7.5

Configuring IBM WebSphere MQ

Note

Before using this information and the product it supports, read the information in “Notices” on page 439.

This edition applies to version 7 release 5 of IBM® 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.

Contents

Configuring

Create one or more queue managers on one or more computers, and configure them on your development, test, and production systems to process messages that contain your business data.

Before you configure IBM WebSphere MQ, read about the IBM WebSphere MQ concepts in IBM WebSphere MQ Technical overview. Read about how to plan your IBM WebSphere MQ environment in Planning .

There are a number of different methods that you can use to create, configure, and administer your queue managers and their related resources in IBM WebSphere MQ. These methods include command line interfaces, a graphical user interface, and an administration API. For more information about these interfaces, see Administering IBM WebSphere MQ .

For instructions on how to create, start, stop, and delete a queue manager, see “Creating and managing queue managers” on page 18.

For information about how to create the components required to connect your IBM WebSphere MQ installations and applications together, see “Connecting applications using distributed queuing” on page 27.

For instructions on how to connect your clients to an IBM WebSphere MQ server by using different methods, see “Configuring connections between the client and server” on page 96.

For instructions on how to configure a queue manager cluster, see “Configuring a queue manager cluster” on page 154.

You can change the behavior of IBM WebSphere MQ or a queue manager by changing configuration information. For more information, see “Changing IBM WebSphere MQ and queue manager configuration information” on page 407. In general, you do not need to restart a queue manager for any configuration changes to take effect, except for when stated in this product documentation.

Related concepts

WebSphere MQ technical overview

Related tasks

Administering local WebSphere MQ objects

Administering remote WebSphere MQ objects

Planning

Configuring multiple installations on UNIX, Linux, and Windows

When using multiple installations on the same system, you must configure the installations and queue managers.

This information applies to UNIX, Linux®, and Windows.

Use the information in the following links to configure your installations:

- “Changing the primary installation” on page 14
- “Associating a queue manager with an installation” on page 16
- “Connecting applications in a multiple installation environment” on page 6

Related concepts

Multiple installations

Related tasks

Choosing a primary installation

Choosing an installation name

Connecting applications in a multiple installation environment

On UNIX, Linux, and Windows systems, if IBM WebSphere MQ Version 7.1, or later, libraries are loaded, IBM WebSphere MQ automatically uses the appropriate libraries without you needing to take any further action. IBM WebSphere MQ uses libraries from the installation associated with the queue manager that the application connects to.

The following concepts are used to explain the way applications connect to IBM WebSphere MQ:

Linking
When the application is compiled, the application is linked to the IBM WebSphere MQ libraries to get the function exports that are then loaded when the application runs.

Loading
When the application is run, the IBM WebSphere MQ libraries are located and loaded. The specific mechanism used to locate the libraries varies by operating system, and by how the application is built. For more information about how to locate and load libraries in a multiple installation environment, see “Loading IBM WebSphere MQ Version 7.1 or later libraries” on page 8.

Connecting
When the application connects to a running queue manager, for example, using a `MQCONN` or `MQCONNX` call, it connects using the loaded IBM WebSphere MQ libraries.

When a server application connects to a queue manager, the loaded libraries must come from the installation associated with the queue manager. With multiple installations on a system, this restriction introduces new challenges when choosing the mechanism that the operating system uses to locate the IBM WebSphere MQ libraries to load:

- When the **setmqm** command is used to change the installation associated with a queue manager, the libraries that need to be loaded change.
- When an application connects to multiple queue managers that are owned by different installations, multiple sets of libraries need to be loaded.

However, if IBM WebSphere MQ Version 7.1, or later, libraries, are located and loaded, IBM WebSphere MQ then loads and uses the appropriate libraries without you needing to take any further action. When the application connects to a queue manager, IBM WebSphere MQ loads libraries from the installation that the queue manager is associated with.

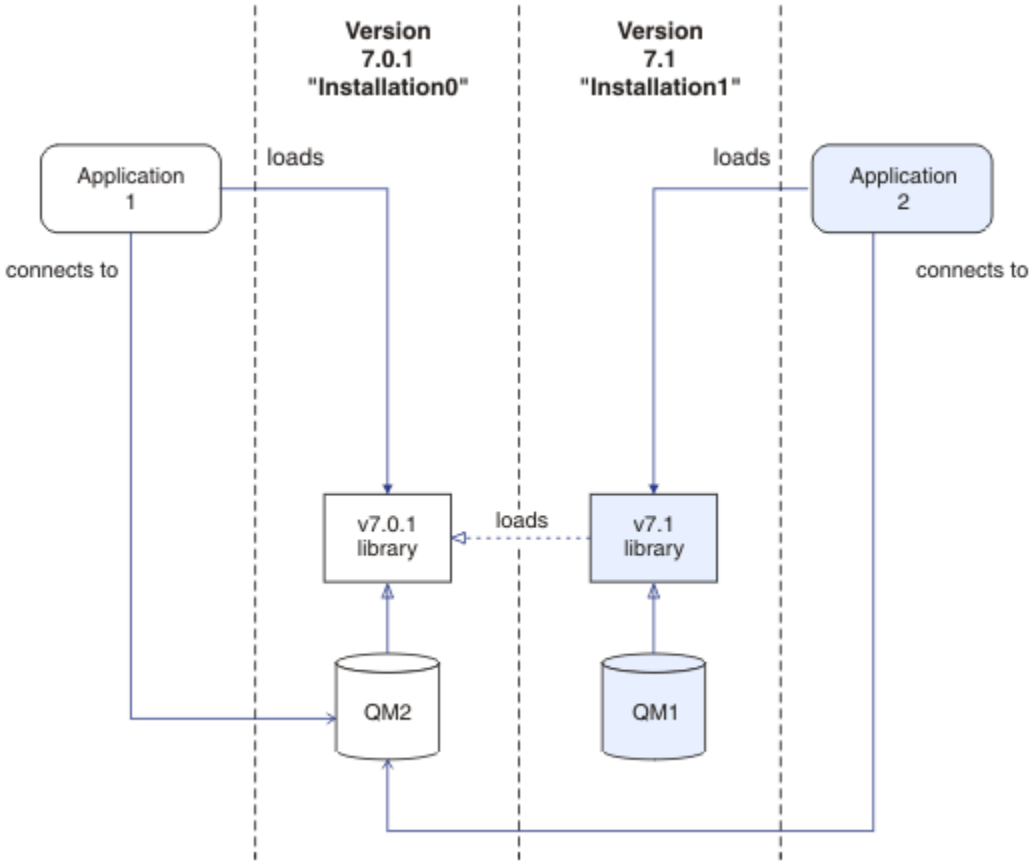


Figure 1. Connecting applications in a multiple installation environment

For example, Figure 1 on page 7 shows a multiple installation environment with a version 7.0.1 installation (`Installation0`), and a version 7.1 installation (`Installation1`). Two applications are connected to these installations, but they load different library versions.

`Application 1` directly loads a version 7.0.1 library. When `application 1` connects to `QM2`, the version 7.0.1 libraries are used . If `application 1` attempts to connect to `QM1`, or if `QM2` is associated with `Installation1`, `application 1` fails with a 2059 (080B) (RC2059): MQRC_Q_MGR_NOT_AVAILABLE error. The application fails because the version 7.0.1 library is not capable of loading other library versions. That is, if version 7.0.1 libraries are directly loaded, you cannot use a queue manager associated with an installation at a later version of IBM WebSphere MQ.

`Application 2` directly loads a version 7.1 library. When `application 2` connects to `QM2`, the version 7.1 library then loads and uses the version 7.0.1 library. If `application 2` connects to `QM1`, or if `QM2` is associated with `Installation1`, the version 7.1 library is loaded, and the application works as expected.

Migration scenarios and connecting applications with multiple installations is considered in more detail in Multi-installation queue manager coexistence on UNIX, Linux, and Windows.

For more information about how to load IBM WebSphere MQ Version 7.1 libraries, see “Loading IBM WebSphere MQ Version 7.1 or later libraries” on page 8.

Support and restrictions

If any of the following version 7.1, or later, libraries, are located and loaded, IBM WebSphere MQ can automatically load and use the appropriate libraries:

- The C server libraries
- The C++ server libraries

- The XA server libraries
- The COBOL server libraries
- The COM+ server libraries
- .NET in unmanaged mode

IBM WebSphere MQ also automatically loads and uses the appropriate libraries for Java and JMS applications in bindings mode.

There are a number of restrictions for applications using multiple installations. For more information, see “Restrictions for applications using multiple installations” on page 11.

Related concepts

“Associating a queue manager with an installation” on page 16
When you create a queue manager, it is automatically associated with the installation that issued the **crtmqm** command. On UNIX, Linux, and Windows, you can change the installation associated with a queue manager using the **setmqm** command.

“Restrictions for applications using multiple installations” on page 11
There are restrictions when using CICS server libraries, fast path connections, message handles, and exits in a multiple installation environment.

“Loading IBM WebSphere MQ Version 7.1 or later libraries” on page 8
When deciding how to load IBM WebSphere MQ libraries, you need to consider a number of factors, including: your environment, whether you can change your existing applications, whether you want a primary installation, where IBM WebSphere MQ is installed, and whether the location of IBM WebSphere MQ is likely to change.

Related tasks

Choosing a primary installation
“Changing the primary installation” on page 14
You can use the **setmqinst** command to set or unset an installation as the primary installation.

Loading IBM WebSphere MQ Version 7.1 or later libraries

When deciding how to load IBM WebSphere MQ libraries, you need to consider a number of factors, including: your environment, whether you can change your existing applications, whether you want a primary installation, where IBM WebSphere MQ is installed, and whether the location of IBM WebSphere MQ is likely to change.

How IBM WebSphere MQ Version 7.1 libraries are located and loaded depends on your installation environment:

- On UNIX and Linux systems, if a copy of IBM WebSphere MQ Version 7.1 is installed in the default location, existing applications continue to work in the same way as previous versions. However, if the applications need symbolic links in `/usr/lib`, you must either select a version 7.1 installation to be the primary installation, or manually create the symbolic links.
- If IBM WebSphere MQ Version 7.1 is installed in a non-default location, which is the case if IBM WebSphere MQ Version 7.0.1 is also installed, you might need to change your existing applications so that the correct libraries are loaded.

How IBM WebSphere MQ Version 7.1, or later, libraries can be located and loaded also depends on how any existing applications are set up to load libraries. For more information about how libraries can be loaded, see “Operating system library loading mechanisms” on page 10.

Optimally, you should ensure that the queue manager is associated with the IBM WebSphere MQ library that is loaded by the operating system.

The methods for loading IBM WebSphere MQ libraries vary by platform, and each method has benefits and drawbacks.

Table 1. Benefits and drawbacks of the options for loading libraries

Platform	Option	Benefits	Drawbacks
UNIX and Linux systems	Set or change the embedded runtime search path (RPath) of the application. This option requires you to recompile and link the application. For more information about compiling and linking applications, see Building a WebSphere MQ application.	• Scope of the change is clear.	• You must be able to recompile and link the application. • If the location of IBM WebSphere MQ changes, you must change the RPath.
UNIX and Linux systems	Set the *LD_LIBRARY_PATH* environment variable (*LIBPATH* on AIX®), using setmqenv, or crtmqenv, with the **-k** or **-l** option.	• No changes to existing applications required. • Overrides embedded RPaths in an application. • Easy to change the variable if the location of IBM WebSphere MQ changes.	• `setuid` and `setgid` applications, or applications built in other ways, might ignore *LD_LIBRARY_PATH* for security reasons. • Environment specific, so must be set in each environment where the application is run. • Possible impact on other applications that rely on *LD_LIBRARY_PATH*. • HP-UX: Options used when the application was compiled might disable the use of *LD_LIBRARY_PATH*. For more information, see Runtime linking considerations for HP-UX. • Linux: The compiler used to build the application might disable the use of *LD_LIBRARY_PATH*. For more information, see Runtime linking considerations for Linux.
Windows systems	Set the PATH variable using setmqenv, or crtmqenv.	• No changes required for existing applications. • Easy to change the variable if the location of IBM WebSphere MQ changes.	• Environment specific, so must be set in each environment where the application is run. • Possible impact on other applications.

Table 1. Benefits and drawbacks of the options for loading libraries (continued)			
Platform	**Option**	**Benefits**	**Drawbacks**
UNIX, Linux, and Windows systems	Set the primary installation to a version 7.1, or later, installation. See "Changing the primary installation" on page 14. For more information about the primary installation, see Choosing a primary installation .	• No changes required for existing applications. • Easy to change the primary installation if the location of IBM WebSphere MQ changes. • Gives similar behavior to previous versions of IBM WebSphere MQ.	• When WebSphere MQ version 7.0.1 is installed, you cannot set the primary installation to version 7.1, or later. • UNIX and Linux: Does not work if `/usr/lib` is not in the default search path.

Library loading considerations for HP-UX

The sample compilation commands in the product documentation for previous versions of IBM WebSphere MQ included the `-W1, +noenvvar` link option for 64-bit applications. This option disables the use of *LD_LIBRARY_PATH* to load shared libraries. If you want your applications to load IBM WebSphere MQ libraries from a location other than the location specified in the RPath, you must update your applications. You can update the applications by recompiling and linking without the `-W1, +noenvvar` link option, or by using the **chatr** command.

To find out how your applications currently load libraries, see "Operating system library loading mechanisms" on page 10.

Library loading considerations for Linux

Applications compiled using some versions of gcc, for example, version 3.2.x, can have an embedded RPath that cannot be overridden using the *LD_LIBRARY_PATH* environment variable. You can determine if an application is affected by using the `readelf -d` *applicationName* command. The RPath cannot be overridden if the RPATH symbol is present and the RUNPATH symbol is not present.

Library loading considerations for Solaris

The sample compilation commands in the product documentation for previous versions of IBM WebSphere MQ included the `-lmqmcs -lmqmzse` link options. The appropriate versions of these libraries are now loaded automatically by IBM WebSphere MQ. If IBM WebSphere MQ is installed in a non-default location, or if there are multiple installations on the system, you must update your applications. You can update the applications by recompiling and linking without the `-lmqmcs -lmqmzse` link options.

Operating system library loading mechanisms

On Windows systems, several directories are searched to find the libraries:

- The directory the application is loaded from.
- The current directory.
- The directories in the *PATH* environment variable, both the global *PATH* variable and the *PATH* variable of the current user.

On UNIX and Linux systems, there are a number of methods that might have been used to locate the libraries to load:

- Using the *LD_LIBRARY_PATH* environment variable (also *LIBPATH* on AIX, and *SHLIB_PATH* on HP-UX). If this variable is set, it defines a set of directories that are searched for the required WebSphere MQ libraries. If any libraries are found in these directories, they are used in preference of any libraries that might be found using the other methods.

- Using an embedded search path (RPath). The application might contain a set of directories to search for the IBM WebSphere MQ libraries. If the *LD_LIBRARY_PATH* is not set, or if the required libraries were not found using the variable, the RPath is searched for the libraries. If your existing applications use an RPath, but you cannot recompile and link the application, you must either install IBM WebSphere MQ Version 7.1 in the default location, or use another method to find the libraries.
- Using the default library path. If the WebSphere MQ libraries are not found after searching the *LD_LIBRARY_PATH* variable and RPath locations, the default library path is searched. Usually, this path contains `/usr/lib` or `/usr/lib64`. If the libraries are not found after searching the default library path, the application fails to start because of missing dependencies.

You can use operating system mechanisms to find out if your applications have an embedded search path. For example:

- AIX: **dump**
- HP-UX: **chatr**
- Linux: **readelf**
- Solaris: **elfdump**

Related concepts

"Associating a queue manager with an installation" on page 16
When you create a queue manager, it is automatically associated with the installation that issued the **crtmqm** command. On UNIX, Linux, and Windows, you can change the installation associated with a queue manager using the **setmqm** command.

"Restrictions for applications using multiple installations" on page 11
There are restrictions when using CICS server libraries, fast path connections, message handles, and exits in a multiple installation environment.

"Connecting applications in a multiple installation environment" on page 6
On UNIX, Linux, and Windows systems, if IBM WebSphere MQ Version 7.1, or later, libraries are loaded, IBM WebSphere MQ automatically uses the appropriate libraries without you needing to take any further action. IBM WebSphere MQ uses libraries from the installation associated with the queue manager that the application connects to.

Related tasks

Choosing a primary installation

"Changing the primary installation" on page 14
You can use the **setmqinst** command to set or unset an installation as the primary installation.

Restrictions for applications using multiple installations

There are restrictions when using CICS server libraries, fast path connections, message handles, and exits in a multiple installation environment.

CICS server libraries

If you are using the CICS server libraries, IBM WebSphere MQ does not automatically select the correct library level for you. You must compile and link your applications with the appropriate library level for the queue manager to which the application connects. For more information, see Building libraries for use with TXSeries® for Multiplatforms version 5 .

Message handles

Message handles that use the special value of `MQHC_UNASSOCIATED_HCONN` are limited to use with the first installation loaded in a process. If the message handle cannot be used by a particular installation, reason code `MQRC_HMSG_NOT_AVAILABLE` is returned.

