grant connect on database to user mqm
```

See “Security considerations” on page 64 for more information about security.

### **Windows** Change the TP\_MON\_NAME parameter

For Db2 on Windows systems only, change the TP\_MON\_NAME configuration parameter to name the DLL that Db2 uses to call the queue manager for dynamic registration.

Use the command db2 update dbm cfg using TP\_MON\_NAME mqmax to name MQMAX.DLL as the library that Db2 uses to call the queue manager. This must be present in a directory within PATH.

## Reset the maxappls parameter

You might need to review your setting for the *maxappls* parameter, which limits the maximum number of applications that can be connected to a database. Refer to “Installing and configuring the database product” on page 50.

## Configuring Oracle

Oracle support and configuration information.

Complete the following steps:

1. Check environment variable settings.
2. Create the Oracle switch load file.
3. Add resource manager configuration information.
4. Change the Oracle configuration parameters, if necessary.

A current list of levels of Oracle supported by IBM MQ is provided at the WebSphere MQ detailed system requirements page.

## Checking the Oracle environment variable settings

Ensure that your Oracle environment variables are set for queue manager processes as well as in your application processes. In particular, always set the following environment variables before starting the queue manager:

### ORACLE\_HOME

The Oracle home directory. For example, on UNIX and Linux systems, use:

```
export ORACLE_HOME=/opt/oracle/product/8.1.6
```



On Windows systems, use:

```
set ORACLE_HOME=c:\oracle\ora81
```

### **ORACLE\_SID**

The Oracle SID being used. If you are using Net8 for client/server connectivity, you might not have to set this environment variable. Consult your Oracle documentation.

The subsequent example is an example of setting this environment variable, on UNIX and Linux systems:

```
export ORACLE_SID=sid1
```

The equivalent on Windows systems is:

```
set ORACLE_SID=sid1
```

If you run queue managers on Windows 64 bit systems, then only 64 bit Oracle clients must be installed. The switch load file, loaded by 64 bit queue managers, must access the Oracle 64 bit client libraries.

### **Creating the Oracle switch load file**

To create the Oracle switch load file, use the sample file `xaswit.mak`, which IBM MQ provides to build the switch load files for various database products. On Windows systems, you can find `xaswit.mak` in the directory `C:\Program Files\IBM\WebSphere MQ\tools\c\samples\atm`. To create the Oracle switch load file with Microsoft Visual C++, use: `nmake /f xaswit.mak oraswit.dll`

**Note:** These switch load files can be used only with C applications. For Java applications, see JTA/JDBC coordination using IBM MQ classes for Java.

The generated switch file is placed in `MQ_INSTALLATION_PATH\exits`. `MQ_INSTALLATION_PATH` represents the high-level directory in which IBM MQ is installed.

You can find `xaswit.mak` in the directory `MQ_INSTALLATION_PATH/samp/atm`. `MQ_INSTALLATION_PATH` represents the high-level directory in which IBM MQ is installed.

Edit `xaswit.mak` to uncomment the lines appropriate to the version of Oracle you are using. Then execute the makefile using the command:

```
make -f xaswit.mak oraswit
```

The contents of `MQ_INSTALLATION_PATH/samp/atm` are read-only when IBM MQ is installed, so to edit `xaswit.mak`, copy all the files out of `samp/atm` to another directory, modify `xaswit.mak`, and then run `make -f xaswit.mak oraswit` from that directory.

The generated 32 bit switch load file is placed in `/var/mqm/exits`.

The generated 64 bit switch load file is placed in `/var/mqm/exits64`.

### **Adding resource manager configuration information for Oracle**

You must modify the configuration information for the queue manager to declare Oracle as a participant in global units of work. Modifying the configuration information for the queue manager in this way is described in more detail in “Adding configuration information to the queue manager” on page 52.

- On Windows and Linux (x86 and x86-64 platforms) systems, use the MQ Explorer. Specify the details of the switch load file in the queue manager properties panel, under XA resource manager.
- On all other systems specify the details of the switch load file in the XAResourceManager stanza in the `qm.ini` file of the queue manager.

Figure 7 is a UNIX and Linux systems sample showing an XAResourceManager entry. You must add a LogDir to the XA open string so that all error and tracing information is logged to the same place.

```
XAResourceManager:
 Name=myoracle
 SwitchFile=oraswit
 XAOpenString=Oracle_XA+Acc=P/myuser/mypasswd+SesTm=35+LogDir=/tmp+threads=true
 ThreadOfControl=THREAD
```

Figure 7. Sample XAResourceManager entry for Oracle on UNIX and Linux platforms

**Note:**

1. In Figure 7, the xa\_open string has been used with four parameters. Additional parameters can be included as described in Oracle's documentation.
2. When using the IBM MQ parameter ThreadOfControl=THREAD you must use the Oracle parameter +threads=true in the XAResourceManager stanza.

See the *Oracle8 Server Application Developer's Guide* for more information about the xa\_open string.

### Changing Oracle configuration parameters

For each Oracle database that the queue manager is coordinating, you must review your maximum sessions and set database privileges. To do so, complete these steps:

#### Review your maximum sessions

You might have to review your LICENSE\_MAX\_SESSIONS and PROCESSES settings to take into account the additional connections required by processes belonging to the queue manager. See "Installing and configuring the database product" on page 50 for more details.

#### Set database privileges

The Oracle user name specified in the xa\_open string must have privileges to access the DBA\_PENDING\_TRANSACTIONS view, as described in the Oracle documentation.

The necessary privilege can be given using the following example command:

```
grant select on DBA_PENDING_TRANSACTIONS to myuser;
```

### Configuring Informix

Informix support and configuration information.

Complete the following steps:

1. Ensure that you have installed the appropriate Informix client SDK:
  - 32 bit queue managers and applications require a 32 bit Informix client SDK.
  - 64 bit queue managers and applications require a 64 bit Informix client SDK.
2. Ensure that Informix databases are created correctly.
3. Check environment variable settings.
4. Build the Informix switch load file.
5. Add resource manager configuration information.

A current list of levels of Informix supported by IBM MQ is provided at the WebSphere MQ detailed system requirements page.

### Ensuring that Informix databases are created correctly

Every Informix database that is to be coordinated by an IBM MQ queue manager must be created specifying the log parameter. For example:

```
create database mydbname with log;
```

IBM MQ queue managers are unable to coordinate Informix databases that do not have the `log` parameter specified on creation. If a queue manager attempts to coordinate an Informix database that does not have the `log` parameter specified on creation, the `xa_open` call to Informix fails, and a number of FFST™ errors are generated.

## Checking the Informix environment variable settings

Ensure that your Informix environment variables are set for queue manager processes *as well as in* your application processes. In particular, always set the following environment variables **before** starting the queue manager:

### INFORMIXDIR

The directory of the Informix product installation.

- For 32 bit UNIX and Linux applications, use the following command:  
`export INFORMIXDIR=/opt/informix/32-bit`
- For 64 bit UNIX and Linux applications, use the following command:  
`export INFORMIXDIR=/opt/informix/64-bit`
- For Windows applications, use the following command:  
`set INFORMIXDIR=c:\informix`

For systems that have 64 bit queue managers that must support both 32 bit and 64 bit applications, you need both the Informix 32 bit and 64 bit client SDKs installed. The sample makefile `xaswit.mak`, used for creating a switch load file also sets both product installation directories.

### INFORMIXSERVER

The name of the Informix server. For example, on UNIX and Linux systems, use:

```
export INFORMIXSERVER=hostname_1
```

On Windows systems, use:

```
set INFORMIXSERVER=hostname_1
```

### ONCONFIG

The name of the Informix server configuration file. For example, on UNIX and Linux systems, use:

```
export ONCONFIG=onconfig.hostname_1
```

On Windows systems, use:

```
set ONCONFIG=onconfig.hostname_1
```

## Creating the Informix switch load file

To create the Informix switch load file, use the sample file `xaswit.mak`, which IBM MQ provides to build the switch load files for various database products. On Windows systems, you can find `xaswit.mak` in the directory `MQ_INSTALLATION_PATH\tools\c\samples\xatm`. `MQ_INSTALLATION_PATH` represents the high-level directory in which IBM MQ is installed. To create the Informix switch load file with Microsoft Visual C++, use:

```
nmake /f xaswit.mak infswit.dll
```

The generated switch file is placed in `C:\Program Files\IBM\WebSphere MQ\exits`.

You can find `xaswit.mak` in the directory `MQ_INSTALLATION_PATH/samp/xatm`. `MQ_INSTALLATION_PATH` represents the high-level directory in which IBM MQ is installed.

Edit `xaswit.mak` to *uncomment* the lines appropriate to the version of Informix you are using. Then execute the makefile using the command:

```
make -f xaswit.mak infswit
```

The generated 32 bit switch load file is placed in /var/mqm/exits.

The generated 64 bit switch load file is placed in /var/mqm/exits64.

## Adding resource manager configuration information for Informix

You must modify the configuration information for the queue manager to declare Informix as a participant in global units of work. Modifying the configuration information for the queue manager in this way is described in more detail in “Adding configuration information to the queue manager” on page 52.

- On Windows and Linux (x86 and x86-64 platforms) systems, use the IBM MQ Explorer. Specify the details of the switch load file in the queue manager properties panel, under XA resource manager.
- On all other systems specify the details of the switch load file in the XAResourceManager stanza in the qm.ini file of the queue manager.

Figure 8 is a UNIX sample, showing a qm.ini XAResourceManager entry where the database to be coordinated is called mydbname, this name being specified in the XAOpenString:

```
XAResourceManager:
 Name=myinformix
 SwitchFile=infswit
 XAOpenString=DB=mydbname@myinformixserver\;USER=myuser\;PASSWD=mypasswd
 ThreadOfControl=THREAD
```

Figure 8. Sample XAResourceManager entry for Informix on UNIX platforms

**Note:** By default the sample xaswit.mak on UNIX platforms creates a switch load file that uses threaded Informix libraries. You must ensure that ThreadOfControl is set to THREAD when using these Informix libraries. In Figure 8, the qm.ini file XAResourceManager stanza attribute ThreadOfControl is set to THREAD. When THREAD is specified, applications must be built using the threaded Informix libraries and the IBM MQ threaded API libraries.

The XAOpenString attribute must contain the database name, followed by the @ symbol, and then followed by the Informix server name.

To use the nonthreaded Informix libraries, you must ensure that the qm.ini file XAResourceManager stanza attribute ThreadOfControl is set to PROCESS. You must also make the following changes to the sample xaswit.mak:

1. Uncomment the generation of a nonthreaded switch load file.
2. Comment out the generation of the threaded switch load file.

## Sybase configuration

Sybase support and configuration information.

Complete the following steps:

1. Ensure you have installed the Sybase XA libraries, for example by installing the XA DTM option.
2. Check environment variable settings.
3. Enable Sybase XA support.
4. Create the Sybase switch load file.
5. Add resource manager configuration information.

A current list of levels of Sybase supported by IBM MQ is provided at the WebSphere MQ detailed system requirements page.

## Checking the Sybase environment variable settings

Ensure that your Sybase environment variables are set for queue manager processes *as well as in* your application processes. In particular, always set the following environment variables **before** starting the queue manager:

### SYBASE

The location of the Sybase product installation. For example, on UNIX and Linux systems, use:  
export SYBASE=/sybase

On Windows systems, use:

```
set SYBASE=c:\sybase
```

### SYBASE\_OCS

The directory under SYBASE where you have installed the Sybase client files. For example, on UNIX and Linux systems, use:

```
export SYBASE_OCS=OCS-12_0
```

On Windows systems, use:

```
set SYBASE_OCS=OCS-12_0
```

## Enabling Sybase XA support

Within the Sybase XA configuration file `SYBASE/SYBASE_OCS/xa_config`, define a Logical Resource Manager (LRM) for each connection to the Sybase server that is being updated. An example of the contents of `SYBASE/SYBASE_OCS/xa_config` is shown in Figure 9.

```
The first line must always be a comment

[xa]

LRM=lrmname
server=servername
```

Figure 9. Example contents of `SYBASE/SYBASE_OCS/xa_config`

## Creating the Sybase switch load file

To create the Sybase switch load file, use the sample files supplied with IBM MQ. On Windows systems, you can find `xaswit.mak` in the directory `C:\Program Files\IBM\WebSphere MQ\tools\c\samples\xatm`. To create the Sybase switch load file with Microsoft Visual C++, use:

```
nmake /f xaswit.mak sybswit.dll
```

The generated switch file is placed in C:\Program Files\IBM\WebSphere MQ\exits.

You can find xaswit.mak in the directory `MQ_INSTALLATION_PATH/samp/xatm`. `MQ_INSTALLATION_PATH` represents the high-level directory in which IBM MQ is installed.

Edit xaswit.mak to *uncomment* the lines appropriate to the version of Sybase you are using. Then execute the makefile using the command:

```
make -f xaswit.mak sybswit
```

The generated 32-bit switch load file is placed in `/var/mqm/exits`.

The generated 64-bit switch load file is placed in `/var/mqm/exits64`.

## Adding resource manager configuration information for Sybase

You must modify the configuration information for the queue manager to declare Sybase as a participant in global units of work. Modifying the configuration information is described in more detail in “Adding configuration information to the queue manager” on page 52.

- On Windows and Linux (x86 and x86-64 platforms) systems, use the IBM MQ Explorer. Specify the details of the switch load file in the queue manager properties panel, under XA resource manager.
- On all other systems specify the details of the switch load file in the XAResourceManager stanza in the queue manager's `qm.ini` file.

Figure 10 shows a UNIX and Linux sample, which uses the database associated with the `lrmname` LRM definition in the Sybase XA configuration file, `$SYBASE/$SYBASE_OCS/xa_config`. Include a log file name if you want XA function calls to be logged:

```
XAResourceManager:
 Name=mysybase
 SwitchFile=sybswit
 XAOpenString=-Uuser -Ppassword -Nlrmname -L/tmp/sybase.log -Txa
 ThreadOfControl=THREAD
```

Figure 10. Sample XAResourceManager entry for Sybase on UNIX and Linux platforms

## Using multi-threaded programs with Sybase

If you are using multi-threaded programs with IBM MQ global units of work incorporating updates to Sybase, you *must* use the value `THREAD` for the `ThreadOfControl` parameter. Also ensure that you link your program (and the switch load file) with the threadsafe Sybase libraries (the `_r` versions). Using the value `THREAD` for the `ThreadOfControl` parameter is shown in Figure 10.

## Multiple database configurations

If you want to configure the queue manager so that updates to multiple databases can be included within global units of work, add an `XAResourceManager` stanza for each database.

**If the databases are all managed by the same database manager**, each stanza defines a separate database. Each stanza specifies the same `SwitchFile`, but the contents of the `XAOpenString` are different because it specifies the name of the database being updated. For example, the stanzas shown in Figure 11 configure the queue manager with the Db2 databases `MQBankDB` and `MQFeeDB` on UNIX and Linux systems.

**Important:** You cannot have multiple stanzas pointing to the same database. This configuration does not work under any circumstances, and if you try this configuration it fails.

You will receive errors of the form when the MQ code makes its second `xa_open` call in any process in this environment, the database software fails the second `xa_open` with a -5 error, `XAER_INVAL`.