This restriction affects message properties. You cannot use message handles to get message properties from a queue manager on one installation and put them to a queue manager on a different installation. For more information about message handles, see MQCRTMH - Create message handle.

Exits

In a multiple installation environment, existing exits must be updated for use with IBM WebSphere MQ Version 7.1, or later, installations. Data conversion exits generated using the **crtmqcvx** command must be regenerated using the updated command.

All exits must be written using the MQIEP structure, cannot use an embedded RPATH to locate the IBM WebSphere MQ libraries, and cannot link to the IBM WebSphere MQ libraries. For more information, see Writing and compiling exits and installable services.

Fast path

On a server with multiple installations, applications using a fast path connection to IBM WebSphere MQ Version 7.1 or later must follow these rules:

1. The queue manager must be associated with the same installation as the one from which the application loaded the IBM WebSphere MQ run time libraries. The application must not use a fast path connection to a queue manager associated with a different installation. An attempt to make the connection results in an error, and reason code MQRC_INSTALLATION_MISMATCH.
2. Connecting non-fast path to a queue manager associated with the same installation as the one from which the application has loaded the IBM WebSphere MQ run time libraries prevents the application connecting fast path, unless either of these conditions are true:
   - The application makes its first connection to a queue manager associated with the same installation a fast path connection.
   - The environment variable, AMQ_SINGLE_INSTALLATION is set.
3. Connecting non-fast path to a queue manager associated with a Version 7.1 or later installation, has no effect on whether an application can connect fast path.
4. You cannot combine connecting to a queue manager associated with a Version 7.0.1 installation and connecting fast path to a queue manager associated with a Version 7.1, or later installation.

With AMQ_SINGLE_INSTALLATION set, you can make any connection to a queue manager a fast path connection. Otherwise almost the same restrictions apply:

- The installation must be the same one from which the IBM WebSphere MQ run time libraries were loaded.
- Every connection on the same process must be to the same installation. If you attempt to connect to a queue manager associated with a different installation, the connection fails with reason code MQRC_INSTALLATION_MISMATCH. Note that with AMQ_SINGLE_INSTALLATION set, this restriction applies to all connections, not only fast path connections.
- Only connect one queue manager with fast path connections.

Related reference

MQCONNX - Connect queue manager (extended)
MQIEP structure
2583 (0A17) (RC2583): MQRC_INSTALLATION_MISMATCH
2587 (0A1B) (RC2587): MQRC_HMSG_NOT_AVAILABLE
2590 (0A1E) (RC2590): MQRC_FASTPATH_NOT_AVAILABLE

Connecting .NET applications in a multiple installation environment

By default, applications use the .NET assemblies from the primary installation. If there is no primary installation, or you do not want to use the primary installation assemblies, you must update the application configuration file, or the *DEVPATH* environment variable.

If there is a primary installation on the system, the .NET assemblies and policy files of that installation are registered to the global assembly cache (GAC). The .NET assemblies for all other installations can be found in the installation path of each installation, but the assemblies are not registered to the GAC.

Therefore, by default, applications run using the .NET assemblies from the primary installation. You must update the application configuration file if any of the following cases are true:

- You do not have a primary installation.
- You do not want the application to use the primary installation assemblies.
- The primary installation is a lower version of IBM WebSphere MQ than the version that the application was compiled with.

For information about how to update the application configuration file, see "Connecting .NET applications using the application configuration file" on page 13.

You must update the *DEVPATH* environment variable if the following case is true:

- You want your application to use the assemblies from a non-primary installation, but the primary installation is at the same version as the non-primary installation.

For more information about how to update the *DEVPATH* variable, see "Connecting .NET applications using DEVPATH" on page 14.

Connecting .NET applications using the application configuration file

Within the application configuration file, you must set various tags to redirect applications to use assemblies that are not from the primary installation.

The following table shows the specific changes that need to be made to the application configuration file to allow .NET applications connect using particular assemblies:

Table 2. Configuring applications to use particular assemblies

	Applications compiled with a lower version of IBM WebSphere MQ	**Applications compiled with a higher version of IBM WebSphere MQ**
To run an application with a higher version IBM WebSphere MQ primary installation. (higher version assemblies in GAC):	No changes necessary	No changes necessary
To run an application with a lower version IBM WebSphere MQ primary installation. (lower version assemblies in GAC):	No changes necessary	In the application configuration file: • Use the *<bindingRedirect>* tag to indicate the use of the lower version of the assemblies that are in the GAC
To run an application with a higher version of IBM WebSphere MQ non-primary installation. (higher version assemblies in installation folder):	In the application configuration file: • Use the *<codebase>* tag to point to the location of the higher version assemblies • Use the *<bindingRedirect>* tag to indicate the use of the higher version assemblies	In the application configuration file: • Use the *<codebase>* tag to point to the location of the higher version assemblies

Table 2. Configuring applications to use particular assemblies (continued)		
	Applications compiled with a lower version of IBM WebSphere MQ	**Applications compiled with a higher version of IBM WebSphere MQ**
To run an application with a lower version of IBM WebSphere MQ non-primary installation. (lower version assemblies in installation folder):	In the application configuration file: • Use the *<codebase>* tag to point to the location of the lower version assemblies • Include the tag *<publisherpolicy Apply=no>*	In the application configuration file: • Use the *<codebase>* tag to point to the location of the lower version assemblies • Use the *<bindingRedirect>* tag to indicate the use of the lower version assemblies • Include the tag *<publisherpolicy Apply=no>*

A sample application configuration file `NonPrimaryRedirect.config` is shipped in the folder *MQ_INSTALLATION_PATH*`\tools\dotnet\samples\base`. This file can be modified with the IBM WebSphere MQ installation path of any non-primary installation. The file can also be directly included in other configuration files using the *<linkedConfiguration>* tag. Samples are provided for `nmqsget.exe.config` and `nmqsput.exe.config`. Both samples use the *<linkedConfiguration>* tag and include the `NonPrimaryRedirect.config` file.

Connecting .NET applications using DEVPATH

You can find the assemblies using the *DEVPATH* environment variable. The assemblies specified by the *DEVPATH* variable are used in preference to any assemblies in the GAC. See the appropriate Microsoft documentation on *DEVPATH* for more information about when to use this variable.

To find the assemblies using the *DEVPATH* environment variable, you must set the *DEVPATH* variable to the folder that contains the assemblies you want to use. Then, you must then update the application configuration file and add the following runtime configuration information:

```
<configuration>
   <runtime>
       <developmentMode developerInstallation="true" />
   </runtime>
</configuration>
```

Related concepts

“Connecting applications in a multiple installation environment” on page 6
On UNIX, Linux, and Windows systems, if IBM WebSphere MQ Version 7.1, or later, libraries are loaded, IBM WebSphere MQ automatically uses the appropriate libraries without you needing to take any further action. IBM WebSphere MQ uses libraries from the installation associated with the queue manager that the application connects to.

Multiple installations

Related tasks

Choosing a primary installation

Using .NET

Changing the primary installation

You can use the **setmqinst** command to set or unset an installation as the primary installation.

About this task

This task applies to UNIX, Linux, and Windows.

The primary installation is the installation to which required system-wide locations refer. For more information about the primary installation, and considerations for choosing your primary installation, see Choosing a primary installation.

If an installation of IBM WebSphere MQ Version 7.1 or later is coexisting with an installation of IBM WebSphere MQ Version 7.0.1, the IBM WebSphere MQ Version 7.0.1 installation must be the primary. It is flagged as primary when the IBM WebSphere MQ Version 7.1 or later version is installed, and the IBM WebSphere MQ Version 7.1 or later installation cannot be made primary.

During the installation process on Windows, you can specify that the installation is to be the primary installation. On UNIX and Linux systems, you must issue a **setmqinst** command after installation to set the installation as the primary installation.

"Set the primary installation" on page 15.

"Unset the primary installation" on page 15.

Set the primary installation

Procedure

To set an installation as the primary installation:

1. Check if an installation is already the primary installation by entering the following command:

   ```
   MQ_INSTALLATION_PATH/bin/dspmqinst
   ```

   where *MQ_INSTALLATION_PATH* is the installation path of a IBM WebSphere MQ Version 7.1 or later installation.
2. If an existing IBM WebSphere MQ Version 7.1 or later installation is set as the primary installation, unset it by following the instructions in "Unset the primary installation" on page 15. If IBM WebSphere MQ Version 7.0.1 is installed on the system, the primary installation cannot be changed.
3. As root on UNIX and Linux systems, or a member of the Administrators group on Windows systems, enter one of the following commands:
   - To set the primary installation using the path of the installation you want to be the primary installation:

     ```
     MQ_INSTALLATION_PATH/bin/setmqinst -i -p MQ_INSTALLATION_PATH
     ```

   - To set the primary installation using the name of the installation you want to be the primary installation:

     ```
     MQ_INSTALLATION_PATH/bin/setmqinst -i -n installationName
     ```
4. On Windows systems, restart the system.

Unset the primary installation

Procedure

To unset an installation as the primary installation:

1. Check which installation is the primary installation by entering the following command:

   ```
   MQ_INSTALLATION_PATH/bin/dspmqinst
   ```

   where *MQ_INSTALLATION_PATH* is the installation path of a IBM WebSphere MQ Version 7.1 or later installation.

   If IBM WebSphere MQ Version 7.0.1 is the primary installation, you cannot unset the primary installation.

2. As root on UNIX and Linux systems, or a member of the Administrators group on Windows systems, enter one of the following commands:
   - To unset the primary installation using the path of the installation you no longer want to be the primary installation:

     ```
     MQ_INSTALLATION_PATH/bin/setmqinst -x -p MQ_INSTALLATION_PATH
     ```

   - To unset the primary installation using the name of the installation you no longer want to be the primary installation:

     ```
     MQ_INSTALLATION_PATH/bin/setmqinst -x -n installationName
     ```

Related concepts

Features that can be used only with the primary installation on Windows

External library and control command links to primary installation on UNIX and Linux

Related tasks

Uninstalling, upgrading, and maintaining the primary installation

Choosing an installation name

Related reference

setmqinst

Associating a queue manager with an installation

When you create a queue manager, it is automatically associated with the installation that issued the **crtmqm** command. On UNIX, Linux, and Windows, you can change the installation associated with a queue manager using the **setmqm** command.

You can use the **setmqm** command in the following ways:

- Moving individual queue managers between equivalent versions of WebSphere MQ. For example, moving a queue manager from a test to a production system.
- Migrating individual queue managers from an older version of WebSphere MQ to a newer version of WebSphere MQ. Migrating queue managers between versions has various implications that you must be aware of. For more information about migrating, see Migrating and upgrading WebSphere MQ.

To associate a queue manager with an installation:

1. Stop the queue manager using the **endmqm** command from the installation that is currently associated with the queue manager.
2. Associate the queue manager with another installation using the **setmqm** command from that installation.

   For example, to set queue manager QMB to be associated with an installation with the name Installation2, enter the following command from Installation2:

   ```
   MQ_INSTALLATION_PATH/bin/setmqm -m QMB -n Installation2
   ```

   where *MQ_INSTALLATION_PATH* is the path where Installation2 is installed.
3. Start the queue manager using the **strmqm** command from the installation that is now associated with the queue manager.

   This command performs any necessary queue manager migration and results in the queue manager being ready to use.

The installation that a queue manager is associated with limits that queue manager so that it can be administered only by commands from that installation. There are three key exceptions:

- **setmqm** changes the installation associated with the queue manager. This command must be issued from the installation that you want to associate with the queue manager, not the installation that the

queue manager is currently associated with. The installation name specified by the **setmqm** command has to match the installation from which the command is issued.

- **strmqm** usually has to be issued from the installation that is associated with the queue manager. However, when a V7.0.1 or earlier queue manager is started on a V7.1 or later installation for the first time, **strmqm** can be used. In this case, **strmqm** starts the queue manager and associates it with the installation from which the command is issued.
- **dspmq** displays information about all queue managers on a system, not just those queue managers associated with the same installation as the **dspmq** command. The `dspmq -o installation` command displays information about which queue managers are associated with which installations.

Queue manager association in HA environments

For HA environments, the **addmqinf** command automatically associates the queue manager with the installation from which the **addmqinf** command is issued. As long as the **strmqm** command is then issued from the same installation as the **addmqinf** command, no further setup is required. To start the queue manager using a different installation, you must first change the associated installation using the **setmqm** command.

Queue managers associated with deleted installations

If the installation that a queue manager is associated with has been deleted, or if the queue manager status information is unavailable, the **setmqm** command fails to associate the queue manager with another installation. In this situation, take the following actions:

1. Use the **dspmqinst** command to see the other installations on your system.
2. Manually modify the `InstallationName` field of the `QueueManager` stanza in mqs.ini to specify another installation.
3. Use the **dltmqm** command from that installation to delete the queue manager.

Related concepts

"Finding installations of IBM WebSphere MQ on a system" on page 17
If you have multiple IBM WebSphere MQ installations on a system, you can check which versions are installed and where they are.

"The IBM WebSphere MQ configuration file, mqs.ini" on page 409
The IBM WebSphere MQ configuration file, mqs.ini, contains information relevant to all the queue managers on the node. It is created automatically during installation.

Related tasks

Choosing a primary installation

Related reference

setmqm

strmqm

dspmq

dspmqinst

Finding installations of IBM WebSphere MQ on a system

If you have multiple IBM WebSphere MQ installations on a system, you can check which versions are installed and where they are.

You can use the following methods to find the IBM WebSphere MQ installations on your system:

- Use the **dspmqver** command. This command does not provide details of all installations on a system if it is issued from a Version 7.0.1 installation.
- Use the platform installation tools to query where IBM WebSphere MQ has been installed. Then use the **dspmqver** command from a Version 7.1 or later installation. The following commands are examples of commands you can use to query where IBM WebSphere MQ has been installed:

- On AIX systems, you can use the **lslpp** command:

  ```
  lslpp -R ALL -l mqm.base.runtime
  ```

- On HP-UX systems, you can use the **swlist** command:

  ```
  swlist -a location -a revision -l product MQSERIES
  ```

- On Linux systems, you can use the **rpm** command:

  ```
  rpm -qa --qf "%{NAME}-%{VERSION}-%{RELEASE}\t%{INSTPREFIXES}\n" | grep MQSeriesRuntime
  ```

- On Solaris systems, you can use the **pkginfo** and **pkgparam** commands:

  1. List the installed packages by entering the following command:

     ```
     pkginfo | grep -w mqm
     ```

  2. For each package listed, enter following command:

     ```
     pkgparam pkgname BASEDIR
     ```

- On Windows systems, you can use the **wmic** command. This command might install the wmic client:

  ```
  wmic product where "(Name like '%MQ%') AND (not Name like '%bitSupport')" get Name,
  Version, InstallLocation
  ```

- On UNIX and Linux systems, issue the following command to find out where IBM WebSphere MQ has been installed:

  ```
  cat /etc/opt/mqm/mqinst.ini
  ```

  Then use the **dspmqver** command from a Version 7.1 or later installation.

- To display details of installations on the system, on 32-bit Windows, issue the following command:

  ```
  reg.exe query "HKEY_LOCAL_MACHINE\SOFTWARE\IBM\WebSphere MQ\Installation" /s
  ```

- On 64-bit Windows, issue the following command:

  ```
  reg.exe query "HKEY_LOCAL_MACHINE\SOFTWARE\Wow6432Node\IBM\WebSphere MQ\Installation" /s
  ```

Note: the **reg.exe** command will only display information for Version 7.1 or later installations.

Related concepts
Multiple installations
Related reference
dspmqver
dspmqinst

Creating and managing queue managers

Before you can use messages and queues, you must create and start at least one queue manager and its associated objects.

Creating a queue manager

A queue manager manages the resources associated with it, in particular the queues that it owns. It provides queuing services to applications for Message Queuing Interface (MQI) calls and commands to create, modify, display, and delete IBM WebSphere MQ objects.

To create a queue manager, use the IBM WebSphere MQ control command **crtmqm** (described in **crtmqm**). The **crtmqm** command automatically creates the required default objects and system objects (described in System default objects). Default objects form the basis of any object definitions that

you make; system objects are required for queue manager operation. When you have created a queue manager and its objects, use the `strmqm` command to start the queue manager.

Note: IBM WebSphere MQ does not support machine names that contain spaces. If you install IBM WebSphere MQ on a computer with a machine name that contains spaces, you cannot create any queue managers.

On

Before you can create a queue manager, there are several points you must consider (especially in a production environment). Work through the following checklist:

The installation associated with the queue manager
The **`crtmqm`** command automatically associates a queue manager with the installation from which the **`crtmqm`** command was issued. For commands that operate on a queue manager, you must issue the command from the installation associated with the queue manager. You can change the associated installation of a queue manager using the setmqm command. Note the Windows installer does not add the user that performs the install to the mqm group, for more details, see Authority to administer IBM WebSphere MQ on UNIX, Linux and Windows systems.

Naming conventions
Use uppercase names so that you can communicate with queue managers on all platforms. Remember that names are assigned exactly as you enter them. To avoid the inconvenience of lots of typing, do not use unnecessarily long names.

Specify a unique queue manager name
When you create a queue manager, ensure that no other queue manager has the same name *anywhere* in your network. Queue manager names are not checked when the queue manager is created, and names that are not unique prevent you from creating channels for distributed queuing.

One way of ensuring uniqueness is to prefix each queue manager name with its own unique node name. For example, if a node is called `ACCOUNTS`, you can name your queue manager `ACCOUNTS.SATURN.QUEUE.MANAGER`, where `SATURN` identifies a particular queue manager and `QUEUE.MANAGER` is an extension you can give to all queue managers. Alternatively, you can omit this, but note that `ACCOUNTS.SATURN` and `ACCOUNTS.SATURN.QUEUE.MANAGER` are *different* queue manager names.

If you are using IBM WebSphere MQ for communication with other enterprises, you can also include your own enterprise name as a prefix. This is not done in the examples, because it makes them more difficult to follow.

Note: Queue manager names in control commands are case-sensitive. This means that you are allowed to create two queue managers with the names `jupiter.queue.manager` and `JUPITER.queue.manager`. However, it is better to avoid such complications.

Limit the number of queue managers
You can create as many queue managers as resources allow. However, because each queue manager requires its own resources, it is generally better to have one queue manager with 100 queues on a node than to have ten queue managers with ten queues each.

In production systems, many processors can be exploited with a single queue manager, but larger server machines might run more effectively with multiple queue managers.

Specify a default queue manager
Each node should have a default queue manager, though it is possible to configure IBM WebSphere MQ on a node without one. The default queue manager is the queue manager that applications connect to if they do not specify a queue manager name in an `MQCONN` call. It is also the queue manager that processes MQSC commands when you invoke the `runmqsc` command without specifying a queue manager name.

Specifying a queue manager as the default *replaces* any existing default queue manager specification for the node.

Changing the default queue manage can affect other users or applications. The change has no effect on currently-connected applications, because they can use the handle from their original connect call

in any further MQI calls. This handle ensures that the calls are directed to the same queue manager. Any applications connecting *after* you have changed the default queue manager connect to the new default queue manager. This might be what you intend, but you should take this into account before you change the default.

Creating a default queue manager is described in "Creating a default queue manager" on page 21.

Specify a dead-letter queue

The dead-letter queue is a local queue where messages are put if they cannot be routed to their intended destination.

It is important to have a dead-letter queue on each queue manager in your network. If you do not define one, errors in application programs might cause channels to be closed, and replies to administration commands might not be received.

For example, if an application tries to put a message on a queue on another queue manager, but gives the wrong queue name, the channel is stopped and the message remains on the transmission queue. Other applications cannot then use this channel for their messages.

The channels are not affected if the queue managers have dead-letter queues. The undelivered message is put on the dead-letter queue at the receiving end, leaving the channel and its transmission queue available.

When you create a queue manager, use the -u flag to specify the name of the dead-letter queue. You can also use an MQSC command to alter the attributes of a queue manager that you have already defined to specify the dead-letter queue to be used. See Working with queue managers for an example of the MQSC command ALTER.

Specify a default transmission queue

A transmission queue is a local queue on which messages in transit to a remote queue manager are queued before transmission. The default transmission queue is the queue that is used when no transmission queue is explicitly defined. Each queue manager can be assigned a default transmission queue.

When you create a queue manager, use the -d flag to specify the name of the default transmission queue. This does not actually create the queue; you have to do this explicitly later on. See Working with local queues for more information.

Specify the logging parameters you require

You can specify logging parameters on the `crtmqm` command, including the type of logging, and the path and size of the log files.

In a development environment, the default logging parameters should be adequate. However, you can change the defaults if, for example:

- You have a low-end system configuration that cannot support large logs.
- You anticipate a large number of long messages being on your queues at the same time.
- You anticipate a lot of persistent messages passing through the queue manager.

Once you have set the logging parameters, some of them can only be changed by deleting the queue manager and recreating it with the same name but with different logging parameters.

For more information about logging parameters, see "Availability, recovery and restart" on page 303.

For IBM WebSphere MQ for UNIX systems only

You can create the queue manager directory `/var/mqm/qmgrs/<qmgr>`, even on a separate local file system, before you use the `crtmqm` command. When you use `crtmqm`, if the `/var/mqm/qmgrs/<qmgr>` directory exists, is empty, and is owned by mqm, it is used for the queue manager data. If the directory is not owned by mqm, the creation fails with a First Failure Support Technology (FFST) message. If the directory is not empty, a new directory is created.

Related concepts

"Configuring" on page 5

Create one or more queue managers on one or more computers, and configure them on your development, test, and production systems to process messages that contain your business data.

“Backing up configuration files after creating a queue manager” on page 23
IBM WebSphere MQ configuration information is stored in configuration files on Windows, UNIX and Linux systems.

“Starting a queue manager” on page 24
When you create a queue manager, you must start it to enable it to process commands or MQI calls.

“Stopping a queue manager” on page 24
There are three ways to stop a queue manager: a quiesced shutdown, and immediate shutdown, and a preemptive shutdown.

“Restarting a queue manager” on page 25
You can use the **strmqm** command to restart a queue manager, or, on IBM WebSphere MQ for Windows and IBM WebSphere MQ for Linux (x86 and x86-64 platforms) systems, restart a queue manager from IBM WebSphere MQ Explorer.

“Changing IBM WebSphere MQ and queue manager configuration information” on page 407
Change the behavior of IBM WebSphere MQ or an individual queue manager to suit the needs of your installation.

System and default objects

Related tasks

“Making an existing queue manager the default” on page 22
You can make an existing queue manager the default queue manager. The way you do this depends on the platform you are using.

“Deleting a queue manager” on page 26
You can delete a queue manager using the **dltmqm** command or by using the WebSphere MQ Explorer.

distributed Creating a default queue manager

The default queue manager is the queue manager that applications connect to if they do not specify a queue manager name in an MQCONN call. It is also the queue manager that processes MQSC commands when you invoke the **runmqsc** command without specifying a queue manager name. To create a queue manager, you use the IBM WebSphere MQ control command **crtmqm**.

Before you begin

Before creating a default queue manager, read through the considerations described in “Creating and managing queue managers” on page 18.

UNIX When you use **crtmqm** to create a queue manager on UNIX and Linux, if the `/var/mqm/qmgrs/<qmgr>` directory already exists, is owned by mqm, and is empty, it is used for the queue manager data. If the directory is not owned by mqm, the creation of the queue manager fails with a First Failure Support Technology (FFST) message. If the directory is not empty, a new directory is created for the queue manager data.

This consideration applies even when the `/var/mqm/qmgrs/<qmgr>` directory already exists on a separate local file system.

About this task

When you create a queue manager by using the **crtmqm** command, the command automatically creates the required default objects and system objects. Default objects form the basis of any object definitions that you make and system objects are required for queue manager operation.

By including the relevant parameters in the command, you can also define, for example, the name of the default transmission queue to be used by the queue manager, and the name of the dead letter queue.

Windows On Windows, you can use the **sax** option of the **crtmqm** command to start multiple instances of the queue manager.

For more information about the **crtmqm** command and its syntax, see **crtmqm**.

Procedure

- To create a default queue manager, use the **crtmqm** command with the **-q** flag.

  The following example of the **crtmqm** command creates a default queue manager called SATURN.QUEUE.MANAGER:

  ```
  crtmqm -q -d MY.DEFAULT.XMIT.QUEUE -u SYSTEM.DEAD.LETTER.QUEUE SATURN.QUEUE.MANAGER
  ```

  where:

  -q
  Indicates that this queue manager is the default queue manager.

  -d MY.DEFAULT.XMIT.QUEUE
  Is the name of the default transmission queue to be used by this queue manager.

  Note: IBM WebSphere MQ does not create a default transmission queue for you; you have to define it yourself.

  -u SYSTEM.DEAD.LETTER.QUEUE
  Is the name of the default dead-letter queue created by IBM WebSphere MQ on installation.

  SATURN.QUEUE.MANAGER
  Is the name of this queue manager. This must be the last parameter specified on the crtmqm command.

What to do next

When you have created a queue manager and its objects, use the **strmqm** command to start the queue manager.

Related concepts

“Backing up configuration files after creating a queue manager” on page 23
IBM WebSphere MQ configuration information is stored in configuration files on Windows, UNIX and Linux systems.

Working with queue managers

Working with local queues

Related reference

System and default objects

Making an existing queue manager the default

You can make an existing queue manager the default queue manager. The way you do this depends on the platform you are using.

WebSphere MQ for Windows and WebSphere MQ for Linux (x86 and x86-64 platforms) systems

About this task

Use the following instructions to make an existing queue manager the default queue manager on WebSphere MQ for Windows and WebSphere MQ for Linux (x86 and x86-64 platforms) systems:

Procedure

1. Open the IBM WebSphere MQ Explorer.
2. Right-click `IBM WebSphere MQ`, then select `Properties...`. The Properties for WebSphere MQ panel is displayed.
3. Type the name of the default queue manager into the Default queue manager name field.
4. Click OK.

UNIX and Linux systems

About this task

When you create a default queue manager, its name is inserted in the `Name` attribute of the `DefaultQueueManager` stanza in the WebSphere MQ configuration file (mqs.ini). The stanza and its contents are automatically created if they do not exist.

Procedure

- To make an existing queue manager the default, change the queue manager name on the `Name` attribute to the name of the new default queue manager. You can do this manually, using a text editor.
- If you do not have a default queue manager on the node, and you want to make an existing queue manager the default, create the *`DefaultQueueManager`* stanza with the required name yourself.
- If you accidentally make another queue manager the default and want to revert to the original default queue manager, edit the `DefaultQueueManager` stanza in mqs.ini, replacing the unwanted default queue manager with that of the one you want.

What to do next

See "Changing IBM WebSphere MQ and queue manager configuration information" on page 407 for information about configuration files.

Backing up configuration files after creating a queue manager

IBM WebSphere MQ configuration information is stored in configuration files on Windows, UNIX and Linux systems.

On Windows and Linux (x86 and x86-64) systems use IBM WebSphere MQ Explorer to make changes to the configuration files.

On Windows systems you can also use the `amqmdain` command to make changes to the configuration files. See, amqmdain

There are two types of configuration file:

- When you install the product, the IBM WebSphere MQ configuration file (mqs.ini) is created. It contains a list of queue managers that is updated each time you create or delete a queue manager. There is one mqs.ini file per node.
- When you create a new queue manager, a new queue manager configuration file (qm.ini) is automatically created. This contains configuration parameters for the queue manager.

After creating a queue manager, back up your configuration files. Then, if you create another queue manager that causes you problems, you can reinstate the backups when you have removed the source of the problem. As a general rule, back up your configuration files each time you create a new queue manager.

For more information about configuration files, see "Changing IBM WebSphere MQ and queue manager configuration information" on page 407.

Starting a queue manager

When you create a queue manager, you must start it to enable it to process commands or MQI calls.

To start a queue manager, use the **strmqm** command.

Note: You must use the **strmqm** command from the installation associated with the queue manager that you are working with. You can find out which installation a queue manager is associated with using the `dspmq -o installation` command.

For example, to start a queue manager QMB enter the following command:

```
strmqm QMB
```

On WebSphere MQ for Windows and WebSphere MQ for Linux (x86 and x86-64 platforms) systems, you can start a queue manager as follows:

1. Open the IBM WebSphere MQ Explorer.
2. Select the queue manager from the Navigator View.
3. Click `Start`. The queue manager starts.

If the queue manager start-up takes more than a few seconds WebSphere MQ issues information messages intermittently detailing the start-up progress.

The `strmqm` command does not return control until the queue manager has started and is ready to accept connection requests.

Starting a queue manager automatically

In WebSphere MQ for Windows you can start a queue manager automatically when the system starts using the IBM WebSphere MQ Explorer. For more information, see Administration using the IBM WebSphere MQ Explorer.

Stopping a queue manager

There are three ways to stop a queue manager: a quiesced shutdown, and immediate shutdown, and a preemptive shutdown.

Use the **endmqm** command to stop a queue manager.

Note: You must use the **endmqm** command from the installation associated with the queue manager that you are working with. You can find out which installation a queue manager is associated with using the `dspmq -o installation` command.

For example, to stop a queue manager called QMB, enter the following command:

```
endmqm QMB
```

On WebSphere MQ for Windows and WebSphere MQ for Linux (x86 and x86-64 platforms) systems, you can stop a queue manager as follows:

1. Open the IBM WebSphere MQ Explorer.
2. Select the queue manager from the Navigator View.
3. Click `Stop...`. The End Queue Manager panel is displayed.
4. Select Controlled, or Immediate.
5. Click OK. The queue manager stops.

Quiesced shutdown

By default, the **endmqm** command performs a quiesced shutdown of the specified queue manager. This might take a while to complete. A quiesced shutdown waits until all connected applications have disconnected.

Use this type of shutdown to notify applications to stop. If you issue:

```
endmqm -c QMB
```

you are not told when all applications have stopped. (An `endmqm -c QMB` command is equivalent to an `endmqm QMB` command.)

However, if you issue:

```
endmqm -w QMB
```

the command waits until all applications have stopped and the queue manager has ended.

Immediate shutdown

For an immediate shutdown any current MQI calls are allowed to complete, but any new calls fail. This type of shutdown does not wait for applications to disconnect from the queue manager.

For an immediate shutdown, type:

```
endmqm -i QMB
```

Preemptive shutdown

Note: Do not use this method unless all other attempts to stop the queue manager using the **endmqm** command have failed. This method can have unpredictable consequences for connected applications.

If an immediate shutdown does not work, you must resort to a *preemptive* shutdown, specifying the `-p` flag. For example:

```
endmqm -p QMB
```

This stops the queue manager immediately. If this method still does not work, see Stopping a queue manager manually for an alternative solution.

For a detailed description of the **endmqm** command and its options, see endmqm.

If you have problems shutting down a queue manager

Problems in shutting down a queue manager are often caused by applications. For example, when applications:

- Do not check MQI return codes properly
- Do not request notification of a quiesce
- Terminate without disconnecting from the queue manager (by issuing an `MQDISC` call)

If a problem occurs when you stop the queue manager, you can break out of the **endmqm** command using Ctrl-C. You can then issue another **endmqm** command, but this time with a flag that specifies the type of shutdown that you require.

Restarting a queue manager

You can use the **strmqm** command to restart a queue manager, or, on IBM WebSphere MQ for Windows and IBM WebSphere MQ for Linux (x86 and x86-64 platforms) systems, restart a queue manager from IBM WebSphere MQ Explorer.

To restart a queue manager, type:

```
strmqm saturn.queue.manager
```

On IBM WebSphere MQ for Windows and IBM WebSphere MQ for Linux (x86 and x86-64 platforms) systems, you can restart a queue manager in the same way as starting it, as follows:

1. Open the IBM WebSphere MQ Explorer.
2. Select the queue manager from the Navigator View.
3. Click `Start`. The queue manager restarts.

If the queue manager restart takes more than a few seconds IBM WebSphere MQ issues information messages intermittently detailing the start-up progress.

Deleting a queue manager

You can delete a queue manager using the **dltmqm** command or by using the WebSphere MQ Explorer.

Before you begin

Stop the queue manager.

Procedure

- Issue the following command: `dltmqm QMB`

  Note: You must use the **dltmqm** command from the installation associated with the queue manager that you are working with. You can find out which installation a queue manager is associated with using the `dspmq -o installation` command.

Steps for deleting a queue manager

About this task

On WebSphere MQ for Windows and WebSphere MQ for Linux (x86 and x86-64 platforms) systems, you can delete a queue manager as follows:

Procedure

1. Open the WebSphere MQ Explorer.
2. In the Navigator view, select the queue manager.
3. If the queue manager is not stopped, stop it.
   a) Right-click the queue manager.
   b) Click **Stop**.
4. Right-click the queue manager.
5. Click **Delete**.

Results

The queue manager is deleted.

Attention:

- Deleting a queue manager is a drastic step, because you also delete all resources associated with the queue manager, including all queues and their messages and all object definitions. If you use the **dltmqm** command, there is no displayed prompt that allows you to change your mind; when you press the Enter key all the associated resources are lost.
- In WebSphere MQ for Windows, deleting a queue manager also removes the queue manager from the automatic startup list (described in “Starting a queue manager” on page 24). When the command has completed, a `WebSphere MQ queue manager ending` message is displayed; you are not told that the queue manager has been deleted.
- Deleting a cluster queue manager does not remove it from the cluster. See the note in the description of **dltmqm** for more information.

For a description of the **dltmqm** command and its options, see dltmqm. Ensure that only trusted administrators have the authority to use this command. (For information about security, see Setting up security on Windows, UNIX and Linux systems .)

Connecting applications using distributed queuing

This section provides more detailed information about intercommunication between WebSphere MQ installations, including queue definition, channel definition, triggering, and sync point procedures

Before reading this section it is helpful to have an understanding of channels, queues, and the other concepts introduced in Concepts of intercommunication .