```
XAResourceManager:
Name=DB2 MQBankDB
SwitchFile=db2swit
XAOpenString=MQBankDB

XAResourceManager:
Name=DB2 MQFeeDB
SwitchFile=db2swit
XAOpenString=MQFeeDB
```

Figure 11. Sample `XAResourceManager` entries for multiple Db2 databases

**If the databases to be updated are managed by different database managers**, add an `XAResourceManager` stanza for each. In this case, each stanza specifies a different `SwitchFile`. For example, if `MQFeeDB` is managed by Oracle instead of Db2, use the following stanzas on UNIX and Linux systems:

```
XAResourceManager:
Name=DB2 MQBankDB
SwitchFile=db2swit
XAOpenString=MQBankDB

XAResourceManager:
Name=Oracle MQFeeDB
SwitchFile=oraswit
XAOpenString=Oracle_XA+Acc=P/myuser/mypassword+SesTm=35+LogDir=/tmp/ora.log+DB=MQFeeDB
```

Figure 12. Sample `XAResourceManager` entries for a Db2 and Oracle database

In principle, there is no limit to the number of database instances that can be configured with a single queue manager.

**Note:** For information on support for including Informix databases in multiple database updates within global units of work, check the product readme file.

## Security considerations

Considerations for running your database under the XA model.

The following information is provided for guidance only. In all cases, refer to the documentation provided with the database manager to determine the security implications of running your database under the XA model.

An application process denotes the start of a global unit of work using the **MQBEGIN** verb. The first **MQBEGIN** call that an application issues connects to all participating databases by calling their client library code at the `xa_open` entry point. All the database managers provide a mechanism for supplying a user ID and password in their `XAOpenString`. This is the only time that authentication information flows.

Note that, on UNIX and Linux platforms, fastpath applications must run with an effective user ID of `mqm` while making MQI calls.

## Considerations when contact is lost with the XA resource manager

The queue manager tolerates database managers not being available. This means you can start and stop the queue manager independently from the database server. When contact is restored, the queue manager and database resynchronize. You can also use the `rsvmqtrn` command to manually resolve in-doubt units of work.

In normal operations, only a minimal amount of administration is necessary after you have completed the configuration steps. The administration job is made easier because the queue manager tolerates database managers not being available. In particular this means that:

- The queue manager can start at any time without first starting each of the database managers.
- The queue manager does not need to stop and restart if one of the database managers becomes unavailable.

This allows you to start and stop the queue manager independently from the database server.

Whenever contact is lost between the queue manager and a database, they need to resynchronize when both become available again. Resynchronization is the process by which any in-doubt units of work involving that database are completed. In general, this occurs automatically without the need for user intervention. The queue manager asks the database for a list of units of work that are in doubt. It then instructs the database to either commit or roll back each of these in-doubt units of work.

When a queue manager starts, it resynchronizes with each database. When an individual database becomes unavailable, only that database needs to be resynchronized the next time that the queue manager notices it is available again.

The queue manager regains contact with a previously unavailable database automatically as new global units of work are started with **MQBEGIN**. It does this by calling the `xa_open` function in the database client library. If this `xa_open` call fails, **MQBEGIN** returns with a completion code of `MQCC_WARNING` and a reason code of `MQRC_PARTICIPANT_NOT_AVAILABLE`. You can retry the **MQBEGIN** call later.

Do not continue to attempt a global unit of work that involves updates to a database that has indicated failure during **MQBEGIN**. There will not be a connection to that database through which updates can be made. Your only options are to end the program, or to retry **MQBEGIN** periodically in the hope that the database might become available again.

Alternatively, you can use the `rsvmqtrn` command to resolve explicitly all in-doubt units of work.



### **In-doubt units of work:**

A database might have in-doubt units of work if contact with the queue manager is lost after the database manager has been instructed to prepare. Until the database server receives the outcome from the queue manager (commit or roll back), it needs to retain the database locks associated with the updates.

Because these locks prevent other applications from updating or reading database records, resynchronization needs to take place as soon as possible.

If, for some reason, you cannot wait for the queue manager to resynchronize with the database automatically, you can use facilities provided by the database manager to commit or roll back the database updates manually. In the *X/Open Distributed Transaction Processing: The XA Specification*, this is called making a *heuristic* decision. Use it only as a last resort because of the possibility of compromising data integrity; you might, for example, mistakenly roll back the database updates when all other participants have committed their updates.

It is far better to restart the queue manager, or use the **rsvmqtrn** command when the database has been restarted, to initiate automatic resynchronization.

### **Displaying outstanding units of work with the dspmqtrn command:**

While a database manager is unavailable, you can use the **dspmqtrn** command to check the state of outstanding global units of work involving that database.

The **dspmqtrn** command displays only those units of work in which one or more participants are in doubt. The participants are awaiting the decision from the queue manager to commit or roll back the prepared updates.

For each of these global units of work, the state of each participant is displayed in the output from **dspmqtrn**. If the unit of work did not update the resources of a particular resource manager, it is not displayed.

With respect to an in-doubt unit of work, a resource manager is said to have done one of the following things:

#### **Prepared**

The resource manager is prepared to commit its updates.

#### **Committed**

The resource manager has committed its updates.

#### **Rolled-back**

The resource manager has rolled back its updates.

#### **Participated**

The resource manager is a participant, but has not prepared, committed, or rolled back its updates.

When the queue manager is restarted, it asks each database having an XAResourceManager stanza for a list of its in-doubt global units of work. If the database has not been restarted, or is otherwise unavailable, the queue manager cannot yet deliver to the database the final outcomes for those units of work. The outcome of the in-doubt units of work is delivered to the database at the first opportunity when the database is again available.

In this case, the database manager is reported as being in *prepared* state until resynchronization has occurred.

Whenever the `dspmqtrn` command displays an in-doubt unit of work, it first lists all the possible resource managers that might be participating. These are allocated a unique identifier, *RMId*, which is used instead of the *Name* of the resource managers when reporting their state with respect to an in-doubt unit of work.

Sample `dspmqtrn` output shows the result of issuing the following command:

```
dspmqtrn -m MY_QMGR
```

```
AMQ7107: Resource manager 0 is MQSeries.
AMQ7107: Resource manager 1 is DB2 MQBankDB.
AMQ7107: Resource manager 2 is DB2 MQFeeDB.

AMQ7056: Transaction number 0,1.
 XID: formatID 5067085, gtrid_length 12, bqual_length 4
 gtrid [3291A5060000201374657374]
 bqual [00000001]
AMQ7105: Resource manager 0 has committed.
AMQ7104: Resource manager 1 has prepared.
AMQ7104: Resource manager 2 has prepared.
```

where *Transaction number* is the ID of the transaction which can be used with the `rsvmqtrn` command. See AMQ7xxx: IBM MQ product messages for further information. The *XID* variables are part of the *X/Open XA Specification* ; for the most up-to-date information about this specification see: <http://www.opengroup.org/publications/catalog/c193.htm>.

Figure 13. Sample `dspmqtrn` output

The output in Sample `dspmqtrn` output shows that there are three resource managers associated with the queue manager. The first is resource manager 0, which is the queue manager itself. The other two resource manager instances are the MQBankDB and MQFeeDB Db2 databases.

The example shows only a single in-doubt unit of work. A message is issued for all three resource managers, which means that updates were made to the queue manager and both Db2 databases within the unit of work.

The updates made to the queue manager, resource manager 0, have been *committed*. The updates to the Db2 databases are in *prepared* state, which means that Db2 must have become unavailable before it was called to commit the updates to the MQBankDB and MQFeeDB databases.

The in-doubt unit of work has an external identifier called an *XID* ( *transaction id* ). This is a piece of data given to Db2 by the queue manager to identify its portion of the global unit of work.

### **Resolving outstanding units of work with the rsvmqtrn command:**

Outstanding units of work complete when the queue manager and Db2 resynchronize.

The output shown in Figure 13 on page 66 shows a single in-doubt unit of work in which the commit decision has yet to be delivered to both Db2 databases.

To complete this unit of work, the queue manager and Db2 need to resynchronize when Db2 next becomes available. The queue manager uses the start of new units of work as an opportunity to regain contact with Db2. Alternatively, you can instruct the queue manager to resynchronize explicitly using the **rsvmqtrn** command.

Do this soon after Db2 has been restarted, so that any database locks associated with the in-doubt unit of work are released as quickly as possible. Use the **-a** option, which tells the queue manager to resolve all in-doubt units of work. In the following example, Db2 has restarted, so the queue manager can resolve the in-doubt unit of work:

```
> rsvmqtrn -m MY_QMGR -a
Any in-doubt transactions have been resolved.
```

### **Mixed outcomes and errors:**

Although the queue manager uses a two-phase commit protocol, this does not completely remove the possibility of some units of work completing with mixed outcomes. This is where some participants commit their updates and some back out their updates.

Units of work that complete with a mixed outcome have serious implications because shared resources that should have been updated as a single unit of work are no longer in a consistent state.

Mixed outcomes are mainly caused when heuristic decisions are made about units of work instead of allowing the queue manager to resolve in-doubt units of work itself. Such decisions are outside the queue manager's control.

Whenever the queue manager detects a mixed outcome, it produces FFST information and documents the failure in its error logs, with one of two messages:

- If a database manager rolls back instead of committing:  
AMQ7606 A transaction has been committed but one or more resource managers have rolled back.
- If a database manager commits instead of rolling back:  
AMQ7607 A transaction has been rolled back but one or more resource managers have committed.

Further messages identify the databases that are heuristically damaged. It is then your responsibility to locally restore consistency to the affected databases. This is a complicated procedure in which you need first to isolate the update that has been wrongly committed or rolled back, then to undo or redo the database change manually.

## Changing configuration information:

After the queue manager has successfully started to coordinate global units of work, do not change any of the resource manager configuration information.

If you need to change the configuration information you can do so at any time, but the changes do not take effect until after the queue manager has been restarted.

If you remove the resource manager configuration information for a database, you are effectively removing the ability for the queue manager to contact that database manager.

*Never* change the *Name* attribute in any of your resource manager configuration information. This attribute uniquely identifies that database manager instance to the queue manager. If you change this unique identifier, the queue manager assumes that the database has been removed and a completely new instance has been added. The queue manager still associates outstanding units of work with the old *Name*, possibly leaving the database in an in-doubt state.

### *Removing database manager instances:*

If you need to remove a database from your configuration permanently, ensure that the database is not in doubt before you restart the queue manager.

Database products provide commands for listing in-doubt transactions. If there are any in-doubt transactions, first allow the queue manager to resynchronize with the database. Do this by starting the queue manager. You can verify that resynchronization has taken place by using the **rsvmqtrn** command or the database's own command for viewing in-doubt units of work. Once you are satisfied that resynchronization has taken place, end the queue manager and remove the database's configuration information.

If you fail to observe this procedure the queue manager still remembers all in-doubt units of work involving that database. A warning message, AMQ7623, is issued every time the queue manager is restarted. If you are never going to configure this database with the queue manager again, use the **-r** option of the **rsvmqtrn** command to instruct the queue manager to forget about the database's participation in its in-doubt transactions. The queue manager forgets about such transactions only when in-doubt transactions have been completed with all participants.

There are times when you might need to remove some resource manager configuration information temporarily. On UNIX and Linux systems this is best achieved by commenting out the stanza so that it can be easily reinstated at a later time. You might decide to do this if there are errors every time the queue manager contacts a particular database or database manager. Temporarily removing the resource manager configuration information concerned allows the queue manager to start global units of work involving all the other participants. Here is an example of a commented-out XAResourceManager stanza follows:

```
This database has been temporarily removed
#XAResourceManager:
Name=mydb2
SwitchFile=db2swit
XAOpenString=mydbname,myuser,mypassword,toc=t
ThreadOfControl=THREAD
```

Figure 14. Commented- out XAResourceManager stanza on UNIX and Linux systems

On Windows systems, use the IBM MQ Explorer to delete the information about the database manager instance. Take great care to type in the correct name in the *Name* field when reinstating it. If you mistype the name, you may face in-doubt problems, as described in “Changing configuration information.”

## XA dynamic registration

The XA specification provides a way of reducing the number of `xa_*` calls that a transaction manager makes to a resource manager. This optimization is known as *dynamic registration*.

Dynamic registration is supported by Db2. Other databases might support it; consult the documentation for your database product for details.

Why is the dynamic registration optimization useful? In your application, some global units of work might contain updates to database tables; others might not contain such updates. When no persistent update has been made to a database's tables, there is no need to include that database in the commit protocol that occurs during MQCOMMIT.

Whether or not your database supports dynamic registration, your application calls `xa_open` during the first MQBEGIN call on an IBM MQ connection. It also calls `xa_close` on the subsequent MQDISC call. The pattern of subsequent XA calls depends on whether the database supports dynamic registration:

### If your database does not support dynamic registration...

Every global unit of work involves several XA function calls made by IBM MQ code into the database client library, regardless of whether you made a persistent update to the tables of that database within your unit of work. These include:

- `xa_start` and `xa_end` from the application process. These are used to declare the beginning and end of a global unit of work.
- `xa_prepare`, `xa_commit`, and `xa_rollback` from the queue manager agent process, `amqzlaa0`. These are used to deliver the outcome of the global unit of work: the commit or rollback decision.

In addition, the queue manager agent process also calls `xa_open` during the first MQBEGIN.

### If your database supports dynamic registration...

The IBM MQ code makes only those XA function calls that are necessary. For a global unit of work that has **not** involved persistent updates to database resources, there are **no** XA calls to the database. For a global unit of work that **has** involved such persistent updates, the calls are to:

- `xa_end` from the application process to declare the end of the global unit of work.
- `xa_prepare`, `xa_commit`, and `xa_rollback` from the queue manager agent process, `amqzlaa0`. These are used to deliver the outcome of the global unit of work: the commit or rollback decision.

For dynamic registration to work, it is vital that the database has a way of telling IBM MQ when it has performed a persistent update that it wants to be included in the current global unit of work. IBM MQ provides the `ax_reg` function for this purpose.

The database's client code that runs in your application process finds the `ax_reg` function and calls it, to *dynamically register* the fact it has done persistent work within the current global unit of work. In response to this `ax_reg` call, IBM MQ records that the database has participated. If this is the first `ax_reg` call on this IBM MQ connection, the queue manager agent process calls `xa_open`.

The database client code make this `ax_reg` call when it is running in your process, for example, during an SQL UPDATE call or whatever call in the database's client API is responsible

## Error conditions:

In XA dynamic registration there is a possibility of a confusing failure in the queue manager.

A common example is if you forget to set your database environment variables properly before starting your queue manager, the queue manager's calls to `xa_open` fail. No global units of work can be used.

To avoid this, ensure that you have set the relevant environment variables before starting the queue manager. Review your database product's documentation, and the advice given in "Configuring Db2" on page 54, "Configuring Oracle" on page 56, and "Sybase configuration" on page 61.

With all database products, the queue manager calls `xa_open` once at queue manager startup, as part of the recovery session (as explained in "Considerations when contact is lost with the XA resource manager" on page 64 ). This `xa_open` call fails if you set your database environment variables incorrectly, but it does not cause the queue manager to fail to start. This is because the same `xa_open` error code is used by the database client library to indicate that the database server is unavailable. IBM MQ does not treat this as a serious error, as the queue manager must be able to start to continue processing data outside global units of work involving that database.

Subsequent calls to `xa_open` are made from the queue manager during the first MQBEGIN on an IBM MQ connection (if dynamic registration is not being used) or during a call by the database client code to the IBM MQ-provided `ax_reg` function (if dynamic registration is being used).

The **timing** of any error conditions (or, occasionally, FFST reports) depends on whether you are using dynamic registration:

- If you are using dynamic registration, your MQBEGIN call could succeed, but your SQL UPDATE (or similar) database call will fail.
- If you are not using dynamic registration, your MQBEGIN call will fail.

Ensure that your environment variables are set correctly in your application and queue manager processes.

## Summarizing XA calls:

Here is a list of the calls that are made to the XA functions in a database client library as a result of the various MQI calls that control global units of work. This is not a complete description of the protocol described in the XA specification; it is provided as a brief overview.

Note that `xa_start` and `xa_end` calls are always called by IBM MQ code in the application process, whereas `xa_prepare`, `xa_commit`, and `xa_rollback` are always called from the queue manager agent process, `amqzlaa0`.

The `xa_open` and `xa_close` calls shown in this table are all made from the application process. The queue manager agent process calls `xa_open` in the circumstances described in "Error conditions."

Table 4. Summary of XA function calls

| MQI call                                                                                         | XA calls made with dynamic registration | XA calls made without dynamic registration                                                           |
|--------------------------------------------------------------------------------------------------|-----------------------------------------|------------------------------------------------------------------------------------------------------|
| First MQBEGIN                                                                                    | <code>xa_open</code>                    | <code>xa_open</code><br><code>xa_start</code>                                                        |
| Subsequent MQBEGIN                                                                               | No XA calls                             | <code>xa_start</code>                                                                                |
| MQCMIT ( <b>without</b> <code>ax_reg</code> being called during the current global unit of work) | No XA calls                             | <code>xa_end</code><br><code>xa_prepare</code><br><code>xa_commit</code><br><code>xa_rollback</code> |

Table 4. Summary of XA function calls (continued)

| MQI call                                                                                                          | XA calls made with dynamic registration          | XA calls made without dynamic registration                       |
|-------------------------------------------------------------------------------------------------------------------|--------------------------------------------------|------------------------------------------------------------------|
| MQCMIT ( <b>with</b> ax_reg being called during the current global unit of work)                                  | xa_end<br>xa_prepare<br>xa_commit<br>xa_rollback | Not applicable. No calls are made to ax_reg in non-dynamic mode. |
| MQBACK ( <b>without</b> ax_reg being called during the current global unit of work)                               | No XA calls                                      | xa_end<br>xa_rollback                                            |
| MQBACK ( <b>with</b> ax_reg being called during the current global unit of work)                                  | xa_end<br>xa_rollback                            | Not applicable. No calls are made to ax_reg in non-dynamic mode. |
| MQDISC, where MQCMIT or MQBACK was called first. If they were not, MQCMIT processing is first done during MQDISC. | xa_close                                         | xa_close                                                         |

**Notes:**

1. For MQCMIT, xa\_commit is called if xa\_prepare is successful. Otherwise, xa\_rollback is called.

## Scenario 2: Other software provides the coordination

In scenario 2, an external transaction manager coordinates global units of work, starting and committing them under control of the transaction manager's API. The MQBEGIN, MQCMIT, and MQBACK verbs are unavailable.

This section describes this scenario, including:

- “External sync point coordination”
- “Using CICS” on page 74
- “Using the Microsoft Transaction Server (COM+)” on page 78

The IBM MQ client for HP Integrity NonStop Server can use the HP NonStop Transaction Management Facility (TMF) to coordinate global units of work. For more information, see Using HP NonStop TMF.

### External sync point coordination

A global unit of work can also be coordinated by an external X/Open XA-compliant transaction manager. Here the IBM MQ queue manager participates in, but does not coordinate, the unit of work.

The flow of control in a global unit of work coordinated by an external transaction manager is as follows:

1. An application tells the external sync point coordinator (for example, TXSeries ) that it wants to start a transaction.
2. The sync point coordinator tells known resource managers, such as IBM MQ, about the current transaction.
3. The application issues calls to resource managers associated with the current transaction. For example, the application could issue MQGET calls to IBM MQ.
4. The application issues a commit or backout request to the external sync point coordinator.
5. The sync point coordinator completes the transaction by issuing the appropriate calls to each resource manager, typically using two-phase commit protocols.

The supported levels of external sync point coordinators that can provide a two-phase commit process for transactions in which IBM MQ participates are defined at WebSphere MQ detailed system requirements.

The rest of this section describes how to enable external units of work.

**The IBM MQ XA switch structure:**

Each resource manager participating in an externally coordinated unit of work must provide an XA switch structure. This structure defines both the capabilities of the resource manager and the functions that are to be called by the sync point coordinator.

IBM MQ provides two versions of this structure:

- *MQRMIASwitch* for static XA resource management
- *MQRMIASwitchDynamic* for dynamic XA resource management

Consult your transaction manager documentation to determine whether to use the static or dynamic resource management interface. Wherever a transaction manager supports it, we recommend that you use dynamic XA resource management.

Some 64-bit transaction managers treat the *long* type in the XA specification as 64-bit, and some treat it as 32-bit. IBM MQ supports both models:

- If your transaction manager is 32-bit, or your transaction manager is 64-bit but treats the *long* type as 32-bit, use the switch load file listed in Table 5.
- If your transaction manager is 64-bit and treats the *long* type as 64-bit, use the switch load file listed in Table 6.

A list of known 64-bit transaction managers that treat the *long* type as 64-bit is provided in Table 7 on page 73. Consult your transaction manager documentation if you are unsure which model your transaction manager uses.

*Table 5. XA switch load file names*

| Platform            | Switch load file name (server) | Switch load file name (extended transactional client) |
|---------------------|--------------------------------|-------------------------------------------------------|
| Windows             | <i>mqmxa.dll</i>               | <i>mqcxa.dll</i>                                      |
| AIX® (nonthreaded)  | <i>libmqmxa.a</i>              | <i>libmqcxa.a</i>                                     |
| AIX (threaded)      | <i>libmqmxa_r.a</i>            | <i>libmqcxa_r.a</i>                                   |
| HP-UX (nonthreaded) | <i>libmqmxa.so</i>             | <i>libmqcxa.so</i>                                    |
| HP-UX (threaded)    | <i>libmqmxa_r.so</i>           | <i>libmqcxa_r.so</i>                                  |
| Linux (nonthreaded) | <i>libmqmxa.so</i>             | <i>libmqcxa.so</i>                                    |
| Linux (threaded)    | <i>libmqmxa_r.so</i>           | <i>libmqcxa_r.so</i>                                  |
| Solaris             | <i>libmqmxa.so</i>             | <i>libmqcxa.so</i>                                    |

*Table 6. Alternative 64-bit XA switch load file names*

| Platform            | Switch load file name (server) | Switch load file name (extended transactional client) |
|---------------------|--------------------------------|-------------------------------------------------------|
| AIX (nonthreaded)   | <i>libmqmxa64.a</i>            | <i>libmqcxa64.a</i>                                   |
| AIX (threaded)      | <i>libmqmxa64_r.a</i>          | <i>libmqcxa64_r.a</i>                                 |
| HP-UX (nonthreaded) | <i>libmqmxa64.so</i>           | <i>libmqcxa64.so</i>                                  |
| HP-UX (threaded)    | <i>libmqmxa64_r.so</i>         | <i>libmqcxa64_r.so</i>                                |
| Linux (nonthreaded) | <i>libmqmxa64.so</i>           | <i>libmqcxa64.so</i>                                  |
| Linux (threaded)    | <i>libmqmxa64_r.so</i>         | <i>libmqcxa64_r.so</i>                                |
| Solaris             | <i>libmqmxa64.so</i>           | <i>libmqcxa64.so</i>                                  |



Table 7. 64-bit transaction managers that require the alternative 64-bit switch load file

| Transaction Manager |
|---------------------|
| Tuxedo              |

Some external sync point coordinators (not CICS ) require that each resource manager participating in a unit of work supplies its name in the name field of the XA switch structure. The IBM MQ resource manager name is MQSeries\_XA\_RMI.

The sync point coordinator defines how the IBM MQ XA switch structure links to it. Information about linking the IBM MQ XA switch structure with CICS is provided in “Using CICS” on page 74. For information about linking the IBM MQ XA switch structure with other XA-compliant sync point coordinators, consult the documentation supplied with those products.

The following considerations apply to using IBM MQ with all XA-compliant sync point coordinators:

- It is expected that the transaction manager library code (running as part of their API that the application programmer has called) will call **xa\_open** into IBM MQ at some point, before calling MQCONN.

The **xa\_open** call must be made on the same thread where the MQCONN call is made. The reason for this requirement, is that the XA specification requires that the thread is used to imply context.

Note that this is the approach taken in sample program amqstxsx.c. This sample program assumes that an **xa\_open** call is made into IBM MQ, from the library code of the transaction manager, within their function tpopen).

If no **xa\_open** call is made on the same thread, before the MQCONN call, then the IBM MQ queue manager connection will not be associated with an XA context.

For more information, see MQCTL.

- The xa\_info structure passed on any xa\_open call by the sync point coordinator includes the name of an IBM MQ queue manager. The name takes the same form as the queue-manager name passed to the MQCONN call. If the name passed on the xa\_open call is blank, the default queue manager is used. Alternatively, the xa\_info structure can contain values for the TPM and AXLIB parameters. The TPM parameter specifies the transaction manager being used. The valid values are CICS, TUXEDO and ENCINA. The AXLIB parameter specifies the name of the library that contains the transaction manager's ax\_reg and ax\_unreg functions. For more information on these parameters, see Configuring an extended transactional client. If the xa\_info structure contains either of these parameters, the queue manager name is specified in the QMNAME parameter, unless the default queue manager is being used.
- Only one queue manager at a time can participate in a transaction coordinated by an instance of an external sync point coordinator. The sync point coordinator is effectively connected to the queue manager, and is subject to the rule that only one connection at a time is supported.
- All applications that include calls to an external sync point coordinator can connect only to the queue manager that is participating in the transaction managed by the external coordinator (because they are already effectively connected to that queue manager). However, such applications must issue an MQCONN call to obtain a connection handle, and an MQDISC call before they exit.
- A queue manager with resource updates coordinated by an external sync point coordinator must start before the external sync point coordinator. Similarly, the sync point coordinator must end before the queue manager.
- If your external sync point coordinator terminates abnormally, stop and restart your queue manager *before* restarting the sync point coordinator to ensure that any messaging operations uncommitted at the time of the failure are properly resolved.

## Using CICS

CICS is one of the elements of TXSeries.

The versions of TXSeries that are XA-compliant (and use a two-phase commit process) are defined at: [WebSphere MQ detailed system requirements](#)

IBM MQ also supports other transaction managers. See [WebSphere MQ detailed system requirements](#) for the current lists of supported software.

### Requirements of the two-phase commit process:

Requirements of the two-phase commit process when you use the CICS two-phase commit process with IBM MQ. These requirements do not apply to z/OS®.

Note the following requirements:

- IBM MQ and CICS must reside on the same physical machine.
- IBM MQ does not support CICS on an IBM MQ MQI client.
- You must start the queue manager, with its name specified in the XAD resource definition stanza, *before* you attempt to start CICS. Failure to do this will prevent you from starting CICS if you have added an XAD resource definition stanza for IBM MQ to the CICS region.
- Only one IBM MQ queue manager can be accessed at a time from a single CICS region.
- A CICS transaction must issue an **MQCONN** request before it can access IBM MQ resources. The **MQCONN** call must specify the name of the IBM MQ queue manager specified on the XAOpen entry of the XAD resource definition stanza for the CICS region. If this entry is blank, the **MQCONN** request must specify the default queue manager.
- A CICS transaction that accesses IBM MQ resources must issue an **MQDISC** call from the transaction before returning to CICS. Failure to do this might mean that the CICS application server is still connected, leaving queues open. Additionally, if you do not install a task termination exit (see “Sample task termination exit” on page 77 ), the CICS application server might later end abnormally, perhaps during a subsequent transaction.
- You must ensure that the CICS user ID (cics) is a member of the mqm group, so that the CICS code has the authority to call IBM MQ.

For transactions running in a CICS environment, the queue manager adapts its methods of authorization and determining context as follows:

- The queue manager queries the user ID under which CICS runs the transaction. This is the user ID checked by the Object Authority Manager, and is used for context information.
- In the message context, the application type is MQAT\_CICS.
- The application name in the context is copied from the CICS transaction name.

## General XA support:

**General XA is not supported on IBM i.** An XA switch load module is provided to enable you to link CICS with IBM MQ on UNIX and Linux systems. Additionally, sample source code files are provided to enable you to develop the XA switches for other transaction messages.

The names of the switch load modules provided are:

*Table 8. Essential code for CICS applications: XA initialization routine*

| C (source) | C (exec) - add one of the following to your XAD.Stanza                                                                                            |
|------------|---------------------------------------------------------------------------------------------------------------------------------------------------|
| amqzscix.c | amqzsc -<br>TXSeries for<br>AIX, Version 5.1,<br>amqzsc -<br>TXSeries for HP-UX, Version 5.1<br>amqzsc -<br>TXSeries for Sun Solaris, Version 5.1 |
| amqzscin.c | mqmc4swi -<br>TXSeries for<br>Windows, Version 5.1                                                                                                |

*Building libraries for use with TXSeries for Multiplatforms:*

Use this information when building libraries for use with TXSeries for Multiplatforms.

*Pre-built switch load files* are shared libraries (called *DLLs* on the Windows system) that you can use with CICS programs, which require a 2-phase commit transaction by using the XA protocol. The names of these pre-built libraries are in the table Essential code for CICS applications: XA initialization routine. Sample source code is also supplied in the following directories:

*Table 9. Installation directories on Windows, UNIX and Linux operating systems*

| Platform       | Directory                                        | Source file |
|----------------|--------------------------------------------------|-------------|
| UNIX and Linux | <i>MQ_INSTALLATION_PATH</i> /samp/               | amqzscix.c  |
| Windows        | <i>MQ_INSTALLATION_PATH</i> \Tools\c\<br>Samples | amqzscin.c  |

where *MQ\_INSTALLATION\_PATH* is the directory in which you installed IBM MQ.

To build the switch load file from the sample source, follow the instructions appropriate for your operating system:

### AIX

Issue the following command:

```
export MQM_HOME=/usr/mqm
echo "amqzscix" > tmp.exp
xlc_r $MQM_HOME/samp/amqzscix.c -I/usr/lpp/cics/include -I$MQM_HOME/inc -e amqzscix -bE:tmp.exp -bM:SRE
-o amqzsc /usr/lpp/cics/lib/regxa_swxa.o -L$MQM_HOME/lib -L/usr/lpp/cics/lib -lcicsrt -lEncina
-lEncServer -lpthreads -lsarpc -lmqmcics_r -lmqmx_r -lmqzi_r -lmqmcs_r
rm tmp.exp
```

### Solaris

Issue the following command:

```
/opt/SUNWspro/bin/cc -s -l/opt/encina/include amqzscix.c -G -o amqzscix -e
CICS_XA_Init -L MQ_INSTALLATION_PATH/lib -L/opt/encina/lib
-L/opt/dce/local/lib /opt/cics/lib/reqxa_swxa.o
-lmqmcics -lmqmx -lmqzi -lmqmcs -lmqmzse -lcicsrt -lEncina -lEncSfs -ldce
```

## HP-UX

Issue the following command:

```
cc -c -s -I/opt/encina/include MQ_INSTALLATION_PATH/samp/amqzscix.c -Aa +z -o amqzscix.o ld -b
-o amqzscix amqzscix.o /opt/cics/lib/regxa_swxa.o +e CICS_XA_Init \
-L MQ_INSTALLATION_PATH/lib -L/opt/encina/lib -L/opt/cics/lib
-lmqmxa_r -lmqzi_r -lmqmc_r -lmqmzse -ldbm -lc -lm
```

## Linux platforms

Issue the following command:

```
gcc -m32 -shared -fPIC -o amqzscix amqzscix.c
\ -IMQ_INSTALLATION_PATH/inc -I CICS_INSTALLATION_PATH/include
\ -LMQ_INSTALLATION_PATH/lib -Wl,-rpath=MQ_INSTALLATION_PATH/lib
\ -Wl,-rpath=/usr/lib -Wl,-rpath-link,/usr/lib -Wl,--no-undefined
-Wl,--allow-shlib-undefined \-L CICS_LIB_PATH/regxa_swxa.o \-lpthread -ldl -lc
-shared -lmqzi_r -lmqmxa_r -lmqmcics_r -ldl -lc
```

## Windows

Follow these steps:

1. Use the `cl` command to build `amqzscin.obj` by compiling at least the following variables:

```
cl.exe -c -I EncinaPath\include -I MQ_INSTALLATION_PATH\include -Gz -LD amqzscin.c
```

2. Create a module definition file named `mqmc1415.def`, which contains the following lines:

```
LIBRARY MQMC4SWI
EXPORTS
CICS_XA_Init
```

3. Use the `lib` command to build an export file and an import library by using at least the following option:

```
lib -def:mqmc4swi.def -out:mqmc4swi.lib
```

If the `lib` command is successful then an `mqmc4swi.exp` file is also built.

4. Use the `link` command to build `mqmc4swi.dll` by using at least the following option:

```
link.exe -dll -nod -out:mqmc4swi.dll
amqzscin.obj CicsPath\lib\regxa_swxa.obj
mqmc4swi.exp mqmcics4.lib
CicsPath\lib\libcicsrt.lib
DcePath\lib\libdce.lib DcePath\lib\pthreads.lib
EncinaPath\lib\libEncina.lib
EncinaPath\lib\libEncServer.lib
msvcrt.lib kernel32.lib
```

## IBM MQ XA support and Tuxedo:

IBM MQ on Windows, UNIX and Linux systems can block Tuxedo-coordinated XA applications indefinitely in `xa_start`.

This can occur only when two or more processes coordinated by Tuxedo in a single global transaction attempt to access IBM MQ using the same transaction branch ID (XID). If Tuxedo gives each process in the global transaction a different XID to use with IBM MQ, this cannot occur.

To avoid the problem, configure each application in Tuxedo that accesses IBM MQ under a single global transaction ID (gtrid), within its own Tuxedo server group. Processes in the same server group use the same XID when accessing resource managers on behalf of a single gtrid, and are therefore vulnerable to blocking in `xa_start` in IBM MQ. Processes in different server groups use separate XIDs when accessing resource managers and so do not have to serialize their transaction work in IBM MQ.

### Enabling the CICS two-phase commit process:

To enable CICS to use a two-phase commit process to coordinate transactions that include MQI calls, add a CICS XAD resource definition stanza entry to the CICS region. Note, this topic is not applicable to z/OS.

Here is an example of adding an XAD stanza entry for IBM MQ for Windows, where <Drive> is the drive where IBM MQ is installed (for example, D:).