Use the information in the following links to connect your applications using distributed queuing:

- “How to send a message to another queue manager” on page 49
- “Triggering channels” on page 66
- “Safety of messages” on page 64
- “IBM WebSphere MQ distributed-messaging techniques” on page 27
- “Introduction to distributed queue management” on page 47
- Windows UNIX Linux “Monitoring and controlling channels on UNIX, Linux, and Windows” on page 72

Related concepts

“Configuring connections between the client and server” on page 96
To configure the communication links between WebSphere MQ MQI clients and servers, decide on your communication protocol, define the connections at both ends of the link, start a listener, and define channels.

“Changing IBM WebSphere MQ and queue manager configuration information” on page 407
Change the behavior of IBM WebSphere MQ or an individual queue manager to suit the needs of your installation.

Related tasks

“Configuring a queue manager cluster” on page 154
Use the links in this topic to find out how clusters work, how to design a cluster configuration, and to get an example of how to set up a simple cluster.

IBM WebSphere MQ distributed-messaging techniques

The subtopics in this section describe techniques that are of use when planning channels. These subtopics describe techniques to help you plan how to connect your queue managers together, and manage the flow of messages between your applications.

For message channel planning examples, see:

- Message channel planning example for distributed platforms

Related concepts

“Connecting applications using distributed queuing” on page 27
This section provides more detailed information about intercommunication between WebSphere MQ installations, including queue definition, channel definition, triggering, and sync point procedures

Channels

Introduction to message queuing

Concepts of intercommunication

Related reference

Example configuration information

Message flow control

Message flow control is a task that involves the setting up and maintenance of message routes between queue managers. It is important for routes that multi-hop through many queue managers. This section describes how you use queues, alias queue definitions, and message channels on your system to achieve message flow control.

You control message flow using a number of techniques that were introduced in “Connecting applications using distributed queuing” on page 27. If your queue manager is in a cluster, message flow is controlled using different techniques, as described in “Message flow control” on page 28.

You can use the following objects to achieve message flow control:

- Transmission queues
- Message channels
- Remote queue definition
- Queue manager alias definition
- Reply-to queue alias definition

The queue manager and queue objects are described in Objects . Message channels are described in Distributed queuing components . The following techniques use these objects to create message flows in your system:

- Putting messages to remote queues
- Routing by way of particular transmission queues
- Receiving messages
- Passing messages through your system
- Separating message flows
- Switching a message flow to another destination
- Resolving the reply-to queue name to an alias name

Note

All the concepts described in this section are relevant for all nodes in a network, and include sending and receiving ends of message channels. For this reason, only one node is illustrated in most examples. The exception is where the example requires explicit cooperation by the administrator at the other end of a message channel.

Before proceeding to the individual techniques, it is useful to recap on the concepts of name resolution and the three ways of using remote queue definitions. See Concepts of intercommunication.

Related concepts

“Queue names in transmission header” on page 28
Destination queue names travel with the message in the transmission header until the destination queue has been reached.

“How to create queue manager and reply-to aliases” on page 29
This topic explains the three ways that you can create a remote queue definition.

Queue names in transmission header

Destination queue names travel with the message in the transmission header until the destination queue has been reached.

The queue name used by the application, the logical queue name, is resolved by the queue manager to the destination queue name. In other words, the physical queue name. This destination queue name travels with the message in a separate data area, the transmission header, until the destination queue has been reached. The transmission header is then stripped off.

You change the queue manager part of this queue name when you create parallel classes of service. Remember to return the queue manager name to the original name when the end of the class-of-service diversion has been reached.

How to create queue manager and reply-to aliases

This topic explains the three ways that you can create a remote queue definition.

The remote queue definition object is used in three different ways. Table 3 on page 29 explains how to define each of the three ways:

- Using a remote queue definition to redefine a local queue name.

  The application provides only the queue name when opening a queue, and this queue name is the name of the remote queue definition.

  The remote queue definition contains the names of the target queue and queue manager. Optionally, the definition can contain the name of the transmission queue to be used. If no transmission queue name is provided, the queue manager uses the queue manager name, taken from the remote queue definition, for the transmission queue name. If a transmission queue of this name is not defined, but a default transmission queue is defined, the default transmission queue is used.
- Using a remote queue definition to redefine a queue manager name.

  The application, or channel program, provides a queue name together with the remote queue manager name when opening the queue.

  If you have provided a remote queue definition with the same name as the queue manager name, and you have left the queue name in the definition blank, then the queue manager substitutes the queue manager name in the open call with the queue manager name in the definition.

  In addition, the definition can contain the name of the transmission queue to be used. If no transmission queue name is provided, the queue manager takes the queue manager name, taken from the remote queue definition, for the transmission queue name. If a transmission queue of this name is not defined, but a default transmission queue is defined, the default transmission queue is used.
- Using a remote queue definition to redefine a reply-to queue name.

  Each time an application puts a message to a queue, it can provide the name of a reply-to queue for answer messages but with the queue manager name blank.

  If you provide a remote queue definition with the same name as the reply-to queue then the local queue manager replaces the reply-to queue name with the queue name from your definition.

  You can provide a queue manager name in the definition, but not a transmission queue name.

Table 3. Three ways of using the remote queue definition object

Usage	**Queue manager name**	**Queue name**	**Transmission queue name**
1. Remote queue definition (on OPEN call)			
Supplied in the call	blank or local QM	(*) required	-
Supplied in the definition	required	required	optional
2. Queue manager alias (on OPEN call)			
Supplied in the call	(*) required and not local QM	required	-
Supplied in the definition	required	blank	optional
3. Reply-to queue alias (on PUT call)			
Supplied in the call	blank	(*) required	-
Supplied in the definition	optional	optional	blank

Table 3. Three ways of using the remote queue definition object (continued)			
Usage	**Queue manager name**	**Queue name**	**Transmission queue name**
Note: (*) means that this name is the name of the definition object			

For a formal description, see Queue name resolution.

Putting messages on remote queues

You can use remote queue definition objects to resolve a queue name to a transmission queue to an adjacent queue manager.

In a distributed-queuing environment, a transmission queue and channel are the focal point for all messages to a location whether the messages originate from applications in your local system, or arrive through channels from an adjacent system. Figure 2 on page 30 shows an application placing messages on a logical queue named 'QA_norm'. The name resolution uses the remote queue definition 'QA_norm' to select the transmission queue QMB. It then adds a transmission header to the messages stating 'QA_norm at QMB'.

Messages arriving from the adjacent system on 'Channel_back' have a transmission header with the physical queue name 'QA_norm at QMB', for example. These messages are placed unchanged on transmission queue QMB.

The channel moves the messages to an adjacent queue manager.

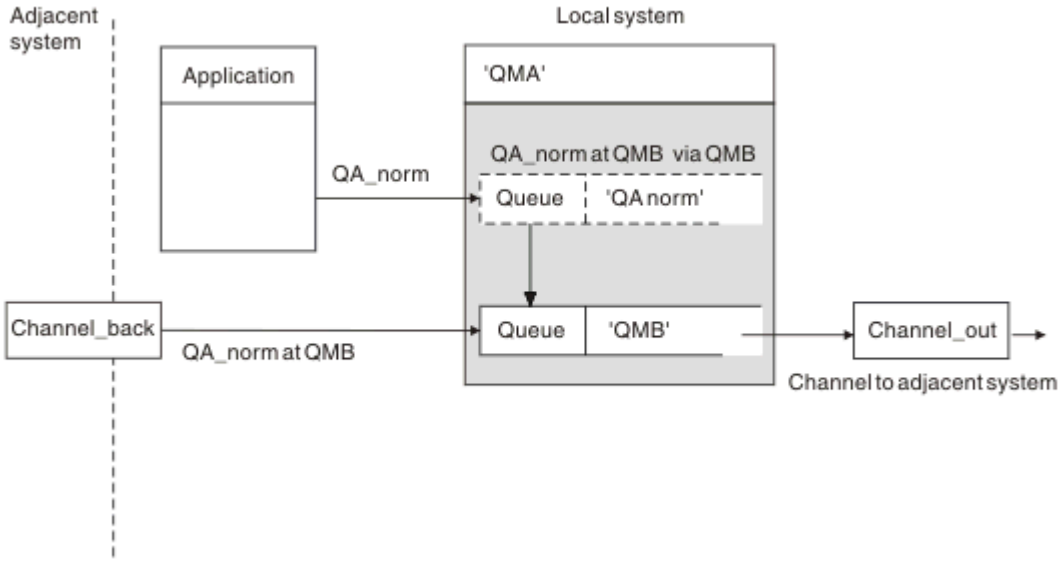


Figure 2. A remote queue definition is used to resolve a queue name to a transmission queue to an adjacent queue manager

If you are the WebSphere MQ system administrator, you must:

- Define the message channel from the adjacent system
- Define the message channel to the adjacent system
- Create the transmission queue QMB
- Define the remote queue object 'QA_norm' to resolve the queue name used by applications to the destination queue name, destination queue manager name, and transmission queue name

In a clustering environment, you only need to define a cluster-receiver channel at the local queue manager. You do not need to define a transmission queue or a remote queue object. For information, see Clusters .

More about name resolution

The effect of the remote queue definition is to define a physical destination queue name and queue manager name. These names are put in the transmission headers of messages.

Incoming messages from an adjacent system have already had this type of name resolution carried out by the original queue manager. Therefore they have the transmission header showing the physical destination queue name and queue manager name. These messages are unaffected by remote queue definitions.

Choosing the transmission queue

You can use a remote queue definition to allow a different transmission queue to send messages to the same adjacent queue manager.

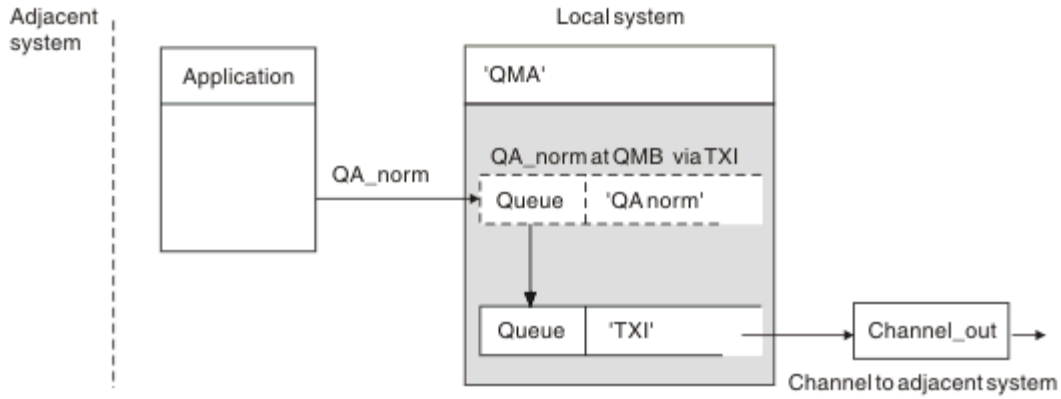


Figure 3. The remote queue definition allows a different transmission queue to be used

In a distributed-queuing environment, when you need to change a message flow from one channel to another, use the same system configuration as shown in Figure 2 on page 30 in "Putting messages on remote queues" on page 30. Figure 3 on page 31 in this topic shows how you use the remote queue definition to send messages over a different transmission queue, and therefore over a different channel, to the same adjacent queue manager.

For the configuration shown in Figure 3 on page 31, you must provide the remote queue object 'QA_norm', and the transmission queue 'TX1'. You must provide 'QA_norm' to choose the 'QA_norm' queue at the remote queue manager, the transmission queue 'TX1', and the queue manager 'QMB_priority'. Specify 'TX1' in the definition of the channel adjacent to the system.

Messages are placed on transmission queue 'TX1' with a transmission header containing 'QA_norm at QMB_priority', and are sent over the channel to the adjacent system.

The channel_back has been left out of this illustration because it would need a queue manager alias.

In a clustering environment, you do not need to define a transmission queue or a remote queue definition. For more information, see "Cluster queues" on page 161.

Receiving messages

You can configure the queue manager to receive messages from other queue managers. You must ensure that unintentional name resolution does not occur.

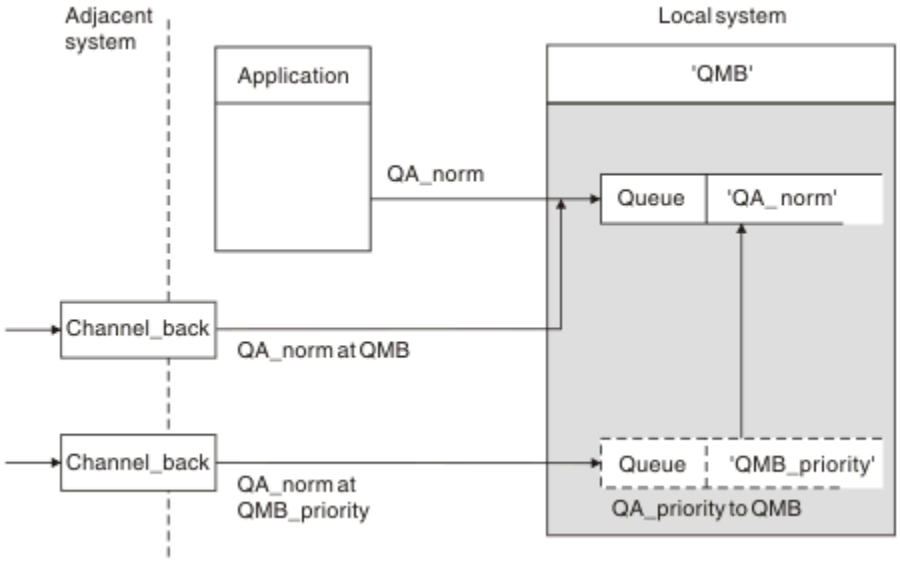


Figure 4. Receiving messages directly, and resolving alias queue manager name

As well as arranging for messages to be sent, the system administrator must also arrange for messages to be received from adjacent queue managers. Received messages contain the physical name of the destination queue manager and queue in the transmission header. They are treated the same as messages from a local application that specifies both queue manager name and queue name. Because of this treatment, you need to ensure that messages entering your system do not have an unintentional name resolution carried out. See Figure 4 on page 32 for this scenario.

For this configuration, you must prepare:

- Message channels to receive messages from adjacent queue managers
- A queue manager alias definition to resolve an incoming message flow, 'QMB_priority', to the local queue manager name, 'QMB'
- The local queue, 'QA_norm', if it does not exist

Receiving alias queue manager names

The use of the queue manager alias definition in this illustration has not selected a different destination queue manager. Messages passing through this local queue manager and addressed to 'QMB_priority' are intended for queue manager 'QMB'. The alias queue manager name is used to create the separate message flow.

Passing messages through your system

You can pass messages through your system in three ways - using the location name, using an alias for the queue manager, or selecting a transmission queue.

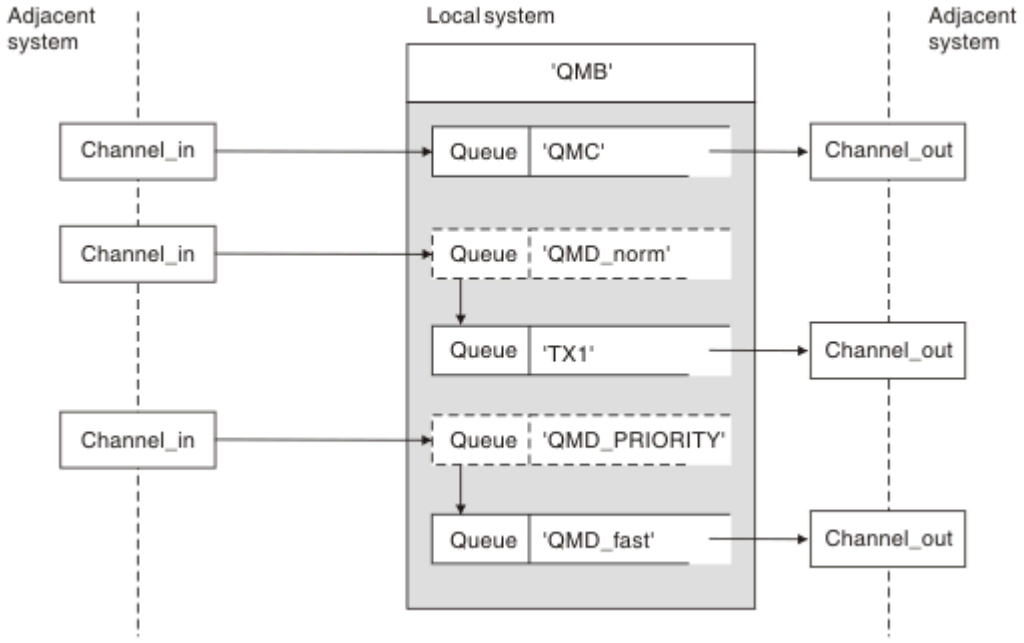


Figure 5. Three methods of passing messages through your system

The technique shown in Figure 4 on page 32 in “Receiving messages” on page 32, showed how an alias flow is captured. Figure 5 on page 33 illustrates the ways networks are built up by bringing together the techniques previously described.

The configuration shows a channel delivering three messages with different destinations:

1. QB at `QMC`
2. QB at `QMD_norm`
3. QB at `QMD_PRIORITY`

You must pass the first message flow through your system unchanged. You must pass the second message flow through a different transmission queue and channel. For the second message flow you must also resolve messages for the alias queue manager name `QMD_norm` to the queue manager `QMD`. The third message flow chooses a different transmission queue without any other change.

In a clustering environment, messages are passed through a cluster transmission queue. Normally a single transmission queue, `SYSTEM.CLUSTER.TRANSMIT.QUEUE`, transfers all messages to all queue managers in all clusters that the queue manager is a member of; see A cluster of queue managers. You can define separate transmission queues for all or some of the queue managers in the clusters that the queue manager is a member of.

The following methods describe techniques applicable to a distributed-queuing environment.

Use these methods

For these configurations, you must prepare the:

- Input channel definitions
- Output channel definitions
- Transmission queues:
  - `QMC`

  - TX1
  - QMD_fast
- Queue manager alias definitions:
  - QMD_norm with QMD_norm to QMD through TX1
  - QMD_PRIORITY with QMD_PRIORITY to QMD_PRIORITY through QMD_fast

Note: None of the message flows shown in the example changes the destination queue. The queue manager name aliases provide separation of message flows.

Method 1: Use the incoming location name

You are going to receive messages with a transmission header containing another location name, such as QMC. The simplest configuration is to create a transmission queue with that name, QMC. The channel that services the transmission queue delivers the message unchanged to the next destination.

Method 2: Use an alias for the queue manager

The second method is to use the queue manager alias object definition, but specify a new location name, QMD, and a particular transmission queue, TX1. This action:

- Terminates the alias message flow setup by the queue manager name alias QMD_norm, that is, the named class of service QMD_norm.
- Changes the transmission headers on these messages from QMD_norm to QMD.

Method 3: Select a transmission queue

The third method is to have a queue manager alias object defined with the same name as the destination location, QMD_PRIORITY. Use the queue manager alias definition to select a particular transmission queue, QMD_fast, and therefore another channel. The transmission headers on these messages remain unchanged.

Separating message flows

You can use a queue manager alias to create separate message flows to send messages to the same queue manager.

In a distributed-queuing environment, the need to separate messages to the same queue manager into different message flows can arise for a number of reasons. For example:

- You might need to provide a separate flow for large, medium, and small messages. This need also applies in a clustering environment and, in this case, you can create clusters that overlap. There are a number of reasons you might do so, for example:
  - To allow different organizations to have their own administration.
  - To allow independent applications to be administered separately.
  - To create a class of service. For example, you could have a cluster called STAFF that is a subset of the cluster called STUDENTS. When you put a message to a queue advertised in the STAFF cluster, a restricted channel is used. When you put a message to a queue advertised in the STUDENTS cluster, either a general channel or a restricted channel can be used.
  - To create test and production environments.
- It might be necessary to route incoming messages by different paths from the path of the locally generated messages.
- Your installation might require to schedule the movement of messages at certain times (for example, overnight) and the messages then need to be stored in reserved queues until scheduled.

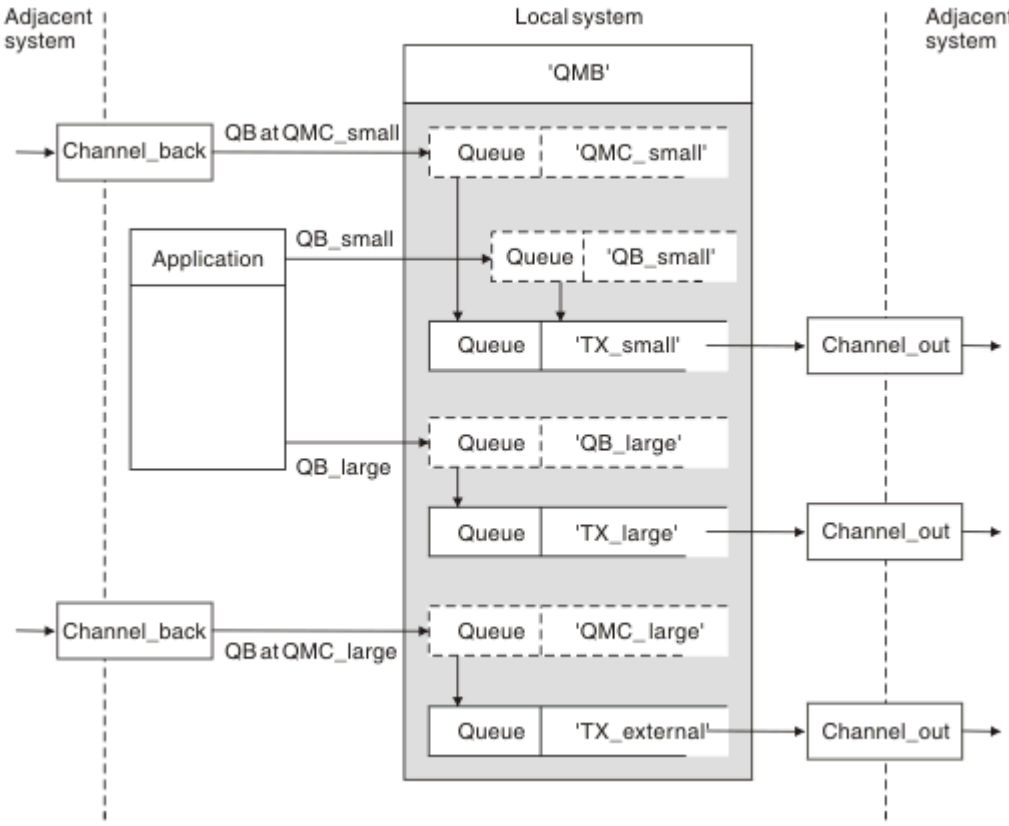


Figure 6. Separating messages flows

In the example shown in Figure 6 on page 35, the two incoming flows are to alias queue manager names 'QMC_small' and 'QMC_large'. You provide these flows with a queue manager alias definition to capture these flows for the local queue manager. You have an application addressing two remote queues and you need these message flows to be kept separate. You provide two remote queue definitions that specify the same location, 'QMC', but specify different transmission queues. This definition keeps the flows separate, and nothing extra is needed at the far end as they have the same destination queue manager name in the transmission headers. You provide:

- The incoming channel definitions
- The two remote queue definitions QB_small and QB_large
- The two queue manager alias definitions QMC_small and QMC_large
- The three sending channel definitions
- Three transmission queues: TX_small, TX_large, and TX_external

Coordination with adjacent systems

When you use a queue manager alias to create a separate message flow, you need to coordinate this activity with the system administrator at the remote end of the message channel to ensure that the corresponding queue manager alias is available there.

Concentrating messages to diverse locations

You can concentrate messages destined for various locations on to a single channel.

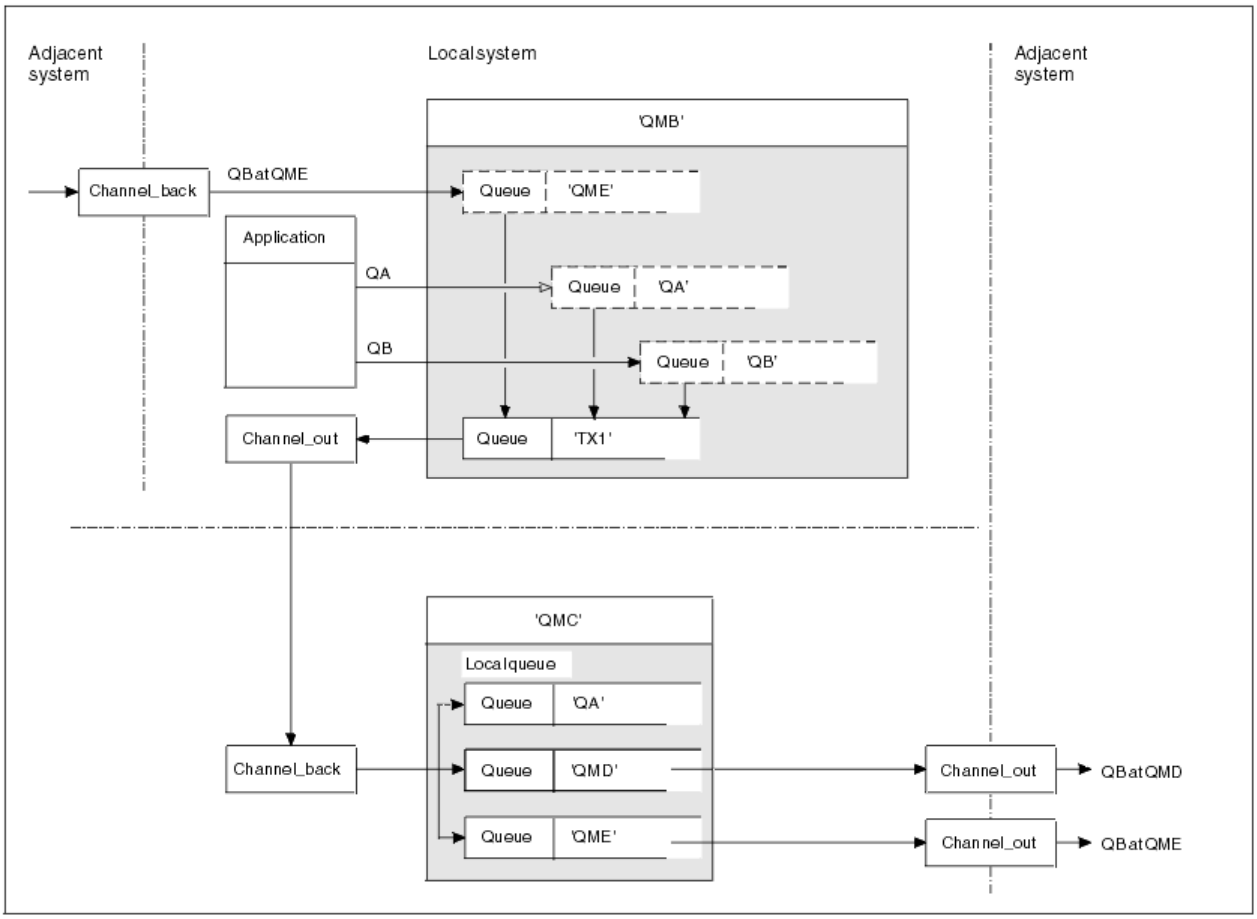


Figure 7. Combining message flows on to a channel

Figure 7 on page 36 illustrates a distributed-queuing technique for concentrating messages that are destined for various locations on to one channel. Two possible uses would be:

- Concentrating message traffic through a gateway
- Using wide bandwidth highways between nodes

In this example, messages from different sources, local and adjacent, and having different destination queues and queue managers, are flowed through transmission queue 'TX1' to queue manager QMC. Queue manager QMC delivers the messages according to the destinations. One set to a transmission queue 'QMD' for onward transmission to queue manager QMD. Another set to a transmission queue 'QME' for onward transmission to queue manager QME. Other messages are put on the local queue 'QA'.

You must provide:

- Channel definitions
- Transmission queue TX1
- Remote queue definitions:
  - QA with 'QA at QMC through TX1'
  - QB with 'QB at QMD through TX1'
- Queue manager alias definition:
  - QME with 'QME through TX1'

The complementary administrator who is configuring QMC must provide:

- Receiving channel definition with the same channel name
- Transmission queue QMD with associated sending channel definition
- Transmission queue QME with associated sending channel definition
- Local queue object QA.

Diverting message flows to another destination

You can redefine the destination of certain messages using queue manager aliases and transmission queues.

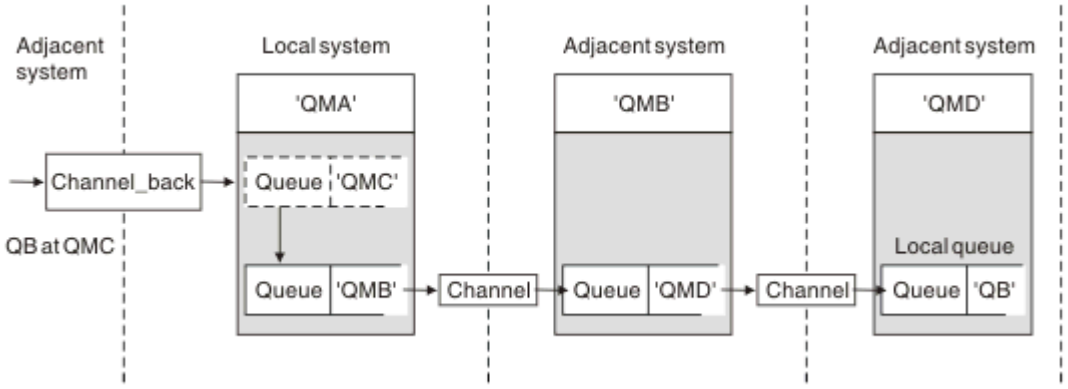


Figure 8. Diverting message streams to another destination

Figure 8 on page 37 illustrates how you can redefine the destination of certain messages. Incoming messages to QMA are destined for 'QB at QMC'. They normally arrive at QMA and be placed on a transmission queue called QMC which has been part of a channel to QMC. QMA must divert the messages to QMD, but is able to reach QMD only over QMB. This method is useful when you need to move a service from one location to another, and allow subscribers to continue to send messages on a temporary basis until they have adjusted to the new address.

The method of rerouting incoming messages destined for a certain queue manager to a different queue manager uses:

- A queue manager alias to change the destination queue manager to another queue manager, and to select a transmission queue to the adjacent system
- A transmission queue to serve the adjacent queue manager
- A transmission queue at the adjacent queue manager for onward routing to the destination queue manager

You must provide:

- Channel_back definition
- Queue manager alias object definition QMC with QB at QMD through QMB
- Channel_out definition
- The associated transmission queue QMB

The complementary administrator who is configuring QMB must provide:

- The corresponding channel_back definition
- The transmission queue, QMD
- The associated channel definition to QMD

You can use aliases within a clustering environment. For information, see “Queue-manager aliases and clusters” on page 254.

Sending messages to a distribution list

You can use a single MQPUT call to have an application send a message to several destinations.

In WebSphere MQ on all platforms except z/OS®, an application can send a message to several destinations with a single MQPUT call. You can do so in both a distributed-queuing environment and a clustering environment. You have to define the destinations in a distribution list, as described in Distribution lists.

Not all queue managers support distribution lists. When an MCA establishes a connection with a partner, it determines whether the partner supports distribution lists and sets a flag on the transmission queue accordingly. If an application tries to send a message that is destined for a distribution list but the partner does not support distribution lists, the sending MCA intercepts the message and puts it onto the transmission queue once for each intended destination.

A receiving MCA ensures that messages sent to a distribution list are safely received at all the intended destinations. If any destinations fail, the MCA establishes which ones have failed. It then can generate exception reports for them and can try to send the messages to them again.

Reply-to queue

You can create a complete remote queue processing loop using a reply-to queue.

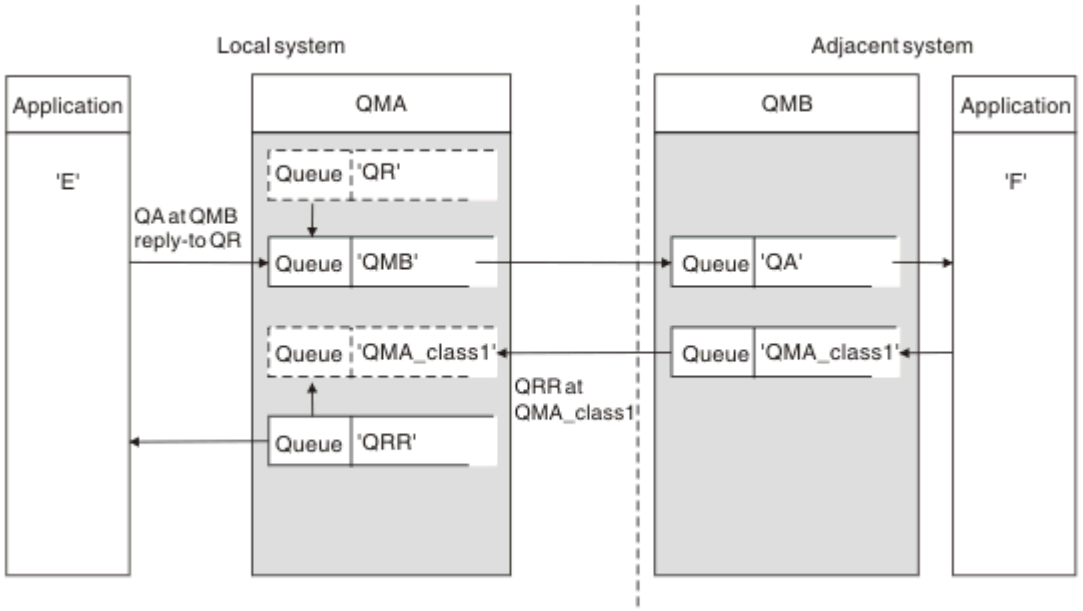


Figure 9. Reply-to queue name substitution during PUT call

A complete remote queue processing loop using a reply-to queue is shown in Figure 9 on page 38. This loop applies in both a distributed-queuing environment and a clustering environment. The details are as shown in Table 7 on page 45.