```
cicsadd -cxad -r<cics_region> \
 ResourceDescription="MQM XA Product Description" \
 SwitchLoadFile="<Drive>:\Program Files\IBM\WebSphere MQ\bin\mqmc4swi.dll" \
 XAOpen=<queue_manager_name>
```

For extended transactional clients, use the switch load file mqcc4swi.dll.

Here is an example of adding an XAD stanza entry for IBM MQ for UNIX and Linux systems, where *MQ\_INSTALLATION\_PATH* represents the high-level directory in which IBM MQ is installed:

```
cicsadd -cxad -r<cics_region> \
 ResourceDescription="MQM XA Product Description" \
 SwitchLoadFile=" MQ_INSTALLATION_PATH/lib/amqzsc" \
 XAOpen=<queue_manager_name>
```

For extended transactional clients, use the switch load file amqzsc.

For information about using the **cicsadd** command, see the *CICS Administration Reference*, or *CICS Administration Guide* for your platform.

Calls to IBM MQ can be included in a CICS transaction, and the IBM MQ resources will be committed or rolled back as directed by CICS. This support is not available to client applications.

You *must* issue an **MQCONN** from your CICS transaction in order to access IBM MQ resources, followed by a corresponding **MQDISC** on exit.

### Enabling CICS user exits:

A CICS user exit *point* (normally referred to as a *user exit* ) is a place in a CICS module at which CICS can transfer control to a program that you have written (a user exit *program* ), and at which CICS can resume control when your exit program has finished its work.

Before using a CICS user exit, read the *CICS Administration Guide* for your platform.

*Sample task termination exit:*

IBM MQ supplies sample source code for a CICS task termination exit.

The sample source code is in the following directories:

*Table 10. CICS task termination exits*

| Platform               | Directory                                        | Source file |
|------------------------|--------------------------------------------------|-------------|
| UNIX and Linux systems | <i>MQ_INSTALLATION_PATH</i> /samp                | amqzscgx.c  |
| Windows                | <i>MQ_INSTALLATION_PATH</i> \Tools\c\<br>Samples | amqzscgn.c  |

*MQ\_INSTALLATION\_PATH* represents the high-level directory in which IBM MQ is installed.

The build instructions for the sample task termination exit are contained in the comments in each source file.

This exit is invoked by CICS at normal and abnormal task termination (after any sync point has been taken). No recoverable work is permitted in the exit program.

These functions are only used in an IBM MQ and CICS context in which the CICS version supports the XA interface. CICS refers to these libraries as programs or user exits.

CICS has a number of user exits and amqzscgx, if used, is defined and enabled on CICS as the Task termination user exit (UE014015), that is, exit number 15.

When the task termination exit is called by CICS, CICS has already informed IBM MQ of the task's termination state and IBM MQ has taken the appropriate action (commit or rollback). All the exit does is to issue an MQDISC to clean up.

One purpose of installing and configuring your CICS system to use a task termination exit is to protect your system against some of the consequences of faulty application code. For example, if your CICS transaction ends abnormally without first calling MQDISC, and has no task termination exit installed, then you might see (within around 10 seconds) a subsequent unrecoverable failure of the CICS region. This is because IBM MQ health thread, which runs in the cicas process, will not have been posted and given time to clean up and return. The symptoms might be that the cicas process ends immediately, having written FFST reports to /var/mqm/errors or the equivalent location on Windows.

## Using the Microsoft Transaction Server (COM+)

COM+ (Microsoft Transaction Server) is designed to help users run business logic applications in a typical middle tier server.

See Features that can be used only with the primary installation on Windows for important information.

COM+ divides work up into *activities*, which are typically short independent chunks of business logic, such as *transfer funds from account A to account B*. COM+ relies heavily on object orientation and in particular on COM; loosely a COM+ activity is represented by a COM (business) object.

COM+ is an integrated part of the operating system.

COM+ provides three services to the business object administrator, removing much of the worry from the business object programmer:

- Transaction management
- Security
- Resource pooling

You usually use COM+ with front-end code that is a COM client to the objects held within COM+, and back-end services such as a database, with IBM MQ bridging between the COM+ business object and the back-end.

The front-end code can be a stand-alone program, or an Active Server Page (ASP) hosted by the Microsoft Internet Information Server (IIS). The frontend code can be on the same computer as COM+ and its business objects, with connection through COM. Alternatively, the frontend code can be on a different computer, with connection through DCOM. You can use different clients to access the same COM+ business object in different situations.

The backend code can be on the same computer as COM+ and its business objects, or on a different computer with connection through any of the IBM MQ supported protocols.

## Expiring global units of work

The queue manager can be configured to expire global units of work after a pre-configured interval of inactivity.

To enable this behavior, set the following environment variables:

- `AMQ_TRANSACTION_EXPIRY_RESCAN` =<rescan interval in milliseconds>
- `AMQ_XA_TRANSACTION_EXPIRY` =<timeout interval in milliseconds>

**Attention:** The environment variables only affect transactions that are in *Idle* state in table 6-4 of the XA Specification, which is available from *Publications* at The OPEN Group.

That is, transactions that are not associated on any application thread, but for which the external Transaction Manager software has not yet called the `xa_prepare` function call.

External transaction managers keep only a log of transactions that are prepared, committed, or rolled back. If the external transaction manager goes down for any reason, on its return it drives prepared, committed, and rolled back transactions to completion but any active transactions that have yet to be prepared become orphaned. To avoid this, set the `AMQ_XA_TRANSACTION_EXPIRY` to allow for the expected interval between an application making MQI transactional API calls and completing the transaction, having carried out transactional work on other resource managers.

To ensure a timely cleanup after the `AMQ_XA_TRANSACTION_EXPIRY` expires, set the `AMQ_TRANSACTION_EXPIRY_RESCAN` value to a lower value than the `AMQ_XA_TRANSACTION_EXPIRY` interval, ideally so that the rescan occurs more than once within the `AMQ_XA_TRANSACTION_EXPIRY` interval.

## Unit of recovery disposition

IBM MQ for z/OS provides unit of recovery dispositions. This feature allows you to configure whether the second phase of 2-phase commit transactions can be driven, for example, during recovery, when connected to another queue manager within the same queue-sharing group (QSG).


IBM MQ for z/OS V7.0.1 and later supports the unit of recovery disposition.

### Unit of recovery disposition

The unit of recovery disposition is related to an application's connection and subsequently any transactions that it starts. There are two possible unit of recovery dispositions.

- A **GROUP** unit of recovery disposition identifies that a transactional application is logically connected to the queue-sharing group and does not have an affinity to any specific queue manager. Any 2-phase commit transactions that it starts that have completed phase-1 of the commit process, that is, they are in doubt, can be inquired and resolved, when connected to any queue manager within the QSG. In a recovery scenario this means that the transaction coordinator does not have to reconnect to the same queue manager, which might be unavailable.
- A **QMGR** unit of recovery disposition identifies that an application has a direct affinity to the queue manager to which it is connected and any transactions that it starts also have this disposition.

In a recovery scenario the transaction coordinator must reconnect to the same queue manager to inquire, and resolve, any in-doubt transactions, irrespective of whether the queue manager belongs to a queue-sharing group.

 See Unit of recovery disposition for details how to implement this feature.

## Security scenarios

A set of scenarios that demonstrate applying security to different configurations.

The available security scenarios are described in the following subtopics:

### Related information:

▶ **z/OS** Setting up security on z/OS

## Security scenario: two queue managers on z/OS

In this scenario, an application uses the **MQPUT1** call to put messages to queues on queue manager QM1. Some of the messages are then forwarded to queues on QM2, using TCP and LU 6.2 channels. The TCP channels can either use SSL or not. The application could be a batch application or a CICS application, and the messages are put using the **MQPMO\_SET\_ALL\_CONTEXT** option.

This is illustrated in Figure 15.

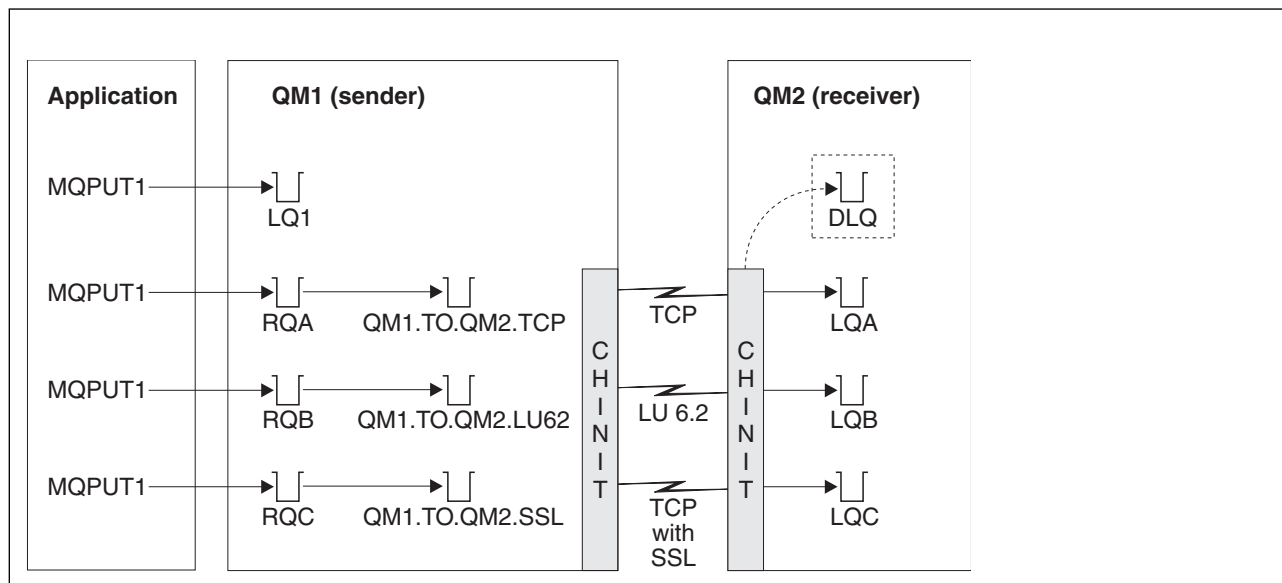


Figure 15. Example security scenario

The following assumptions are made about the queue managers:

- All the required IBM MQ definitions have been predefined or have been made through the CSQINP2 data set processed at queue manager startup.  
If they have not, you need the appropriate access authority to the commands needed to define these objects.
- All the RACF® profiles required have been defined and appropriate access authorities have been granted, before the queue manager and channel initiators started.  
If they have not, you need the appropriate authority to issue the RACF commands required to define all the profiles needed and grant the appropriate access authorities to those profiles. You also need the appropriate authority to issue the MQSC security commands to start using the new security profiles.
- All digital certificates required have been created and connected to key rings. The digital certificate sent by QM1 as part of the SSL handshake is recognized by RACF on QM2's system, either because it is also installed in that RACF profile, or because a matching Certificate Name File (CNF) filter exists.



## Related information:

▶ **z/OS** Setting up security on z/OS

## Security switch settings for two-queue-manager scenario

Switch settings and RACF profiles.

The following security switches are set for both queue managers:

- Subsystem security on
- Queue security on
- Alternate user security on
- Context security on
- Process security off
- Namelist security off
- Topic security off
- Connection security on
- Command security on
- Command resource security on

The following profiles are defined in the MQADMIN class to turn off process, namelist and topic security:

```
QM1.NO.PROCESS.CHECKS
QM1.NO.NLIST.CHECKS
QM1.NO.TOPIC.CHECKS
QM2.NO.PROCESS.CHECKS
QM2.NO.NLIST.CHECKS
QM2.NO.TOPIC.CHECKS
```

## Queue manager QM1 in two-queue-manager scenario

Queues and channels for QM1.

The following queues are defined on queue manager QM1:

**LQ1** A local queue.

**RQA** A remote queue definition, with the following attributes:

- RNAME(LQA)
- RQMNAME(QM2)
- XMITQ(QM1.TO.QM2.TCP)

**RQB** A remote queue definition, with the following attributes:

- RNAME(LQB)
- RQMNAME(QM2)
- XMITQ(QM1.TO.QM2.LU62)

**RQC** A remote queue definition, with the following attributes:

- RNAME(LQC)
- RQMNAME(QM2)
- XMITQ(QM1.TO.QM2.SSL)

**QM1.TO.QM2.TCP**

A transmission queue.

**QM1.TO.QM2.LU62**

A transmission queue.

### **QM1.TO.QM2.SSL**

A transmission queue.

The following channels are defined on QM1:

### **QM1.TO.QM2.TCP**

A sender channel definition, with the following attributes:

- CHLTYPE(SDR)
- TRPTYPE(TCP)
- XMITQ(QM1.TO.QM2.TCP)
- CONNAME(QM2TCP)

### **QM1.TO.QM2.LU62**

A sender channel definition, with the following attributes:

- CHLTYPE(SDR)
- TRPTYPE(LU62)
- XMITQ(QM1.TO.QM2.LU62)
- CONNAME(QM2LU62)

(See Security considerations for the channel initiator on z/OS for information about setting up APPC security.)

### **QM1.TO.QM2.SSL**

A sender channel definition, with the following attributes:

- CHLTYPE(SDR)
- TRPTYPE(TCP)
- XMITQ(QM1.TO.QM2.SSL)
- CONNAME(QM2TCP)
- SSLCIPH(TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA256)

## **Queue manager QM2 in two-queue-manager scenario**

Queues and channels for QM2.

The following queues have been defined on queue manager QM2:

**LQA** A local queue.

**LQB** A local queue.

**LQC** A local queue.

**DLQ** A local queue that is used as the dead-letter queue.

The following channels have been defined on QM2:

### **QM1.TO.QM2.TCP**

A receiver channel definition, with the following attributes:

- CHLTYPE(RCVR)
- TRPTYPE(TCP)
- PUTAUT(CTX)
- MCAUSER(MCATCP)

### **QM1.TO.QM2.LU62**

A receiver channel definition, with the following attributes:

- CHLTYPE(RCVR)
- TRPTYPE(LU62)

- PUTAUT(CTX)
- MCAUSER(MCALU62)

(See Security considerations for the channel initiator on z/OS for information about setting up APPC security.)

### **QM1.TO.QM2.SSL**

A receiver channel definition, with the following attributes:

- CHLTYPE(RCVR)
- TRPTYPE(TCP)
- PUTAUT(CTX)
- MCAUSER(MCASSL)
- SSLCIPH(TLS\_RSA\_WITH\_AES\_128\_CBC\_SHA256)

### **User IDs used in two-queue-manager scenario**

Explanation of the user IDs in the scenario.

The following user IDs are used:

#### **BATCHID**

Batch application (Job or TSO ID)

#### **MSGUSR**

*UserIdentifier* in MQMD (context user ID)

#### **MOVER1**

QM1 channel initiator address space user ID

#### **MOVER2**

QM2 channel initiator address space user ID

#### **MCATCP**

MCAUSER specified on the TCP/IP without SSL receiver channel definition

#### **MCALU62**

MCAUSER specified on the LU 6.2 receiver channel definition

#### **MCASSL**

MCAUSER specified on the TCP/IP with SSL receiver channel definition

#### **CICSAD1**

CICS address space ID

#### **CICSTX1**

CICS task user ID

#### **CERTID**

The user ID associated by RACF with the flowed certificate.

## Security profiles and accesses required for the two-queue-manager scenario

Security profiles and accesses for either a batch or CICS implementation of the two-queue-manager scenario.

The following table shows the security profiles that are required to enable the two-queue-manager scenario to work. Additional security profiles are also needed, depending on whether you are doing a batch or a CICS implementation of the scenario. For further information see “Security profiles required for a batch application” on page 85 and “Security profiles required for a CICS application” on page 87.

*Table 11. Security profiles for the example scenario.*

The four columns in this table show the class, the profile, the user ID, and access for the two-queue-manager scenario.

| Class   | Profile                        | User ID                      | Access  |
|---------|--------------------------------|------------------------------|---------|
| MQCONN  | QM1.CHIN                       | MOVER1                       | READ    |
| MQADMIN | QM1.RESLEVEL                   | BATCHID<br>CICSAD1<br>MOVER1 | NONE    |
| MQADMIN | QM1.CONTEXT.**                 | MOVER1                       | CONTROL |
| MQQUEUE | QM1.SYSTEM.COMMAND.INPUT       | MOVER1                       | UPDATE  |
| MQQUEUE | QM1.SYSTEM.CHANNEL.SYNCQ       | MOVER1                       | UPDATE  |
| MQQUEUE | QM1.SYSTEM.CHANNEL.INITQ       | MOVER1                       | UPDATE  |
| MQQUEUE | QM1.SYSTEM.COMMAND.REPLY.MODEL | MOVER1                       | UPDATE  |
| MQQUEUE | QM1.SYSTEM.ADMIN.CHANNEL.EVENT | MOVER1                       | UPDATE  |
| MQQUEUE | QM1.QM1.TO.QM2.TCP             | MOVER1                       | ALTER   |
| MQQUEUE | QM1.QM1.TO.QM2.LU62            | MOVER1                       | ALTER   |
| MQQUEUE | QM1.QM1.TO.QM2.SSL             | MOVER1                       | ALTER   |
| MQCONN  | QM2.CHIN                       | MOVER2                       | READ    |
| MQADMIN | QM2.RESLEVEL                   | MOVER2                       | NONE    |
| MQADMIN | QM2.CONTEXT.**                 | MOVER2                       | CONTROL |
| MQQUEUE | QM2.SYSTEM.COMMAND.INPUT       | MOVER2                       | UPDATE  |
| MQQUEUE | QM2.SYSTEM.CHANNEL.SYNCQ       | MOVER2                       | UPDATE  |
| MQQUEUE | QM2.SYSTEM.CHANNEL.INITQ       | MOVER2                       | UPDATE  |
| MQQUEUE | QM2.SYSTEM.COMMAND.REPLY.MODEL | MOVER2                       | UPDATE  |
| MQQUEUE | QM2.SYSTEM.ADMIN.CHANNEL.EVENT | MOVER2                       | UPDATE  |
| MQQUEUE | QM2.DLQ                        | MOVER2                       | UPDATE  |

## Security profiles required for a batch application:

Additional security profiles required for a batch implementation of the two-queue-manager scenario.

The batch application runs under user ID BATCHID on QM1. It connects to queue manager QM1 and puts messages to the following queues:

- LQ1
- RQA
- RQB
- RQC

It uses the MQPMO\_SET\_ALL\_CONTEXT option. The alternate user ID found in the *UserIdentifier* field of the message descriptor (MQMD) is MSGUSR.

The following profiles are required on queue manager QM1:

*Table 12. Sample security profiles for the batch application on queue manager QM1*

| Class   | Profile        | User ID | Access  |
|---------|----------------|---------|---------|
| MQCONN  | QM1.BATCH      | BATCHID | READ    |
| MQADMIN | QM1.CONTEXT.** | BATCHID | CONTROL |
| MQQUEUE | QM1.LQ1        | BATCHID | UPDATE  |
| MQQUEUE | QM1.RQA        | BATCHID | UPDATE  |
| MQQUEUE | QM1.RQB        | BATCHID | UPDATE  |
| MQQUEUE | QM1.RQC        | BATCHID | UPDATE  |

The following profiles are required on queue manager QM2 for messages put to queue RQA on queue manager QM1 (for the TCP/IP channel not using SSL):

*Table 13. Sample security profiles for queue manager QM2 using TCP/IP and not SSL*

| Class   | Profile                   | User ID       | Access  |
|---------|---------------------------|---------------|---------|
| MQADMIN | QM2.ALTERNATE.USER.MSGUSR | MCATCP MOVER2 | UPDATE  |
| MQADMIN | QM2.CONTEXT.**            | MCATCP MOVER2 | CONTROL |
| MQQUEUE | QM2.LQA                   | MOVER2 MSGUSR | UPDATE  |
| MQQUEUE | QM2.DLQ                   | MOVER2 MSGUSR | UPDATE  |

### Notes:

1. The user ID passed in the MQMD of the message is used as the user ID for the **MQPUT1** on queue manager QM2 because the receiver channel was defined with PUTAUT(CTX) and MCAUSER(MCATCP).
2. The MCAUSER field of the receiver channel definition is set to MCATCP; this user ID is used in addition to the channel initiator address space user ID for the checks carried out against the alternate user ID and context profile.
3. The MOVER2 user ID and the *UserIdentifier* in the message descriptor (MQMD) are used for the resource checks against the queue.
4. The MOVER2 and MSGUSR user IDs both need access to the dead-letter queue so that messages that cannot be put to the destination queue can be sent there.
5. Two user IDs are checked on all three checks performed because RESLEVEL is set to NONE.

The following profiles are required on queue manager QM2 for messages put to queue RQB on queue manager QM1 (for the LU 6.2 channel):

Table 14. Sample security profiles for queue manager QM2 using LU 6.2

| Class   | Profile                   | User ID        | Access  |
|---------|---------------------------|----------------|---------|
| MQADMIN | QM2.ALTERNATE.USER.MSGUSR | MCALU62 MOVER1 | UPDATE  |
| MQADMIN | QM2.CONTEXT.**            | MCALU62 MOVER1 | CONTROL |
| MQQUEUE | QM2.LQB                   | MOVER1 MSGUSR  | UPDATE  |
| MQQUEUE | QM2.DLQ                   | MOVER1 MSGUSR  | UPDATE  |

**Notes:**

1. The user ID passed in the MQMD of the message is used as the user ID for the **MQPUT1** on queue manager QM2 because the receiver channel was defined with PUTAUT(CTX) and MCAUSER(MCALU62).
2. The MCA user ID is set to the value of the MCAUSER field of the receiver channel definition (MCALU62).
3. Because LU 6.2 supports security on the communications system for the channel, the user ID received from the network is used as the channel user ID (MOVER1).
4. Two user IDs are checked on all three checks performed because RESLEVEL is set to NONE.
5. MCALU62 and MOVER1 are used for the checks performed against the alternate user ID and Context profiles, and MSGUSR and MOVER1 are used for the checks against the queue profile.
6. The MOVER1 and MSGUSR user IDs both need access to the dead-letter queue so that messages that cannot be put to the destination queue can be sent there.

The following profiles are required on queue manager QM2 for messages put to queue RQC on queue manager QM1 (for the TCP/IP channel using SSL):

Table 15. Sample security profiles for queue manager QM2 using TCP/IP and SSL

| Class   | Profile                   | User ID          | Access  |
|---------|---------------------------|------------------|---------|
| MQADMIN | QM2.ALTERNATE.USER.MSGUSR | MCASSL CERTID    | UPDATE  |
| MQADMIN | QM2.CONTEXT.**            | MCASSL CERTID    | CONTROL |
| MQQUEUE | QM2.LQC                   | CERTID MSGUSR    | UPDATE  |
| MQQUEUE | QM2.DLQ                   | CERTID<br>MSGUSR | UPDATE  |

**Notes:**

1. The user ID passed in the MQMD of the message is used as the user ID for the **MQPUT1** on queue manager QM2 because the receiver channel was defined with PUTAUT(CTX) and MCAUSER(MCASSL).
2. The MCA user ID is set to the value of the MCAUSER field of the receiver channel definition (MCASSL).
3. Because the certificate flowed by the channel from QM1 as part of the SSL handshake might be installed on QM2's system, or might match a certificate name filter on QM2's system, the user ID found during that matching is used as the channel user ID (CERTID).
4. Two user IDs are checked on all three checks performed because RESLEVEL is set to NONE.
5. MCASSL and CERTID are used for the checks performed against the alternate user ID and Context profiles, and MSGUSR and MOVER1 are used for the checks against the queue profile.
6. The CERTID and MSGUSR user IDs both need access to the dead-letter queue so that messages that cannot be put to the destination queue can be sent there.

## Security profiles required for a CICS application:

Additional security profiles required for a CICS implementation of the two-queue-manager scenario.

The CICS application uses a CICS address space user ID of CICSAD1 and a CICS task user ID of CICSTX1. The security profiles required on queue manager QM1 are different from those profiles required for the batch application. The profiles required on queue manager QM2 are the same as for the batch application.

The following profiles are required on queue manager QM1:

*Table 16. Sample security profiles for the CICS application on queue manager QM1*

| Class   | Profile        | User ID         | Access  |
|---------|----------------|-----------------|---------|
| MQCONN  | QM1.CICS       | CICSAD1         | READ    |
| MQADMIN | QM1.CONTEXT.** | CICSAD1 CICSTX1 | CONTROL |
| MQQUEUE | QM1.LQ1        | CICSAD1 CICSTX1 | UPDATE  |
| MQQUEUE | QM1.RQA        | CICSAD1 CICSTX1 | UPDATE  |
| MQQUEUE | QM1.RQB        | CICSAD1 CICSTX1 | UPDATE  |

## Security scenario: queue-sharing group on z/OS

In this scenario, an application uses the **MQPUT1** call to put messages to queues on queue manager QM1. Some of the messages are then forwarded to queues on QM2, using TCP and LU 6.2 channels. The application is a batch application, and the messages are put using the **MQPMO\_SET\_ALL\_CONTEXT** option.

This is illustrated in Figure 15 on page 80.

The following assumptions are made about the queue managers:

- All the required IBM MQ definitions have been predefined or have been made through the CSQINP2 data set processed at queue manager startup.  
If they have not, you need the appropriate access authority to the commands needed to define these objects.
- All the RACF profiles required have been defined and appropriate access authorities have been granted, before the queue manager and channel initiators started.  
If they have not, you need the appropriate authority to issue the RACF commands required to define all the profiles needed and grant the appropriate access authorities to those profiles. You also need the appropriate authority to issue the MQSC security commands to start using the new security profiles.

## Related information:

▶ **z/OS** Setting up security on z/OS

## Security switch settings for queue-sharing group scenario

Switch settings and RACF profiles.

The following security switches are set for the queue-sharing group:

- Subsystem security on
- Queue-sharing group security on
- Queue manager security off
- Queue security on
- Alternate user security on
- Context security on
- Process security off
- Namelist security off
- Topic security off
- Connection security on
- Command security on
- Command resource security on

The following profiles are defined in the MQADMIN class to turn process, namelist, topic and queue-manager level security off:

```
QSGA.NO.PROCESS.CHECKS
QSGA.NO.NLIST.CHECKS
QSGA.NO.TOPIC.CHECKS
QSGA.NO.QMGR.CHECKS
```

## Queue manager QM1 in queue-sharing group scenario

Queues and channels for QM1.

The following queues are defined on queue manager QM1:

**LQ1** A local queue.

**RQA** A remote queue definition, with the following attributes:

- RNAME(LQA)
- RQMNAME(QM2)
- XMITQ(QM1.TO.QM2.TCP)

**RQB** A remote queue definition, with the following attributes:

- RNAME(LQB)
- RQMNAME(QM2)
- XMITQ(QM1.TO.QM2.LU62)

**QM1.TO.QM2.TCP**

A transmission queue.

**QM1.TO.QM2.LU62**

A transmission queue.

The following channels are defined on QM1:



### **QM1.TO.QM2.TCP**

A sender channel definition, with the following attributes:

- CHLTYPE(SDR)
- TRPTYPE(TCP)
- XMITQ(QM1.TO.QM2.TCP)
- CONNAME(QM2TCP)

### **QM1.TO.QM2.LU62**

A sender channel definition, with the following attributes:

- CHLTYPE(SDR)
- TRPTYPE(LU62)
- XMITQ(QM1.TO.QM2.LU62)
- CONNAME(QM2LU62)

(See Security considerations for the channel initiator on z/OS for information about setting up APPC security.)

## **Queue manager QM2 in queue-sharing group scenario**

Queues and channels for QM2.

The following queues have been defined on queue manager QM2:

**LQA** A local queue.

**LQB** A local queue.

**DLQ** A local queue that is used as the dead-letter queue.

The following channels have been defined on QM2:

### **QM1.TO.QM2.TCP**

A receiver channel definition, with the following attributes:

- CHLTYPE(RCVR)
- TRPTYPE(TCP)
- PUTAUT(CTX)
- MCAUSER(MCATCP)

### **QM1.TO.QM2.LU62**

A receiver channel definition, with the following attributes:

- CHLTYPE(RCVR)
- TRPTYPE(LU62)
- PUTAUT(CTX)
- MCAUSER(MCALU62)

(See Security considerations for the channel initiator on z/OS for information about setting up APPC security.)

## User IDs used in queue-sharing group scenario

Explanation of the user IDs in the scenario.

The following user IDs are used:

### BATCHID

Batch application (Job or TSO ID)

### MSGUSR

*UserIdentifier* in MQMD (context user ID)

### MOVER1

QM1 channel initiator address space user ID

### MOVER2

QM2 channel initiator address space user ID

### MCATCP

MCAUSER specified on the TCP/IP receiver channel definition

### MCALU62

MCAUSER specified on the LU 6.2 receiver channel definition

## Security profiles and accesses required for queue-sharing group scenario

Security profiles and accesses for either a batch or CICS implementation of the queue-sharing group scenario.

The following table shows the security profiles that are required to enable the queue-sharing group scenario to work. A batch implementation of this scenario also requires the additional security profiles that are described in "Security profiles required for a batch application" on page 91.

*Table 17. Security profiles for the example scenario.*

The four columns in this table show the class, the profile, the user ID, and access for the queue-sharing group scenario.

| Class   | Profile                         | User ID                     | Access  |
|---------|---------------------------------|-----------------------------|---------|
| MQCONN  | QSGA.CHIN                       | MOVER1<br>MOVER2            | READ    |
| MQADMIN | QSGA.RESLEVEL                   | BATCHID<br>MOVER1<br>MOVER2 | NONE    |
| MQADMIN | QSGA.CONTEXT.**                 | MOVER1<br>MOVER2            | CONTROL |
| MQQUEUE | QSGA.SYSTEM.COMMAND.INPUT       | MOVER1<br>MOVER2            | UPDATE  |
| MQQUEUE | QSGA.SYSTEM.CHANNEL.SYNCQ       | MOVER1<br>MOVER             | UPDATE  |
| MQQUEUE | QSGA.SYSTEM.CHANNEL.INITQ       | MOVER1<br>MOVER2            | UPDATE  |
| MQQUEUE | QSGA.SYSTEM.COMMAND.REPLY.MODEL | MOVER1<br>MOVER2            | UPDATE  |
| MQQUEUE | QSGA.SYSTEM.ADMIN.CHANNEL.EVENT | MOVER1<br>MOVER2            | UPDATE  |
| MQQUEUE | QSGA.SYSTEM.QSG.CHANNEL.SYNCQ   | MOVER1<br>MOVER2            | UPDATE  |

Table 17. Security profiles for the example scenario (continued).

The four columns in this table show the class, the profile, the user ID, and access for the queue-sharing group scenario.

| Class   | Profile                        | User ID          | Access |
|---------|--------------------------------|------------------|--------|
| MQQUEUE | QSGA.SYSTEM.QSG.TRANSMIT.QUEUE | MOVER1<br>MOVER2 | UPDATE |
| MQQUEUE | QSGA.QM1.TO.QM2.TCP            | MOVER1           | ALTER  |
| MQQUEUE | QSGA.QM1.TO.QM2.LU62           | MOVER1           | ALTER  |
| MQQUEUE | QSGA.DLQ                       | MOVER2           | UPDATE |

### Security profiles required for a batch application:

Additional security profiles required for a batch implementation of the queue-sharing group scenario.

The batch application runs under user ID BATCHID on QM1. It connects to queue manager QM1 and puts messages to the following queues:

- LQ1
- RQA
- RQB

It uses the MQPMO\_SET\_ALL\_CONTEXT option. The user ID found in the *UserIdentifier* field of the message descriptor (MQMD) is MSGUSR.

The following profiles are required on queue manager QM1:

Table 18. Sample security profiles for the batch application on queue manager QM1

| Class   | Profile         | User ID | Access  |
|---------|-----------------|---------|---------|
| MQCONN  | QSGA.BATCH      | BATCHID | READ    |
| MQADMIN | QSGA.CONTEXT.** | BATCHID | CONTROL |
| MQQUEUE | QSGA.LQ1        | BATCHID | UPDATE  |
| MQQUEUE | QSGA.RQA        | BATCHID | UPDATE  |
| MQQUEUE | QSGA.RQB        | BATCHID | UPDATE  |

The following profiles are required on queue manager QM2 for messages put to queue RQA on queue manager QM1 (for the TCP/IP channel):

Table 19. Sample security profiles for queue manager QM2 using TCP/IP

| Class   | Profile                    | User ID       | Access  |
|---------|----------------------------|---------------|---------|
| MQADMIN | QSGA.ALTERNATE.USER.MSGUSR | MCATCP MOVER2 | UPDATE  |
| MQADMIN | QSGA.CONTEXT.**            | MCATCP MOVER2 | CONTROL |
| MQQUEUE | QSGA.LQA                   | MOVER2 MSGUSR | UPDATE  |
| MQQUEUE | QSGA.DLQ                   | MOVER2 MSGUSR | UPDATE  |

### Notes:

1. The user ID passed in the MQMD of the message is used as the user ID for the **MQPUT1** on queue manager QM2 because the receiver channel was defined with PUTAUT(CTX) and MCAUSER(MCATCP).

2. The MCAUSER field of the receiver channel definition is set to MCATCP; this user ID is used in addition to the channel initiator address space user ID for the checks carried out against the alternate user ID and context profile.
3. The MOVER2 user ID and the *UserIdentifier* in the message descriptor (MQMD) are used for the resource checks against the queue.
4. The MOVER2 and MSGUSR user IDs both need access to the dead-letter queue so that messages that cannot be put to the destination queue can be sent there.
5. Two user IDs are checked on all three checks performed because RESLEVEL is set to NONE.

The following profiles are required on queue manager QM2 for messages put to queue RQB on queue manager QM1 (for the LU 6.2 channel):

*Table 20. Sample security profiles for queue manager QM2 using LU 6.2*

| Class   | Profile                    | User ID           | Access  |
|---------|----------------------------|-------------------|---------|
| MQADMIN | QSGA.ALTERNATE.USER.MSGUSR | MCALU62<br>MOVER1 | UPDATE  |
| MQADMIN | QSGA.CONTEXT.**            | MCALU62<br>MOVER1 | CONTROL |
| MQQUEUE | QSGA.LQB                   | MOVER1<br>MSGUSR  | UPDATE  |
| MQQUEUE | QSGA.DLQ                   | MOVER1<br>MSGUSR  | UPDATE  |

**Notes:**




1. The user ID passed in the MQMD of the message is used as the user ID for the **MQPUT1** on queue manager QM2 because the receiver channel was defined with PUTAUT(CTX) and MCAUSER(MCALU62).
2. The MCA user ID is set to the value of the MCAUSER field of the receiver channel definition (MCALU62).
3. Because LU 6.2 supports security on the communications system for the channel, the user ID received from the network is used as the channel user ID (MOVER1).
4. Two user IDs are checked on all three checks performed because RESLEVEL is set to NONE.
5. MCALU62 and MOVER1 are used for the checks performed against the alternate user ID and Context profiles, and MSGUSR and MOVER1 are used for the checks against the queue profile.
6. The MOVER1 and MSGUSR user IDs both need access to the dead-letter queue so that messages that cannot be put to the destination queue can be sent there.

## Connecting two queue managers using SSL or TLS

Secure communications that use the SSL or TLS cryptographic security protocols involve setting up the communication channels and managing the digital certificates that you will use for authentication.

To set up your SSL or TLS installation you must define your channels to use SSL or TLS. You must also obtain and manage your digital certificates. On a test system, you can use self-signed certificates or certificates issued by a local certificate authority (CA). On a production system, do not use self-signed certificates.


For full information about creating and managing certificates, see the following topics:

-  Working with SSL or TLS on IBM i
-  Working with SSL or TLS on UNIX, Linux and Windows systems
-  Working with SSL or TLS on z/OS

This collection of topics introduces the tasks involved in setting up SSL communications, and provides step-by-step guidance on completing those tasks.

You might also want to test SSL or TLS client authentication, which are an optional part of the protocols. During the SSL or TLS handshake, the SSL or TLS client always obtains and validates a digital certificate from the server. With the IBM MQ implementation, the SSL or TLS server always requests a certificate from the client.

### Notes:

1. In this context, an SSL client refers to the connection initiating the handshake.
2.  When a z/OS queue manager is acting in the role of an SSL client, the queue manager sends only a certificate.

The SSL or TLS client sends a certificate only if it can find a certificate with a matching label. See Digital certificate labels for details.

The SSL or TLS server always validates the client certificate if one is sent. If the client does not send a certificate, authentication fails only if the end of the channel that is acting as the SSL or TLS server is defined with either the **SSLCAUTH** parameter set to **REQUIRED** or an **SSLPEER** parameter has a value set. For more information about connecting a queue manager anonymously, that is, when the SSL or TLS client does not send a certificate, see “Connecting two queue managers using one-way authentication” on page 98.

## Using self-signed certificates for mutual authentication of two queue managers

Follow these sample instructions to implement mutual authentication between two queue managers, using self-signed SSL or TLS certificates.

### About this task

Scenario:

- You have two queue managers, QM1 and QM2, which need to communicate securely. You require mutual authentication to be carried out between QM1 and QM2.
- You have decided to test your secure communication using self-signed certificates.

The resulting configuration looks like this:

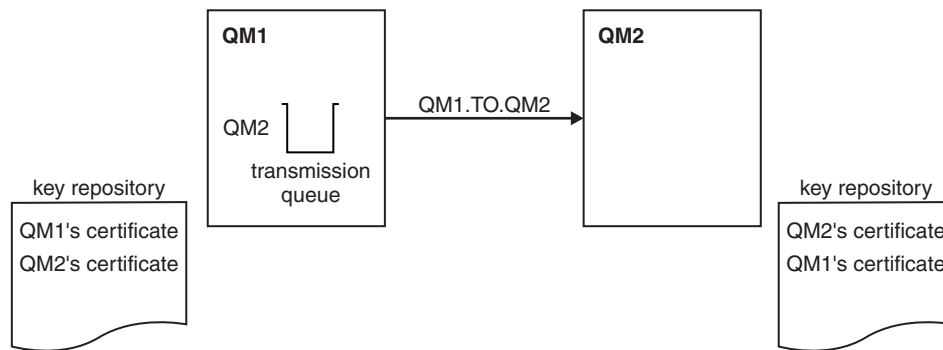







Figure 16. Configuration resulting from this task

In Figure 16, the key repository for QM1 contains the certificate for QM1 and the public certificate from QM2. The key repository for QM2 contains the certificate for QM2 and the public certificate from QM1.

### Procedure

1. Prepare the key repository on each queue manager, according to operating system:
  - On UNIX, Linux, and Windows systems.
  -  On z/OS systems.
2. Create a self-signed certificate for each queue manager:
  - On UNIX, Linux, and Windows systems.
  -  On z/OS systems.
3. Extract a copy of each certificate:
  - On UNIX, Linux, and Windows systems.
  -  On z/OS systems.
4. Transfer the public part of the QM1 certificate to the QM2 system and vice versa, using a utility such as FTP , as described in `../com.ibm.mq.sec.doc/q013210_.dita`.
5. Add the partner certificate to the key repository for each queue manager:
  - On UNIX, Linux, and Windows systems.
  -  On z/OS systems.
6. On QM1, define a sender channel and associated transmission queue, by issuing commands like the following example:

```

DEFINE CHANNEL(QM1.TO.QM2) CHLTYPE(SDR) TRPTYPE(TCP) CONNAME(QM1.MACH.COM) XMITQ(QM2)
SSLCIPH(TLS_RSA_WITH_AES_128_CBC_SHA) DESCRC('Sender channel using SSL from QM1 to QM2')

DEFINE QLOCAL(QM2) USAGE(XMITQ)

```

This example uses CipherSpec TLS\_RSA. The CipherSpecs at each end of the channel must be the same.

- On QM2, define a receiver channel, by issuing a command like the following example:

```

DEFINE CHANNEL(QM1.TO.QM2) CHLTYPE(RCVR) TRPTYPE(TCP) SSLCIPH(TLS_RSA_WITH_AES_128_CBC_SHA)
SSLCAUTH(REQUIRED) DESCRC('Receiver channel using SSL from QM1 to QM2')

```

The channel must have the same name as the sender channel you defined in step 6, and use the same CipherSpec.

- Start the channel , as described in Starting the sender channel .

## Results

Key repositories and channels are created as illustrated in Figure 16 on page 94

## What to do next

Check that the task has been completed successfully by using DISPLAY commands. If the task was successful, the resulting output is similar to that shown in the following examples.

From queue manager QM1, enter the following command:

```

DISPLAY CHS(QM1.TO.QM2) SSLPEER SSLCERTI

```

The resulting output is like the following example:

```

DISPLAY CHSTATUS(QM1.TO.QM2) SSLPEER SSLCERTI
 4 : DISPLAY CHSTATUS(QM1.TO.QM2) SSLPEER SSLCERTI
AMQ8417: Display Channel Status details.
CHANNEL(QM1.TO.QM2) CHLTYPE(SDR)
CONNAME(9.20.25.40) CURRENT
RQMNAME(QM2)
SSLCERTI("CN=QM2,OU=WebSphere MQ Development,O=IBM,ST=Hampshire,C=UK")
SSLPEER("SERIALNUMBER=4C:D0:49:D5:02:5E:02,CN=QM2,OU=WebSphere MQ Development,O=IBM,ST=Hampshire,C=UK")
STATUS(RUNNING) SUBSTATE(MQGET)
XMITQ(QM2)

```

From queue manager QM2, enter the following command:

```

DISPLAY CHS(QM1.TO.QM2) SSLPEER SSLCERTI

```

The resulting output is like the following example:

```

DISPLAY CHSTATUS(QM1.TO.QM2) SSLPEER SSLCERTI
 5 : DISPLAY CHSTATUS(QM1.TO.QM2) SSLPEER SSLCERTI
AMQ8417: Display Channel Status details.
CHANNEL(QM2.TO.QM1) CHLTYPE(RCVR)
CONNAME(9.20.35.92) CURRENT
RQMNAME(QM1)
SSLCERTI("CN=QM1,OU=WebSphere MQ Development,O=IBM,ST=Hampshire,C=UK")
SSLPEER("SERIALNUMBER=4C:D0:49:D5:02:5F:38,CN=QM1,OU=WebSphere MQ Development,O=IBM,ST=Hampshire,C=UK")
STATUS(RUNNING) SUBSTATE(RECEIVE)
XMITQ()

```

In each case, the value of SSLPEER must match that of the DN in the partner certificate that was created in Step 2. The issuers name matches the peer name because the certificate is self-signed .

SSLPEER is optional. If it is specified, its value must be set so that the DN in the partner certificate (created in step 2) is allowed. For more information about the use of SSLPEER, see IBM MQ rules for SSLPEER values.

## Using CA-signed certificates for mutual authentication of two queue managers

Follow these sample instructions to implement mutual authentication between two queue managers, using CA-signed SSL or TLS certificates.

### About this task

Scenario:

- You have two queue managers called QMA and QMB, which need to communicate securely. You require mutual authentication to be carried out between QMA and QMB.
- In the future you are planning to use this network in a production environment, and therefore you have decided to use CA-signed certificates from the beginning.

The resulting configuration looks like this:

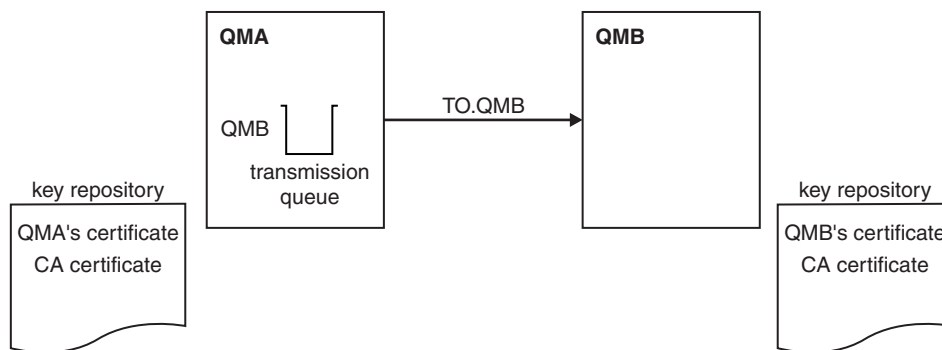






Figure 17. Configuration resulting from this task



In Figure 17, the key repository for QMA contains QMA's certificate and the CA certificate. The key repository for QMB contains QMB's certificate and the CA certificate. In this example both QMA's certificate and QMB's certificate were issued by the same CA. If QMA's certificate and QMB's certificate were issued by different CAs then the key repositories for QMA and QMB must contain both CA certificates.

### Procedure



1. Prepare the key repository on each queue manager, according to operating system:
  -  On IBM i systems.
  - On UNIX, Linux, and Windows systems.
  -  On z/OS systems.
2. Request a CA-signed certificate for each queue manager. You might use different CAs for the two queue managers.
  -  On IBM i systems.
  - On UNIX, Linux, and Windows systems.
  -  On z/OS systems.



3. Add the Certificate Authority certificate to the key repository for each queue manager: If the Queue managers are using different Certificate Authorities then the CA certificate for each Certificate Authority must be added to both key repositories.

-  Do not perform this step on IBM i systems.
- On UNIX, Linux, and Windows systems.
-  On z/OS systems.

4. Add the CA-signed certificate to the key repository for each queue manager:

-  On IBM i systems.
- On UNIX, Linux, and Windows systems.
-  On z/OS systems.

5. On QMA, define a sender channel and associated transmission queue by issuing commands like the following example:

```
DEFINE CHANNEL(TO.QMB) CHLTYPE(SDR) TRPTYPE(TCP)
CONNAME(QMB.MACH.COM) XMITQ(QMB) SSLCIPH(TLS_RSA_WITH_AES_128_CBC_SHA)
DESCR('Sender channel using TLS from QMA to QMB')
```