The application opens QA at QMB and puts messages on that queue. The messages are given a reply-to queue name of QR, without the queue manager name being specified. Queue manager QMA finds the reply-to queue object QR and extracts from it the alias name of QRR and the queue manager name QMA_class1. These names are put into the reply-to fields of the messages.

Reply messages from applications at QMB are addressed to QRR at QMA_class1. The queue manager alias name definition QMA_class1 is used by the queue manager to flow the messages to itself, and to queue QRR.

This scenario depicts the way you give applications the facility to choose a class of service for reply messages. The class is implemented by the transmission queue QMA_class1 at QMB, together with the queue manager alias definition, QMA_class1 at QMA. In this way, you can change an application's reply-to queue so that the flows are segregated without involving the application. The application always chooses

QR for this particular class of service. You have the opportunity to change the class of service with the reply-to queue definition QR.

You must create:

- Reply-to queue definition QR
- Transmission queue object QMB
- Channel_out definition
- Channel_back definition
- Queue manager alias definition QMA_class1
- Local queue object QRR, if it does not exist

The complementary administrator at the adjacent system must create:

- Receiving channel definition
- Transmission queue object QMA_class1
- Associated sending channel
- Local queue object QA.

Your application programs use:

- Reply-to queue name QR in put calls
- Queue name QRR in get calls

In this way, you can change the class of service as necessary, without involving the application. You change the reply-to alias 'QR', together with the transmission queue 'QMA_class1' and queue manager alias 'QMA_class1'.

If no reply-to alias object is found when the message is put on the queue, the local queue manager name is inserted in the blank reply-to queue manager name field. The reply-to queue name remains unchanged.

Name resolution restriction

Because the name resolution has been carried out for the reply-to queue at 'QMA' when the original message was put, no further name resolution is allowed at 'QMB'. The message is put with the physical name of the reply-to queue by the replying application.

The applications must be aware that the name they use for the reply-to queue is different from the name of the actual queue where the return messages are to be found.

For example, when two classes of service are provided for the use of applications with reply-to queue alias names of 'C1_alias', and 'C2_alias', the applications use these names as reply-to queue names in the message put calls. However, the applications actually expect messages to appear in queues 'C1' for 'C1_alias' and 'C2' for 'C2_alias'.

However, an application is able to make an inquiry call on the reply-to alias queue to check for itself the name of the real queue it must use to get the reply messages.

Related concepts

“How to create queue manager and reply-to aliases” on page 29
This topic explains the three ways that you can create a remote queue definition.

“Reply-to queue alias example” on page 40
This example illustrates the use of a reply-to alias to select a different route (transmission queue) for returned messages. The use of this facility requires the reply-to queue name to be changed in cooperation with the applications.

“How the example works” on page 41
An explanation of the example and how the queue manager uses the reply-to queue alias.

“Reply-to queue alias walk-through” on page 42

A walk-through of the process from an application putting a message on a remote queue through to the same application removing the reply message from the alias reply-to queue.

Reply-to queue alias example

This example illustrates the use of a reply-to alias to select a different route (transmission queue) for returned messages. The use of this facility requires the reply-to queue name to be changed in cooperation with the applications.

As shown in Figure 10 on page 40, the return route must be available for the reply messages, including the transmission queue, channel, and queue manager alias.

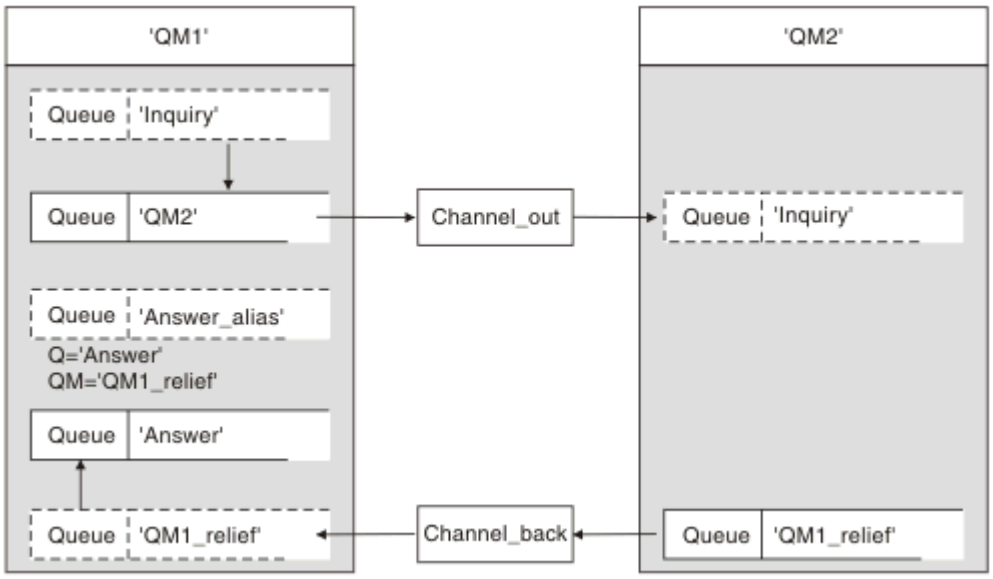


Figure 10. Reply-to queue alias example

This example is for requester applications at 'QM1' that send messages to server applications at 'QM2'. The messages on the server are to be returned through an alternative channel using transmission queue 'QM1_relief' (the default return channel would be served with a transmission queue 'QM1').

The reply-to queue alias is a particular use of the remote queue definition named 'Answer_alias'. Applications at QM1 include this name, 'Answer_alias', in the reply-to field of all messages that they put on queue 'Inquiry'.

Reply-to queue definition 'Answer_alias' is defined as 'Answer at QM1_relief'. Applications at QM1 expect their replies to appear in the local queue named 'Answer'.

Server applications at QM2 use the reply-to field of received messages to obtain the queue and queue manager names for the reply messages to the requester at QM1.

Definitions used in this example at QM1

The WebSphere MQ system administrator at QM1 must ensure that the reply-to queue 'Answer' is created along with the other objects. The name of the queue manager alias, marked with a '*', must agree with the queue manager name in the reply-to queue alias definition, also marked with an '*'.

Object	Definition	
Local transmission queue	QM2	
Remote queue definition	Object name	Inquiry
	Remote queue manager name	QM2
	Remote queue name	Inquiry
	Transmission queue name	QM2 (DEFAULT)

Object	Definition	
Queue manager alias	Object name	QM1_relief *
	Queue manager name	QM1
	Queue name	(blank)
Reply-to queue alias	Object name	Answer_alias
	Remote queue manager name	QM1_relief *
	Remote queue name	Answer

Put definition at QM1

Applications fill the reply-to fields with the reply-to queue alias name, and leave the queue manager name field blank.

Field	Content
Queue name	Inquiry
Queue manager name	(blank)
Reply-to queue name	Answer_alias
Reply-to queue manager	(blank)

Definitions used in this example at QM2

The WebSphere MQ system administrator at QM2 must ensure that the local queue exists for the incoming messages, and that the correctly named transmission queue is available for the reply messages.

Object	Definition
Local queue	Inquiry
Transmission queue	QM1_relief

Put definition at QM2

Applications at QM2 retrieve the reply-to queue name and queue manager name from the original message and use them when putting the reply message on the reply-to queue.

Field	Content
Queue name	Answer
Queue manager name	QM1_relief

How the example works

An explanation of the example and how the queue manager uses the reply-to queue alias.

In this example, requester applications at QM1 always use 'Answer_alias' as the reply-to queue in the relevant field of the put call. They always retrieve their messages from the queue named 'Answer'.

The reply-to queue alias definitions are available for use by the QM1 system administrator to change the name of the reply-to queue 'Answer', and of the return route 'QM1_relief'.

Changing the queue name 'Answer' is normally not useful because the QM1 applications are expecting their answers in this queue. However, the QM1 system administrator is able to change the return route (class of service), as necessary.

How the queue manager uses the reply-to queue alias

Queue manager QM1 retrieves the definitions from the reply-to queue alias when the reply-to queue name, included in the put call by the application, is the same as the reply-to queue alias, and the queue manager part is blank.

The queue manager replaces the reply-to queue name in the put call with the queue name from the definition. It replaces the blank queue manager name in the put call with the queue manager name from the definition.

These names are carried with the message in the message descriptor.

Table 4. Reply-to queue alias

Field name	**Put call**	**Transmission header**
Reply-to queue name	Answer_alias	Answer
Reply-to queue manager name	(blank)	QM1_relief

Reply-to queue alias walk-through

A walk-through of the process from an application putting a message on a remote queue through to the same application removing the reply message from the alias reply-to queue.

To complete this example, let us look at the process.

1. The application opens a queue named 'Inquiry', and puts messages to it. The application sets the reply-to fields of the message descriptor to:

Reply-to queue name	**Answer_alias**
Reply-to queue manager name	(blank)

2. Queue manager 'QM1' responds to the blank queue manager name by checking for a remote queue definition with the name 'Answer_alias'. If none is found, the queue manager places its own name, 'QM1', in the reply-to queue manager field of the message descriptor.
3. If the queue manager finds a remote queue definition with the name 'Answer_alias', it extracts the queue name and queue manager names from the definition (queue name='Answer' and queue manager name= 'QM1_relief'). It then puts them into the reply-to fields of the message descriptor.
4. The queue manager 'QM1' uses the remote queue definition 'Inquiry' to determine that the intended destination queue is at queue manager 'QM2', and the message is placed on the transmission queue 'QM2'. 'QM2' is the default transmission queue name for messages destined for queues at queue manager 'QM2'.
5. When queue manager 'QM1' puts the message on the transmission queue, it adds a transmission header to the message. This header contains the name of the destination queue, 'Inquiry', and the destination queue manager, 'QM2'.
6. The message arrives at queue manager 'QM2', and is placed on the 'Inquiry' local queue.
7. An application gets the message from this queue and processes the message. The application prepares a reply message, and puts this reply message on the reply-to queue name from the message descriptor of the original message:

Reply-to queue name	**Answer**
Reply-to queue manager name	QM1_relief

8. Queue manager 'QM2' carries out the put command. Finding that the queue manager name, 'QM1_relief', is a remote queue manager, it places the message on the transmission queue with the same name, 'QM1_relief'. The message is given a transmission header containing the name of the destination queue, 'Answer', and the destination queue manager, 'QM1_relief'.

9. The message is transferred to queue manager 'QM1'. The queue manager, recognizes that the queue manager name 'QM1_relief' is an alias, extracts from the alias definition 'QM1_relief' the physical queue manager name 'QM1'.
10. Queue manager 'QM1' then puts the message on the queue name contained in the transmission header, 'Answer'.
11. The application extracts its reply message from the queue 'Answer'.

Networking considerations

In a distributed-queuing environment, because message destinations are addressed with just a queue name and a queue manager name, certain rules apply.

1. Where the queue manager name is given, and the name is different from the local queue manager name:
   - A transmission queue must be available with the same name. This transmission queue must be part of a message channel moving messages to another queue manager, or
   - A queue manager alias definition must exist to resolve the queue manager name to the same, or another queue manager name, and optional transmission queue, or
   - If the transmission queue name cannot be resolved, and a default transmission queue has been defined, the default transmission queue is used.
2. Where only the queue name is supplied, a queue of any type but with the same name must be available on the local queue manager. This queue can be a remote queue definition which resolves to: a transmission queue to an adjacent queue manager, a queue manager name, and an optional transmission queue.

To see how this works in a clustering environment, see the appropriate topics in the How clusters work section of the product documentation.

Consider the scenario of a message channel moving messages from one queue manager to another in a distributed-queuing environment.

The messages being moved have originated from any other queue manager in the network, and some messages might arrive that have an unknown queue manager name as destination. This issue can occur when a queue manager name has changed or has been removed from the system, for example.

The channel program recognizes this situation when it cannot find a transmission queue for these messages, and places the messages on your undelivered-message (dead-letter) queue. It is your responsibility to look for these messages and arrange for them to be forwarded to the correct destination. Alternatively, return them to the originator, where the originator can be ascertained.

Exception reports are generated in these circumstances, if report messages were requested in the original message.

Name resolution convention

Name resolution that changes the identity of the destination queue (that is, logical to physical name changing), only occurs once, and only at the originating queue manager.

Subsequent use of the various alias possibilities must only be used when separating and combining message flows.

Return routing

Messages can contain a return address in the form of the name of a queue and queue manager. This return address form can be used in both a distributed-queuing environment and a clustering environment.

This address is normally specified by the application that creates the message. It can be modified by any application that then handles the message, including user exit applications.

Irrespective of the source of this address, any application handling the message might choose to use this address for returning answer, status, or report messages to the originating application.

The way these response messages are routed is not different from the way the original message is routed. You need to be aware that the message flows you create to other queue managers need corresponding return flows.

Physical name conflicts

The destination reply-to queue name has been resolved to a physical queue name at the original queue manager. It must not be resolved again at the responding queue manager.

It is a likely possibility for name conflict problems that can only be prevented by a network-wide agreement on physical and logical queue names.

Managing queue name translations

When you create a queue manager alias definition or a remote queue definition, the name resolution is carried out for every message carrying that name. This situation must be managed.

This description is provided for application designers and channel planners concerned with an individual system that has message channels to adjacent systems. It takes a local view of channel planning and control.

When you create a queue manager alias definition or a remote queue definition, the name resolution is carried out for every message carrying that name, regardless of the source of the message. To oversee this situation, which might involve large numbers of queues in a queue manager network, you keep tables of:

- The names of source queues and of source queue managers with respect to resolved queue names, resolved queue manager names, and resolved transmission queue names, with method of resolution
- The names of source queues with respect to:
  - Resolved destination queue names
  - Resolved destination queue manager names
  - Transmission queues
  - Message channel names
  - Adjacent system names
  - Reply-to queue names

Note: The use of the term *source* in this context refers to the queue name or the queue manager name provided by the application, or a channel program when opening a queue for putting messages.

An example of each of these tables is shown in Table 5 on page 44, Table 6 on page 45, and Table 7 on page 45.

The names in these tables are derived from the examples in this section, and this table is not intended as a practical example of queue name resolution in one node.

Table 5. Queue name resolution at queue manager QMA

Source queue specified when queue is opened	Source queue manager specified when queue is opened	Resolved queue name	Resolved queue manager name	Resolved transmission queue name	Resolution type
QA_norm	-	QA_norm	QMB	QMB	Remote queue
(any)	QMB	-	-	QMB	(none)
QA_norm	-	QA_norm	QMB	TX1	Remote queue

Table 5. Queue name resolution at queue manager QMA (continued)

Source queue specified when queue is opened	Source queue manager specified when queue is opened	Resolved queue name	Resolved queue manager name	Resolved transmission queue name	Resolution type
QB	QMC	QB	QMD	QMB	Queue manager alias

Table 6. Queue name resolution at queue manager QMB

Source queue specified when queue is opened	Source queue manager specified when queue is opened	Resolved queue name	Resolved queue manager name	Resolved transmission queue name	Resolution type
QA_norm	-	QA_norm	QMB	-	(none)
QA_norm	QMB	QA_norm	QMB	-	(none)
QA_norm	QMB_PRIORITY	QA_norm	QMB	-	Queue manager alias
(any)	QMC	(any)	QMC	QMC	(none)
(any)	QMD_norm	(any)	QMD_norm	TX1	Queue manager alias
(any)	QMD_PRIORITY	(any)	QMD_PRIORITY	QMD_fast	Queue manager alias
(any)	QMC_small	(any)	QMC_small	TX_small	Queue manager alias
(any)	QMC_large	(any)	QMC_large	TX_external	Queue manager alias
QB_small	QMC	QB_small	QMC	TX_small	Remote queue
QB_large	QMC	QB_large	QMC	TX_large	Remote queue
(any)	QME	(any)	QME	TX1	Queue manager alias
QA	QMC	QA	QMC	TX1	Remote queue
QB	QMD	QB	QMD	TX1	Remote queue

Table 7. Reply-to queue name translation at queue manager QMA

Application design		Reply-to alias definition	
Local QMGR	**Queue name for messages**	**Reply-to queue alias name**	**Redefined to**
QMA	QRR	QR	QRR at QMA_class1

Channel message sequence numbering

The channel uses sequence numbers to assure that messages are delivered, delivered without duplication, and stored in the same order as they were taken from the transmission queue.

The sequence number is generated at the sending end of the channel and is incremented by one before being used, which means that the current sequence number is the number of the last message sent. This information can be displayed using DISPLAY CHSTATUS (see MQSC reference). The sequence number and an identifier called the LUWID are stored in persistent storage for the last message transferred in a batch. These values are used during channel start-up to ensure that both ends of the link agree on which messages have been transferred successfully.

Sequential retrieval of messages

If an application puts a sequence of messages to the same destination queue, those messages can be retrieved in sequence by a ***single*** application with a sequence of MQGET operations, if the following conditions are met:

- All the put requests were done from the same application.
- All the put requests were either from the same unit of work, or all the put requests were made outside of a unit of work.
- The messages all have the same priority.
- The messages all have the same persistence.
- For remote queuing, the configuration is such that there can only be one path from the application making the put request, through its queue manager, through intercommunication, to the destination queue manager and the target queue.
- The messages are not put to a dead-letter queue (for example, if a queue is temporarily full).
- The application getting the message does not deliberately change the order of retrieval, for example by specifying a particular *MsgId* or *CorrelId* or by using message priorities.
- Only one application is doing get operations to retrieve the messages from the destination queue. If there is more than one application, these applications must be designed to get all the messages in each sequence put by a sending application.

Note: Messages from other tasks and units of work might be interspersed with the sequence, even where the sequence was put from within a single unit of work.

If these conditions cannot be met, and the order of messages on the target queue is important, then the application can be coded to use its own message sequence number as part of the message to assure the order of the messages.

Sequence of retrieval of fast, nonpersistent messages

Nonpersistent messages on a fast channel might overtake persistent messages on the same channel and so arrive out of sequence. The receiving MCA puts the nonpersistent messages on the destination queue immediately and makes them visible. Persistent messages are not made visible until the next sync point.

Loopback testing

Loopback testing is a technique on non- z/OS platforms that allows you to test a communications link without actually linking to another machine.

You set up a connection between two queue managers as though they are on separate machines, but you test the connection by looping back to another process on the same machine. This technique means that you can test your communications code without requiring an active network.

The way you do so depends on which products and protocols you are using.

On Windows systems, you can use the "loopback" adapter.

Refer to the documentation for the products you are using for more information.

Route tracing and activity recording

You can confirm the route a message takes through a series of queue managers in two ways.

You can use the WebSphere MQ display route application, available through the control command `dspmqrte`, or you can use activity recording. Both of these topics are described in Monitoring reference.

Introduction to distributed queue management

Distributed queue management (DQM) is used to define and control communication between queue managers.

Distributed queue management:

- Enables you to define and control communication channels between queue managers
- Provides you with a message channel service to move messages from a type of *local queue*, known as a transmission queue, to communication links on a local system, and from communication links to local queues at a destination queue manager
- Provides you with facilities for monitoring the operation of channels and diagnosing problems, using panels, commands, and programs

Channel definitions associate channel names with transmission queues, communication link identifiers, and channel attributes. Channel definitions are implemented in different ways on different platforms. Message sending and receiving is controlled by programs known as *message channel agents* (MCAs), which use the channel definitions to start up and control communication.

The MCAs in turn are controlled by DQM itself. The structure is platform-dependent, but typically includes listeners and trigger monitors, together with operator commands and panels.

A *message channel* is a one-way pipe for moving messages from one queue manager to another. Thus a message channel has two end-points, represented by a pair of MCAs. Each end point has a definition of its end of the message channel. For example, one end would define a sender, the other end a receiver.

For details of how to define channels, see:

- 

“Monitoring and controlling channels on UNIX, Linux, and Windows” on page 72

For message channel planning examples, see:

- 

Message channel planning example for distributed platforms

For information about channel exits, see Channel-exit programs for messaging channels.

Related concepts

“Message sending and receiving” on page 48
The following figure shows the distributed queue management model, detailing the relationships between entities when messages are transmitted. It also shows the flow for control.

“Channel control function” on page 52
The channel control function provides facilities for you to define, monitor, and control channels.

“What happens when a message cannot be delivered?” on page 65
When a message cannot be delivered, the MCA can process it in several ways. It can try again, it can return-to-sender, or it can put it on the dead-letter queue.

“Initialization and configuration files” on page 69
The handling of channel initialization data depends on your WebSphere MQ platform.

“Data conversion for messages” on page 70
WebSphere MQ messages might require data conversion when sent between queues on different queue managers.

“Writing your own message channel agents” on page 70

WebSphere MQ allows you to write your own message channel agent (MCA) programs or to install one from an independent software vendor.

"Other things to consider for distributed queue management" on page 71
Other topics to consider when preparing WebSphere MQ for distributed queue management. This topic covers Undelivered-message queue, Queues in use, System extensions and user-exit programs, and Running channels and listeners as trusted applications.

Related reference

Example configuration information

Message sending and receiving

The following figure shows the distributed queue management model, detailing the relationships between entities when messages are transmitted. It also shows the flow for control.

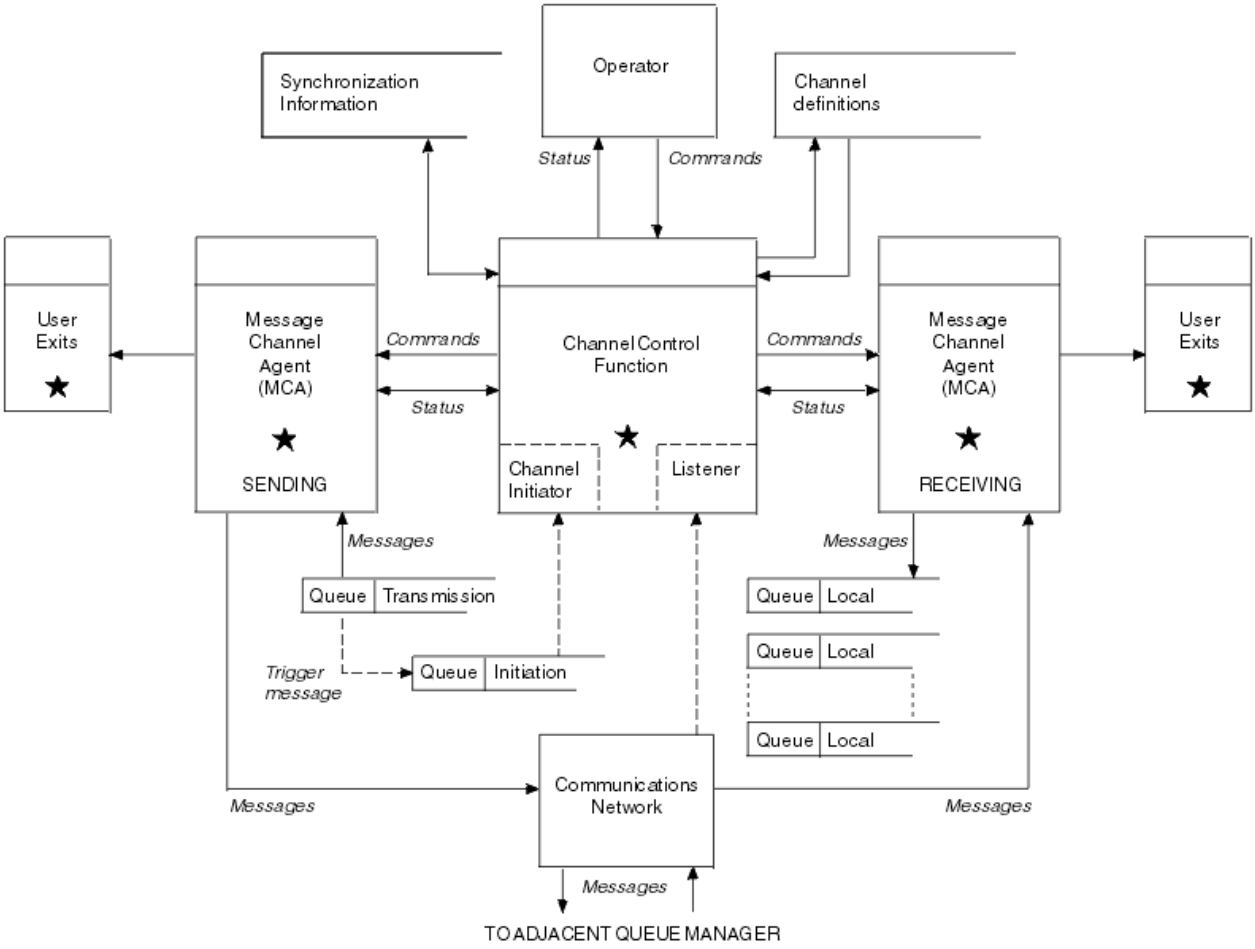


Figure 11. Distributed queue management model

Note:

1. There is one MCA per channel, depending on the platform. There might be one or more channel control functions for a particular queue manager.
2. The implementation of MCAs and channel control functions is highly platform-dependent. They can be programs or processes or threads, and they can be a single entity or many comprising several independent or linked parts.
3. All components marked with a star can use the MQI.

Channel parameters

An MCA receives its parameters in one of several ways:

- If started by a command, the channel name is passed in a data area. The MCA then reads the channel definition directly to obtain its attributes.
- For sender, and in some cases server channels, the MCA can be started automatically by the queue manager trigger. The channel name is retrieved from the trigger process definition, where applicable, and is passed to the MCA. The remaining processing is the same as previously described. Server channels must only be set up to trigger if they are fully qualified, that is, they specify a CONNAME to connect to.
- If started remotely by a sender, server, requester, or client-connection, the channel name is passed in the initial data from the partner message channel agent. The MCA reads the channel definition directly to obtain its attributes.

Certain attributes not defined in the channel definition are also negotiable:

Split messages
If one end does not support split messages then the split messages are not sent.

Conversion capability
If one end cannot perform the necessary code page conversion or numeric encoding conversion when needed, the other end must handle it. If neither end supports it, when needed, the channel cannot start.

Distribution list support
If one end does not support distribution lists, the partner MCA sets a flag in its transmission queue so that it knows to intercept messages intended for multiple destinations.

Channel status and sequence numbers

Message channel agent programs keep records of the current sequence number and logical unit of work number for each channel, and of the general status of the channel. Some platforms allow you to display this status information to help you control channels.

How to send a message to another queue manager

This section describes the simplest way to send a message between queue managers, including prerequisites and authorizations required. Other methods can also be used to send messages to a remote queue manager.

Before you send a message from one queue manager to another, you need to do the following steps:

1. Check that your chosen communication protocol is available.
2. Start the queue managers.
3. Start the channel initiators.
4. Start the listeners.

You also need to have the correct WebSphere MQ security authorization to create the objects required.

To send messages from one queue manager to another:

- Define the following objects on the source queue manager:
  - Sender channel
  - Remote queue definition
  - Initiation queue (optional)
  - Transmission queue
  - Dead-letter queue
- Define the following objects on the target queue manager:

- Receiver channel
- Target queue
- Dead-letter queue

You can use several different methods to define these objects, depending on your WebSphere MQ platform:

- On all platforms, you can use the WebSphere MQ script commands (MQSC) described in The MQSC commands the programmable command format (PCF) commands described in Automating administration tasks , or the WebSphere MQ Explorer.

See the following subtopics for more information on creating the components for sending messages to another queue manager:

Related concepts

"Creating and managing queue managers" on page 18
Before you can use messages and queues, you must create and start at least one queue manager and its associated objects.

"IBM WebSphere MQ distributed-messaging techniques" on page 27
The subtopics in this section describe techniques that are of use when planning channels. These subtopics describe techniques to help you plan how to connect your queue managers together, and manage the flow of messages between your applications.

"Introduction to distributed queue management" on page 47
Distributed queue management (DQM) is used to define and control communication between queue managers.

"Triggering channels" on page 66
WebSphere MQ provides a facility for starting an application automatically when certain conditions on a queue are met. This facility is called triggering.

"Safety of messages" on page 64
In addition to the typical recovery features of WebSphere MQ, distributed queue management ensures that messages are delivered properly by using a sync point procedure coordinated between the two ends of the message channel. If this procedure detects an error, it closes the channel so that you can investigate the problem, and keeps the messages safely in the transmission queue until the channel is restarted.

"Monitoring and controlling channels on UNIX, Linux, and Windows" on page 72
For DQM you need to create, monitor, and control the channels to remote queue managers. You can control channels using commands, programs, IBM WebSphere MQ Explorer, files for the channel definitions, and a storage area for synchronization information.

"Configuring connections between the client and server" on page 96
To configure the communication links between WebSphere MQ MQI clients and servers, decide on your communication protocol, define the connections at both ends of the link, start a listener, and define channels.

Related tasks

"Configuring a queue manager cluster" on page 154
Use the links in this topic to find out how clusters work, how to design a cluster configuration, and to get an example of how to set up a simple cluster.

Defining the channels

To send messages from one queue manager to another, you must define two channels. You must define one channel on the source queue manager and one channel on the target queue manager.

On the source queue manager

Define a channel with a channel type of SENDER. You need to specify the following:

- The name of the transmission queue to be used (the XMITQ attribute).
- The connection name of the partner system (the CONNAME attribute).

- The name of the communication protocol you are using (the TRPTYPE attribute). On WebSphere MQ for z/OS, the protocol must be TCP or LU6.2. On other platforms, you do not have to specify this. You can leave it to pick up the value from your default channel definition.

Details of all the channel attributes are given in Channel attributes .

On the target queue manager

Define a channel with a channel type of RECEIVER, and the same name as the sender channel.

Specify the name of the communication protocol you are using (the TRPTYPE attribute). On WebSphere MQ for z/OS, the protocol must be TCP or LU6.2. On other platforms, you do not have to specify this. You can leave it to pick up the value from your default channel definition.

Receiver channel definitions can be generic. This means that if you have several queue managers communicating with the same receiver, the sending channels can all specify the same name for the receiver, and one receiver definition applies to them all.

Note: The value of the TRPTYPE parameter is ignored by the responding message channel agent. For example, a TRPTYPE of TCP on the sender channel definition successfully starts with a TRPTYPE of LU62 on the receiver channel definition as a partner.

When you have defined the channel, you can test it using the PING CHANNEL command. This command sends a special message from the sender channel to the receiver channel and checks that it is returned.

Defining the queues

To send messages from one queue manager to another, you must define up to six queues. You must define up to four queues on the source queue manager, and up to two queues on the target queue manager.

On the source queue manager

- Remote queue definition

  In this definition, specify the following:

  Remote queue manager name
  The name of the target queue manager.

  Remote queue name
  The name of the target queue on the target queue manager.

  Transmission queue name
  The name of the transmission queue. You do not have to specify this transmission queue name. If you do not, a transmission queue with the same name as the target queue manager is used. If this does not exist, the default transmission queue is used. You are advised to give the transmission queue the same name as the target queue manager so that the queue is found by default.

- Initiation queue definition

  Required on z/OS, and optional on other platforms. Consider naming the initiation queue SYSTEM.CHANNEL.INITQ. on other platforms.

- Transmission queue definition

  A local queue with the USAGE attribute set to XMITQ.

- Dead-letter queue definition

  Define a dead-letter queue to which undelivered messages can be written.

On the target queue manager

- Local queue definition

  The target queue. The name of this queue must be the same as that specified in the remote queue name field of the remote queue definition on the source queue manager.

- Dead-letter queue definition

Define a dead-letter queue to which undelivered messages can be written.

Related concepts

“Creating a transmission queue” on page 52
Before a channel (other than a requester channel) can be started, the transmission queue must be defined as described in this section. The transmission queue must be named in the channel definition.

Creating a transmission queue

Before a channel (other than a requester channel) can be started, the transmission queue must be defined as described in this section. The transmission queue must be named in the channel definition.

Define a local queue with the USAGE attribute set to XMITQ for each sending message channel. If you want to use a specific transmission queue in your remote queue definitions, create a remote queue as shown.

To create a transmission queue, use the WebSphere MQ Commands (MQSC), as shown in the following examples:

Create transmission queue example

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

Create remote queue example

```
DEFINE QREMOTE(PAYROLL) DESCR('Remote queue for QM2') +
XMITQ(QM2) RNAME(PAYROLL) RQMNAME(QM2)
```

Consider naming the transmission queue the queue manager name on the remote system, as shown in the examples.

Starting the channel

When you put messages on the remote queue defined at the source queue manager, they are stored on the transmission queue until the channel is started. When the channel has been started, the messages are delivered to the target queue on the remote queue manager.

Start the channel on the sending queue manager using the START CHANNEL command. When you start the sending channel, the receiving channel is started automatically (by the listener) and the messages are sent to the target queue. Both ends of the message channel must be running for messages to be transferred.

Because the two ends of the channel are on different queue managers, they could have been defined with different attributes. To resolve any differences, there is an initial data negotiation between the two ends when the channel starts. In general, the two ends of the channel operate with the attributes needing the fewer resources. This enables larger systems to accommodate the lesser resources of smaller systems at the other end of the message channel.

The sending MCA splits large messages before sending them across the channel. They are reassembled at the remote queue manager. This is not apparent to the user.

An MCA can transfer messages using multiple threads. This process, called *pipelining* enables the MCA to transfer messages more efficiently, with fewer wait states. Pipelining improves channel performance.