```
DEFINE QLOCAL(QMB) USAGE(XMITQ)
```



This example uses CipherSpec RC4\_MD5. The CipherSpecs at each end of the channel must be the same.

6. On QMB, define a receiver channel by issuing a command like the following example:

```
DEFINE CHANNEL(TO.QMB) CHLTYPE(RCVR) TRPTYPE(TCP)
SSLCIPH(TLS_RSA_WITH_AES_128_CBC_SHA) SSLCAUTH(REQUIRED)
DESCR('Receiver channel using TLS to QMB')
```

The channel must have the same name as the sender channel you defined in step 6, and use the same CipherSpec.

7. Start the channel:  

-  On IBM i systems.
- On UNIX, Linux, and Windows systems.
-  On z/OS systems.

## Results

Key repositories and channels are created as illustrated in Figure 17 on page 96.

## What to do next

Check that the task has been completed successfully by using DISPLAY commands. If the task was successful, the resulting output is like that shown in the following examples.

From queue manager QMA, enter the following command:

```
DISPLAY CHS(TO.QMB) SSLPEER SSLCERTI
```

The resulting output is like the following example:

```
DISPLAY CHSTATUS(TO.QMB) SSLPEER SSLCERTI
 4 : DISPLAY CHSTATUS(TO.QMB) SSLPEER SSLCERTI
AMQ8417: Display Channel Status details.
CHANNEL(TO.QMB) CHLTYPE(SDR)
CONNAME(9.20.25.40) CURRENT
RQMNAME(QMB)
```

```

SSLCERTI("CN=WebSphere MQ CA,OU=WebSphere MQ Devt,O=IBM,ST=Hampshire,C=UK")
SSLPEER("SERIALNUMBER=4C:D0:49:D5:02:5F:38,CN=QMB,OU=WebSphere MQ Development,O=IBM,ST=Hampshire,C=UK")
STATUS(RUNNING) SUBSTATE(MQGET)
XMITQ(QMB)

```

From the queue manager QMB, enter the following command:

```
DISPLAY CHS(TO.QMB) SSLPEER SSLCERTI
```

The resulting output is like the following example:

```

DISPLAY CHSTATUS(TO.QMB) SSLPEER SSLCERTI
 5 : DISPLAY CHSTATUS(TO.QMB) SSLPEER SSLCERTI
AMQ8417: Display Channel Status details.
CHANNEL(TO.QMB) CHLTYPE(RCVR)
CONNNAME(9.20.35.92) CURRENT
RQMNAME(QMA)
SSLCERTI("CN=WebSphere MQ CA,OU=WebSphere MQ Devt,O=IBM,ST=Hampshire,C=UK")
SSLPEER("SERIALNUMBER=4C:D0:49:D5:02:5F:38,CN=QMA,OU=WebSphere MQ Development,O=IBM,ST=Hampshire,C=UK")
STATUS(RUNNING) SUBSTATE(RECEIVE)
XMITQ()

```

In each case, the value of SSLPEER must match that of the Distinguished Name (DN) in the partner certificate that was created in Step 2. The issuer name matches the subject DN of the CA certificate that signed the personal certificate added in Step 4.

## Connecting two queue managers using one-way authentication

Follow these sample instructions to modify a system with mutual authentication to allow a queue manager to connect using one-way authentication to another; that is, when the SSL or TLS client does not send a certificate.

### About this task

Scenario:

- Your two queue managers (QM1 and QM2) have been set up as in “Using CA-signed certificates for mutual authentication of two queue managers” on page 96.
- You want to change QM1 so that it connects using one-way authentication to QM2.

The resulting configuration looks like this:

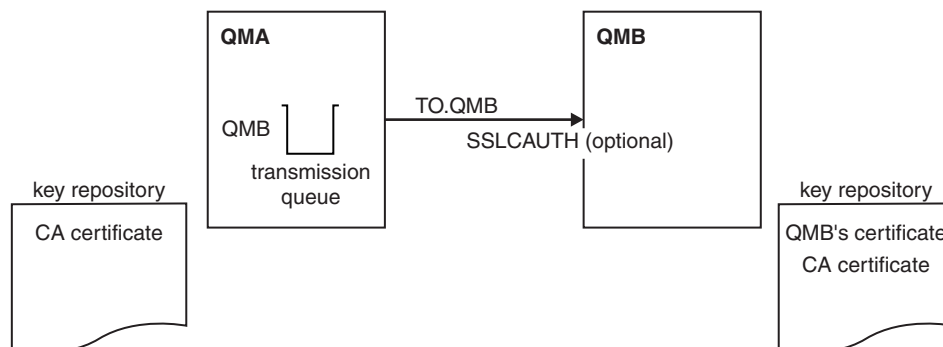







Figure 18. Queue managers allowing one-way authentication

## Procedure

1. Remove the personal certificate of QM1 from its key repository:

-  Removing a certificate on IBM i systems.
-    Removing a certificate on UNIX, Linux, and Windows systems.
-  Removing a certificate on z/OS systems. Perform this step twice, to remove both the personal certificate for QMA, and the default certificate.

For details of how certificates are labeled, see Digital certificate labels.

2. Optional: On QM1, if any SSL or TLS channels have run previously, refresh the SSL or TLS environment , as described in Refreshing the SSL or TLS environment .
3. Allow anonymous connections on the receiver , as described in Allowing anonymous connections on a receiver channel .

Key repositories and channels are changed as illustrated in Figure 18 on page 98

4. If the sender channel was not running, start it.

**Note:** If the sender channel was running and you issued the REFRESH SECURITY TYPE(SSL) command (in step 2), the channel restarts automatically.

At the server end of the channel, the presence of the peer name parameter value on the channel status display indicates that a client certificate has flowed.

5. Verify that the task has been completed successfully by issuing some DISPLAY commands. If the task was successful, the resulting output is similar to that shown in the following examples:

- From the QM1 queue manager, enter the following command:

```
DISPLAY CHS(TO.QM2) SSLPEER SSLCERTI
```

The resulting output will be similar to the following example:

```
DISPLAY CHSTATUS(TO.QMB) SSLPEER SSLCERTI
 4 : DISPLAY CHSTATUS(TO.QMB) SSLPEER
AMQ8417: Display Channel Status details.
CHANNEL(TO.QM2) CHLTYPE(SDR)
CONNAME(9.20.25.40) CURRENT
RQMNAME(QM2)
SSLCERTI("CN=WebSphere MQ CA,OU=WebSphere MQ Devt,O=IBM,ST=Hampshire,C=UK")
SSLPEER("SERIALNUMBER=4C:D0:49:D5:02:5F:38,CN=QMB,OU=WebSphere MQ Development,O=IBM,ST=Hampshire,C=UK")
STATUS(RUNNING) SUBSTATE(MQGET)
XMITQ(QM2)
```

- From the QM2 queue manager, enter the following command:

```
DISPLAY CHS(TO.QM2) SSLPEER SSLCERTI
```

The resulting output will be similar to the following example:

```
DISPLAY CHSTATUS(TO.QM2) SSLPEER SSLCERTI
 5 : DISPLAY CHSTATUS(TO.QM2) SSLPEER SSLCERTI
AMQ8417: Display Channel Status details.
CHANNEL(TO.QM2) CHLTYPE(RCVR)
CONNAME(9.20.35.92) CURRENT
RQMNAME(QMA) SSLCERTI()
SSLPEER() STATUS(RUNNING)
SUBSTATE(RECEIVE) XMITQ()
```




On QM2, the SSLPEER field is empty, showing that QM1 did not send a certificate. On QM1, the value of SSLPEER matches that of the DN in QM2's personal certificate.

## Connecting a client to a queue manager securely

Secure communications that use the SSL or TLS cryptographic security protocols involve setting up the communication channels and managing the digital certificates that you will use for authentication.


To set up your SSL or TLS installation you must define your channels to use SSL or TLS. You must also obtain and manage your digital certificates. On a test system, you can use self-signed certificates or certificates issued by a local certificate authority (CA). On a production system, do not use self-signed certificates.

For full information about creating and managing certificates, see the following topics:

-  Working with SSL or TLS on IBM i
-  Working with SSL or TLS on UNIX, Linux and Windows systems
-  Working with SSL or TLS on z/OS

This collection of topics introduces the tasks involved in setting up SSL communications, and provides step-by-step guidance on completing those tasks.

You might also want to test SSL or TLS client authentication, which are an optional part of the protocols. During the SSL or TLS handshake, the SSL or TLS client always obtains and validates a digital certificate from the server. With the IBM MQ implementation, the SSL or TLS server always requests a certificate from the client.

On  IBM i, UNIX, Linux, and Windows systems, the SSL or TLS client sends a certificate only if it has one labeled in the correct IBM MQ format, which is either `ibmwebspheremq` followed by your logon user ID in lowercase, or the value of the **CERTLABL** attribute. See Digital certificate labels.

The SSL or TLS server always validates the client certificate if one is sent. If the client does not send a certificate, authentication fails only if the end of the channel that is acting as the SSL or TLS server is defined with either the **SSLCAUTH** parameter set to **REQUIRED** or an **SSLPEER** parameter value set. For more information about connecting a queue manager anonymously, see “Connecting a client to a queue manager anonymously” on page 104.

### Related concepts:

SSL/TLS CipherSpecs and CipherSuites in IBM MQ classes for Java

SSL/TLS CipherSpecs and CipherSuites in IBM MQ classes for JMS

### Related information:

Using certificates for the managed .NET client

## Using self-signed certificates for mutual authentication of a client and queue manager

Follow these sample instructions to implement mutual authentication between a client and a queue manager, by using self-signed SSL or TLS certificates.

### About this task

Scenario:

- You have a client, C1, and a queue manager, QM1, which need to communicate securely. You require mutual authentication to be carried out between C1 and QM1.
- You have decided to test your secure communication by using self-signed certificates.

DCM on IBM i does not support self-signed certificates, so this task is not applicable on IBM i systems.

The resulting configuration looks like this:

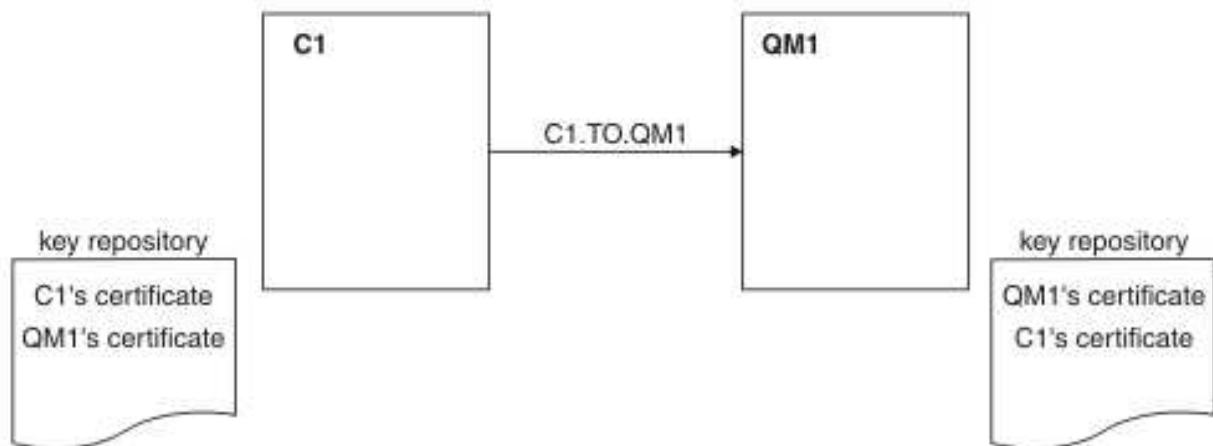


Figure 19. Configuration resulting from this task

In Figure 19, the key repository for QM1 contains the certificate for QM1 and the public certificate from C1. The key repository for C1 contains the certificate for C1 and the public certificate from QM1.

## Procedure

1. Prepare the key repository on the client and queue manager, according to operating system:
  - On UNIX, Linux, and Windows systems.
  - **z/OS** On z/OS systems (queue manager only).
2. Create self-signed certificates for the client and queue manager:
  - On UNIX, Linux, and Windows systems.
  - **z/OS** On z/OS systems (queue manager only).
3. Extract a copy of each certificate:
  - On UNIX, Linux, and Windows systems.
  - **z/OS** On z/OS systems.
4. Transfer the public part of the C1 certificate to the QM1 system and vice versa, using a utility such as FTP **z/OS**, as described in Exchanging self-signed certificates.
5. Add the partner certificate to the key repository for the client and queue manager:
  - On UNIX, Linux, and Windows systems.
  - **z/OS** On z/OS systems.
6. Define a client-connection channel in either of the following ways:
  - Using the MQCONN call with the MQSCO structure on C1, as described in Creating a client-connection channel on the IBM MQ MQI client.
  - Using a client channel definition table, as described in Creating server-connection and client-connection definitions on the server.
7. On QM1, define a server-connection channel, by issuing a command like the following example:
 

```
DEFINE CHANNEL(C1.TO.QM1) CHLTYPE(SVRCONN) TRPTYPE(TCP) SSLCIPH(TLS_RSA_WITH_AES_128_CBC_SHA)
SSLCAUTH(REQUIRED) DESCR('Receiver channel using SSL from C1 to QM1')
```

The channel must have the same name as the client-connection channel you defined in step 6, and use the same CipherSpec.

## Results

Key repositories and channels are created as illustrated in Figure 19 on page 101.

## What to do next

Check that the task has been completed successfully by using DISPLAY commands. If the task was successful, the resulting output is similar to that shown in the following example.

From queue manager QM1, enter the following command:

```
DISPLAY CHSTATUS(C1.TO.QM1) SSLPEER SSLCERTI
```

The resulting output is like the following example:

```
DISPLAY CHSTATUS(C1.TO.QM1) SSLPEER SSLCERTI
 5 : DISPLAY CHSTATUS(C1.TO.QM1) SSLPEER SSLCERTI
AMQ8417: Display Channel Status details.
CHANNEL(C1.TO.QM1) CHLTYPE(SVRCONN)
CONNNAME(9.20.35.92) CURRENT
SSLCERTI("CN=QM1,OU=WebSphere MQ Development,O=IBM,ST=Hampshire,C=UK")
SSLPEER("SERIALNUMBER=4C:D0:49:D5:02:5E:02,CN=QM2,OU=WebSphere MQ Development,O=IBM,ST=Hampshire,C=UK")
STATUS(RUNNING) SUBSTATE(RECEIVE)
```

It is optional to set the SSLPEER filter attribute of the channel definitions. If the channel definition SSLPEER is set, its value must match the subject DN in the partner certificate that was created in Step 2. After a successful connection, the SSLPEER field in the DISPLAY CHSTATUS output shows the subject DN of the remote client certificate.

## Using CA-signed certificates for mutual authentication of a client and queue manager

Follow these sample instructions to implement mutual authentication between a client and a queue manager, by using CA-signed SSL or TLS certificates.

### About this task

Scenario:

- You have a client, C1, and a queue manager, QM1, which need to communicate securely. You require mutual authentication to be carried out between C1 and QM1.
- In the future you are planning to use this network in a production environment, and therefore you have decided to use CA-signed certificates from the beginning.

The resulting configuration looks like this:

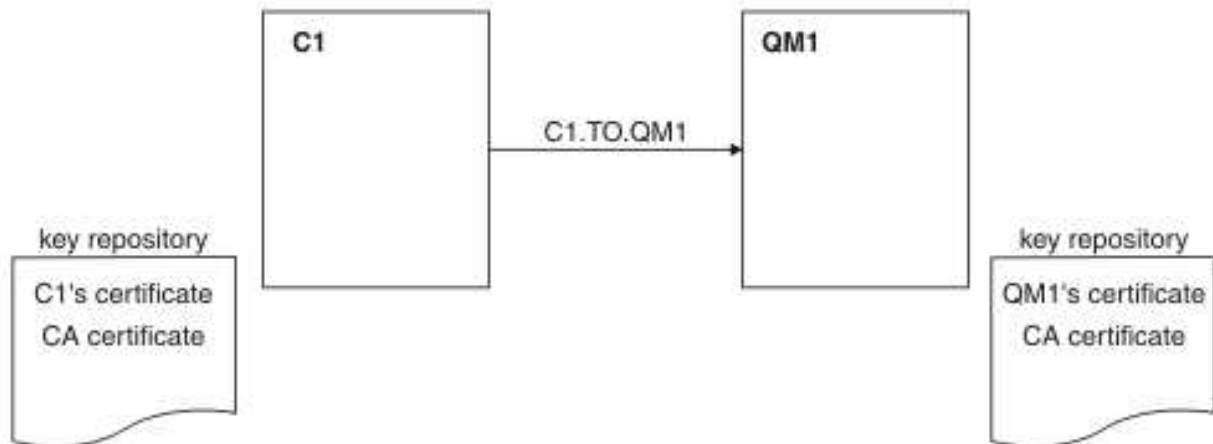


Figure 20. Configuration resulting from this task

In Figure 20, the key repository for C1 contains certificate for C1 and the CA certificate. The key repository for QM1 contains the certificate for QM1 and the CA certificate. In this example both C1's certificate and QM1's certificate were issued by the same CA. If C1's certificate and QM1's certificate were issued by different CAs then the key repositories for C1 and QM1 must contain both CA certificates.

## Procedure

1. Prepare the key repository on the client and queue manager, according to operating system:
  - **IBM i** On IBM i systems.
  - On UNIX, Linux, and Windows systems.
  - **z/OS** On z/OS systems (queue manager only).
2. Request a CA-signed certificate for the client and queue manager. You might use different CAs for the client and queue manager.
  - **IBM i** On IBM i systems.
  - On UNIX, Linux, and Windows systems.
  - **z/OS** On z/OS systems (queue manager only).
3. Add the certificate authority certificate to the key repository for the client and queue manager. If the client and queue manager are using different Certificate Authorities then the CA certificate for each Certificate Authority must be added to both key repositories.
  - **IBM i** Do not perform this step on IBM i systems.
  - On UNIX, Linux, and Windows systems.
  - **z/OS** On z/OS systems (queue manager only).
4. Add the CA-signed certificate to the key repository for the client and queue manager:
  - **IBM i** On IBM i systems.
  - On UNIX, Linux, and Windows systems.
  - **z/OS** On z/OS systems (queue manager only).
5. Define a client-connection channel in either of the following ways:
  - Using the MQCONNX call with the MQSCO structure on C1, as described in Creating a client-connection channel on the IBM MQ MQI client.

- Using a client channel definition table, as described in Creating server-connection and client-connection definitions on the server.
6. On QM1, define a server-connection channel by issuing a command like the following example:

```
DEFINE CHANNEL(C1.TO.QM1) CHLTYPE(SVRCONN) TRPTYPE(TCP)
SSLCIPH(TLS_RSA_WITH_AES_128_CBC_SHA) SSLCAUTH(REQUIRED)
DESCR('Receiver channel using SSL from C1 to QM1')
```

The channel must have the same name as the client-connection channel you defined in step 6, and use the same CipherSpec.

## Results

Key repositories and channels are created as illustrated in Figure 20 on page 103.

## What to do next

Check that the task has been completed successfully by using DISPLAY commands. If the task was successful, the resulting output is like that shown in the following example.

From the queue manager QM1, enter the following command:

```
DISPLAY CHSTATUS(TO.QMB) SSLPEER SSLCERTI
```

The resulting output is like the following example:

```
DISPLAY CHSTATUS(C1.TO.QM1) SSLPEER SSLCERTI
 5 : DISPLAY CHSTATUS(C1.TO.QM1) SSLPEER SSLCERTI
AMQ8417: Display Channel Status details.
CHANNEL(C1.TO.QM1) CHLTYPE(SVRCONN)
CONNAME(9.20.35.92) CURRENT
SSLCERTI("CN=WebSphere MQ CA,OU=WebSphere MQ Devt,0=IBM,ST=Hampshire,C=UK")
SSLPEER("SERIALNUMBER=4C:D0:49:D5:02:5F:38,CN=QMA,OU=WebSphere MQ Development,0=IBM,ST=Hampshire,C=UK")
STATUS(RUNNING) SUBSTATE(RECEIVE)
```

The SSLPEER field in the DISPLAY CHSTATUS output shows the subject DN of the remote client certificate that was created in Step 2. The issuer name matches the subject DN of the CA certificate that signed the personal certificate added in Step 4.

## Connecting a client to a queue manager anonymously

Follow these sample instructions to modify a system with mutual authentication to allow a queue manager to connect anonymously to another.

### About this task

Scenario:

- Your queue manager and client (QM1 and C1) have been set up as in “Using CA-signed certificates for mutual authentication of a client and queue manager” on page 102.
- You want to change C1 so that it connects anonymously to QM1.

The resulting configuration looks like this:



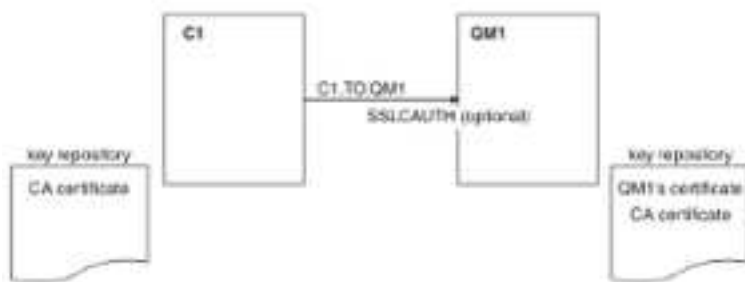


Figure 21. Client and queue manager allowing anonymous connection

## Procedure

1. Remove the personal certificate from the key repository for C1, according to operating system:
  - **IBM i** IBM i systems.
  - **UNIX** **Linux** **Windows** UNIX, Linux, and Windows systems.

The certificate label is either `ibmwebspheremq` followed by your logon user ID in lowercase, or the value of the **CERTLABL** attribute. See Digital certificate labels.

2. Restart the client application, or cause the client application to close and reopen all SSL or TLS connections.
3. Allow anonymous connections on the queue manager, by issuing the following command:  
`ALTER CHANNEL(C1.TO.QM1) CHLTYPE(SVRCONN) SSLCAUTH(OPTIONAL)`

## Results

Key repositories and channels are changed as illustrated in Figure 21

## What to do next

At the server end of the channel, the presence of the peer name parameter value on the channel status display indicates that a client certificate has flowed.

Verify that the task has been completed successfully by issuing some `DISPLAY` commands. If the task was successful, the resulting output is similar to that shown in the following example:

From queue manager QM1, enter the following command:

```
DISPLAY CHSTATUS(C1.TO.QM1) SSLPEER SSLCERTI
```

The resulting output will be similar to the following example:

```
DISPLAY CHSTATUS(C1.TO.QM1) SSLPEER SSLCERTI
 5 : DISPLAY CHSTATUS(C1.TO.QM1) SSLPEER SSLCERTI
AMQ8417: Display Channel Status details.
CHANNEL(C1.TO.QM1) CHLTYPE(SVRCONN)
CONNAME(9.20.35.92) CURRENT
SSLCERTI() SSLPEER()
STATUS(RUNNING) SUBSTATE(RECEIVE)
```

The `SSLCERTI` and `SSLPEER` fields are empty, showing that C1 did not send a certificate.

---

## **File transfer scenario**

A scenario that demonstrates use of the Managed File Transfer capability.

The available file transfer scenario is described in the IBM MQ Managed File Transfer documentation, starting with Scenario overview.

---

# Index

## A

- administration
  - for database managers 64
- AIX operating system
  - Db2 switch load file, creating 55
  - Informix switch load file, creating 59
  - Oracle switch load file, creating 57
  - sybswit, creating the Sybase switch load file 62
- amqzsc (XA switch load module) 75
- amqzsc21 (XA switch load module) 75
- amqzsc (XA switch load module) 75

## C

- CICS
  - enabling the two-phase commit process 77
  - requirements, two-phase commit process 74
  - switch load files 75
  - task termination exit, UE014015 77
  - user exits, enabling 77
  - XA-compliance 74
- configuring
  - database products 50
  - Db2 54
  - Informix 58
  - multiple databases 63
  - Oracle 56
  - Sybase 61
- configuring your system for database coordination 50

## D

- database managers
  - changing the configuration information 68
  - connections to 50
  - coordination
    - application program fails 48
    - configuring database product 50
    - configuring for 50
    - database fails 47
    - installing database product 50
    - introduction 47
    - restrictions 48
    - switch function pointers 49
    - switch load files 49
- database manager instances, removing 68
- dspmqrn command, checking outstanding units of work 65
- in-doubt units of work 65
- multiple databases, configuring 63
- restrictions, database coordination support 48

- database managers (*continued*)
  - rsvmqtrn command, explicit resynchronization of units of work 67
  - security considerations 64
  - server fails 47
  - switch load files, creating 51
  - sync point coordination 71
- database products
  - configuring 50
  - installing 50
- DB2
  - adding XAResourceManager stanza 54
  - configuring 54
  - Db2 configuration parameters, changing 54
  - environment variable settings 54
  - explicit resynchronization of units of work 67
  - security considerations 64
  - switch load file, creating 54
  - switch load file, creating on UNIX 55
  - switch load file, creating on Windows systems 54

## E

- environment variables
  - DB2INSTANCE 54
  - INFORMIXDIR 58
  - INFORMIXSERVER 58
  - ONCONFIG 58
  - ORACLE\_HOME, Oracle 56
  - ORACLE\_SID, Oracle 56
  - SYBASE\_OCS, Sybase 61
  - SYBASE, Sybase 61
- example
  - queue-sharing group security scenario 87

## F

- files
  - XA switch load files 72

## G

- global units of work
  - adding XAResourcemanager stanza to qm.ini, Informix 58
  - adding XAResourcemanager stanza to qm.ini, Oracle 56
  - adding XAResourceManager stanza, Db2 54
  - definition of 45

## H

- HP-UX
  - sybswit, creating the Sybase switch load file 62

## I

- IBM MQ for AIX
  - amqzsc 75
  - amqzsc21 75
  - amqzsc (XA switch load module) 75
  - XA switch load module 75
- IBM MQ for HP-UX
  - amqzsc 75
  - XA switch load module 75
- indoubt transactions
  - database managers 65
- Informix
  - configuration 58
  - database, creation 58
  - environment variable settings, checking 58
  - INFORMIXDIR, environment variable 58
  - ONCONFIG, environment variable 58
  - switch load file, creating 58
  - switch load file, creating on UNIX 59
  - switch load file, creating on Windows systems 58
  - XAResourceManager stanza, adding to qm.ini 58
- installing
  - database products 50
- Isolation level
  - AMQ\_REVERSE\_COMMIT\_ORDER 46
- Isolation
  - introduction 46

## L

- local unit of work
  - definition of 45

## M

- MTS (Microsoft Transaction Server)
  - introduction 78
  - services 78

## O

- Oracle
  - configuration parameters, changing 56
  - configuring 56
  - environment variable settings, checking 56

Oracle (*continued*)  
ORACLE\_HOME, environment variable 56  
ORACLE\_SID, environment variable 56  
security considerations 64  
switch load file, creating 56  
switch load file, creating on UNIX 57  
switch load file, creating on Windows systems 56  
XAResourceManager stanza, adding to qm.ini 56

## Q

queue-sharing groups  
example security scenario 87

## R

restrictions  
database coordination support 48

## S

sample  
queue-sharing group security scenario 87  
scenario  
security  
two queue managers 80  
security  
considerations for transactional support 64  
example queue-sharing group scenario 87  
scenarios  
two queue managers 80  
shared queues  
example security scenario 87  
Solaris  
sybswit, creating the Sybase switch load file 62  
SSL  
setting up  
introduction 93, 100  
stanzas  
CICS XAD resource definition stanza 77  
switch load files  
introduction 49  
switch load files, creating 51  
Sybase  
configuring 61  
environment variable settings, checking 61  
security considerations 64  
switch load file, creating 61  
Sybase XA support, enabling 61  
SYBASE\_OCS, environment variable 61  
SYBASE, environment variable 61  
sybswit, creating the switch load file on UNIX 62  
sybswit, creating the switch load file on Windows systems 61

Sybase (*continued*)  
XAResourceManager stanza, adding 61  
sync point coordination 71  
MQ 72

## T

task termination exit, CICS 77  
transactional support  
MQ XA switch structure 72  
sync point coordination 71  
transactional support 44  
transactions  
security considerations 64  
Tuxedo  
MQ and XA support 76

## U

unit of recovery disposition  
GROUP 79  
QMGR 79  
units of recovery disposition  
introduction 79  
units of work  
explicit resynchronization of (rsvmqtrn command) 67  
global 45  
introduction 45  
local 45  
mixed outcomes 67  
UNIX operating system  
Db2 switch load file, creating 55  
Informix switch load file, creating 59  
Oracle switch load file, creating 57  
switch load structures, library names 72  
sybswit, creating the Sybase switch load file 62  
user exits  
CICS task termination exit, UE014015 77  
enabling CICS user exits 77

## W

Windows operating system  
adding XAResourceManager information for Db2 54  
db2swit.dll, creating 54  
Informix switch load file, creating 58  
oraswit.dll, creating 56  
switch load structures, library names 72  
sybswit, creating the Sybase switch load file 61

## X

XA support  
building libraries for CICS 75  
CICS switch load files 75  
MQ with Tuxedo 76  
switch load module 75

XA switch load files  
description of 72  
XAD resource definition stanza, CICS 77

---

## Notices

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

IBM may not offer the products, services, or features discussed in this document 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 document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

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

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

Intellectual Property Licensing  
Legal and Intellectual Property Law  
IBM Japan, Ltd.  
19-21, Nihonbashi-Hakozakicho, Chuo-ku  
Tokyo 103-8510, 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 publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication 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 Corporation  
Software Interoperability Coordinator, Department 49XA  
3605 Highway 52 N  
Rochester, MN 55901  
U.S.A.

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 Program 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 contains 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.

If you are viewing this information softcopy, the photographs and color illustrations may not appear.

---

## Programming interface information

Programming interface information, if provided, is intended to help you create application software for use with this program.

This book contains information on intended programming interfaces that allow the customer to write programs to obtain the services of WebSphere MQ.

However, this information may also contain diagnosis, modification, and tuning information. Diagnosis, modification and tuning information is provided to help you debug your application software.

**Important:** Do not use this diagnosis, modification, and tuning information as a programming interface because it is subject to change.

---

## Trademarks

IBM, the IBM logo, [ibm.com](http://www.ibm.com)<sup>®</sup>, are trademarks of IBM Corporation, registered in many jurisdictions worldwide. A current list of IBM trademarks is available on the Web at “Copyright and trademark information”[www.ibm.com/legal/copytrade.shtml](http://www.ibm.com/legal/copytrade.shtml). Other product and service names might be trademarks of IBM or other companies.

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.

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

This product includes software developed by the Eclipse Project (<http://www.eclipse.org/>).

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.





---

## **Sending your comments to IBM**

We appreciate your input on this publication. Feel free to comment on the clarity, accuracy, and completeness of the information or give us any other feedback that you might have.

Use one of the following methods to send us your comments:

- Send an email to [ibmkc@us.ibm.com](mailto:ibmkc@us.ibm.com)
- Use the form on the web here: [www.ibm.com/software/data/rcf/](http://www.ibm.com/software/data/rcf/)

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate, without incurring any obligation to you.

Include the following information:

- Your name and address
- Your email address
- Your telephone or fax number
- The publication title and order number
- The topic and page number related to your comment
- The text of your comment

IBM or any other organizations will only use the personal information that you supply to contact you about the issues that you submit.

Thank you for your participation.