Channel control function

The channel control function provides facilities for you to define, monitor, and control channels.

Commands are issued through panels, programs, or from a command line to the channel control function. The panel interface also displays channel status and channel definition data. You can use Programmable Command Formats or those WebSphere MQ commands (MQSC) and control commands that are detailed in “Monitoring and controlling channels on UNIX, Linux, and Windows” on page 72.

The commands fall into the following groups:

- Channel administration

- Channel control
- Channel status monitoring

Channel administration commands deal with the definitions of the channels. They enable you to:

- Create a channel definition
- Copy a channel definition
- Alter a channel definition
- Delete a channel definition

Channel control commands manage the operation of the channels. They enable you to:

- Start a channel
- Stop a channel
- Re-synchronize with partner (in some implementations)
- Reset message sequence numbers
- Resolve an in-doubt batch of messages
- Ping; send a test communication across the channel

Channel monitoring displays the state of channels, for example:

- Current channel settings
- Whether the channel is active or inactive
- Whether the channel terminated in a synchronized state

For more information about defining, controlling and monitoring channels, see the following subtopics:

Preparing channels

Before trying to start a message channel or MQI channel, you must prepare the channel. You must make sure that all the attributes of the local and remote channel definitions are correct and compatible.

Channel attributes describes the channel definitions and attributes.

Although you set up explicit channel definitions, the channel negotiations carried out when a channel starts, might override one or other of the values defined. This behavior is normal, and not apparent to the user, and has been arranged in this way so that otherwise incompatible definitions can work together.

Auto-definition of receiver and server-connection channels

In WebSphere MQ on all platforms except z/OS, if there is no appropriate channel definition, then for a receiver or server-connection channel that has auto-definition enabled, a definition is created automatically. The definition is created using:

1. The appropriate model channel definition, SYSTEM.AUTO.RECEIVER, or SYSTEM.AUTO.SVRCONN. The model channel definitions for auto-definition are the same as the system defaults, SYSTEM.DEF.RECEIVER, and SYSTEM.DEF.SVRCONN, except for the description field, which is "Auto-defined by" followed by 49 blanks. The systems administrator can choose to change any part of the supplied model channel definitions.
2. Information from the partner system. The values from the partner are used for the channel name and the sequence number wrap value.
3. A channel exit program, which you can use to alter the values created by the auto-definition. See Channel auto-definition exit program.

The description is then checked to determine whether it has been altered by an auto-definition exit or because the model definition has been changed. If the first 44 characters are still "Auto-defined by" followed by 29 blanks, the queue manager name is added. If the final 20 characters are still all blanks the local time and date are added.

When the definition has been created and stored the channel start proceeds as though the definition had always existed. The batch size, transmission size, and message size are negotiated with the partner.

Defining other objects

Before a message channel can be started, both ends must be defined (or enabled for auto-definition) at their queue managers. The transmission queue it is to serve must be defined to the queue manager at the sending end. The communication link must be defined and available. It might be necessary for you to prepare other WebSphere MQ objects, such as remote queue definitions, queue manager alias definitions, and reply-to queue alias definitions, to implement the scenarios described in “Connecting applications using distributed queuing” on page 27.

For information about defining MQI channels, see “Defining MQI channels” on page 109.

Multiple message channels per transmission queue

It is possible to define more than one channel per transmission queue, but only one of these channels can be active at any one time. Consider this option for the provision of alternative routes between queue managers for traffic balancing and link failure corrective action. A transmission queue cannot be used by another channel if the previous channel to use it terminated leaving a batch of messages in-doubt at the sending end. For more information, see “In-doubt channels” on page 63.

Starting a channel

A channel can be caused to start transmitting messages in one of four ways. It can be:

- Started by an operator (not receiver, cluster-receiver, or server-connection channels).
- Triggered from the transmission queue. This method applies to sender channels and fully qualified server channels (those channels which specify a CONNAME) only. You must prepare the necessary objects for triggering channels.
- Started from an application program (not receiver, cluster-receiver, or server-connection channels).
- Started remotely from the network by a sender, cluster-sender, requester, server, or client-connection channel. Receiver, cluster-receiver, and possibly server and requester channel transmissions, are started this way; so are server-connection channels. The channels themselves must already be started (that is, enabled).

Note: Because a channel is 'started' it is not necessarily transmitting messages. Instead, it might be 'enabled' to start transmitting when one of the four events previously described occurs. The enabling and disabling of a channel is achieved using the START and STOP operator commands.

Channel states

A channel can be in one of many states at any time. Some states also have substates. From a given state a channel can move into other states.

Figure 12 on page 55 shows the hierarchy of all possible channel states and the substates that apply to each of the channel states.

Figure 13 on page 56 shows the links between channel states. These links apply to all types of message channel and server-connection channels.

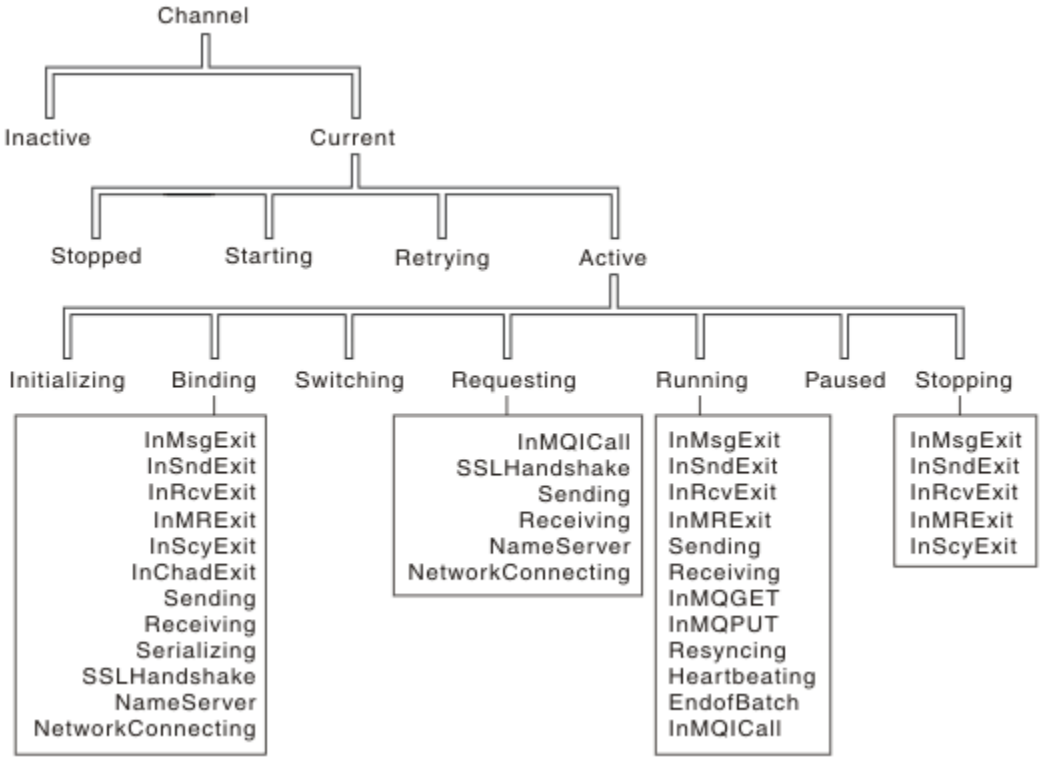


Figure 12. Channel states and substates

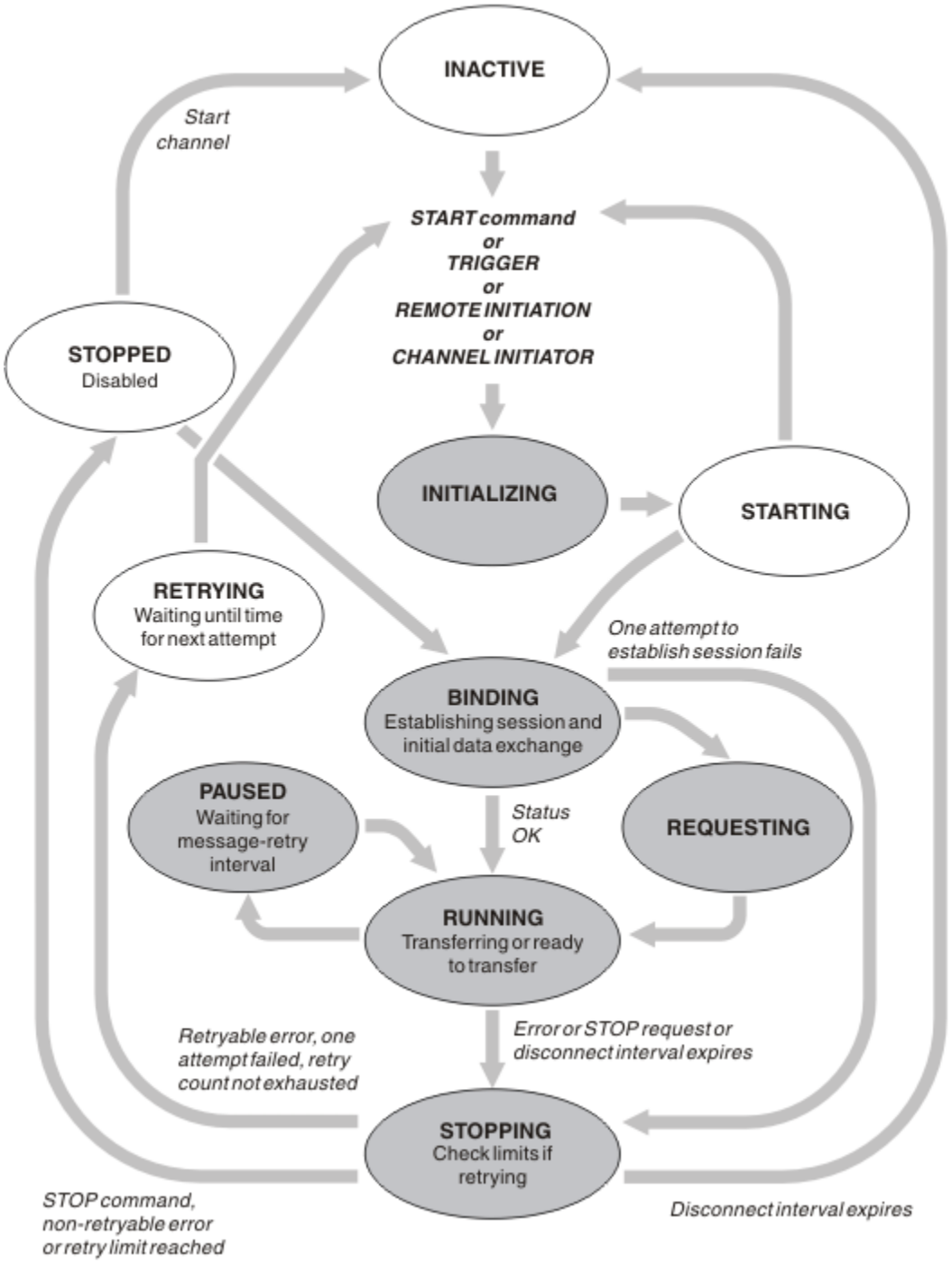


Figure 13. Flows between channel states

Current and active

A channel is *current* if it is in any state other than inactive. A current channel is *active* unless it is in RETRYING, STOPPED, or STARTING state. When a channel is active, it is consuming resource and a process or thread is running. The seven possible states of an active channel (INITIALIZING, BINDING, SWITCHING, REQUESTING, RUNNING, PAUSED, or STOPPING) are highlighted in Figure 13 on page 56.

An active channel can also show a substate giving more detail of exactly what the channel is doing. The substates for each state are shown in Figure 12 on page 55.

Current and active
The channel is "current" if it is in any state other than inactive. A current channel is "active" unless it is in RETRYING, STOPPED, or STARTING state.

If a channel is "active" it might also show a substate giving more detail of exactly what the channel is doing.

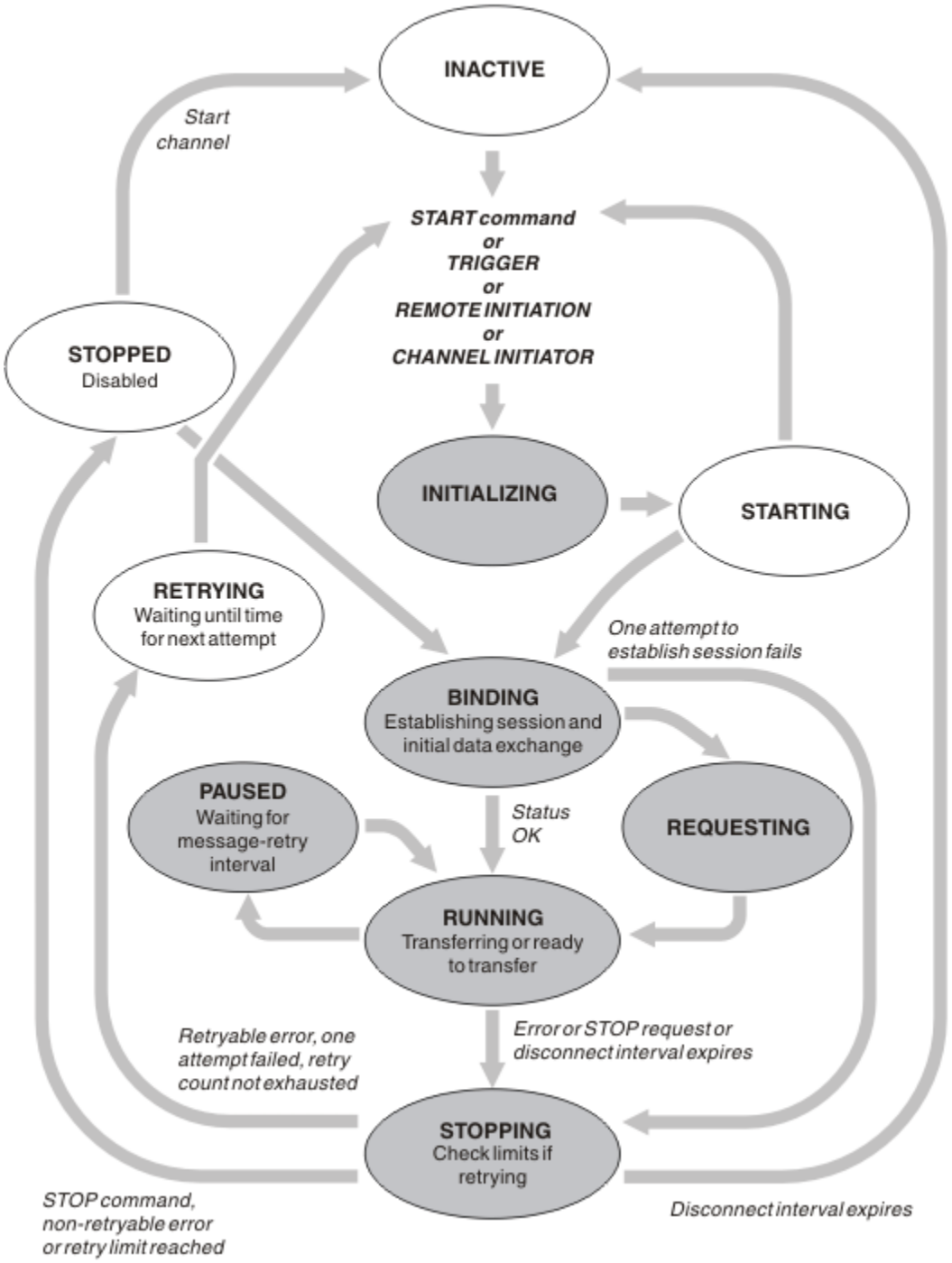


Figure 14. Flows between channel states

Note:

1. When a channel is in one of the six states highlighted in Figure 14 on page 57 (INITIALIZING, BINDING, REQUESTING, RUNNING, PAUSED, or STOPPING), it is consuming resource and a process or thread is running; the channel is *active*.

2. When a channel is in STOPPED state, the session might be active because the next state is not yet known.

Specifying the maximum number of current channels

You can specify the maximum number of channels that can be current at one time. This number is the number of channels that have entries in the channel status table, including channels that are retrying and channels that are stopped. Specify this using the queue manager configuration file for UNIX and Linux systems, or the WebSphere MQ Explorer. For more information about the values set using the initialization or the configuration file see Configuration file stanzas for distributed queuing. For more information about specifying the maximum number of channels, see Administering IBM WebSphere MQ for WebSphere MQ for UNIX and Linux systems, and Windows systems.

Note:

1. Server-connection channels are included in this number.
2. A channel must be current before it can become active. If a channel is started, but cannot become current, the start fails.

Specifying the maximum number of active channels

You can also specify the maximum number of active channels to prevent your system being overloaded by many starting channels. If you use this method, set the disconnect interval attribute to a low value to allow waiting channels to start as soon as other channels terminate.

Each time a channel that is retrying attempts to establish connection with its partner, it must become an active channel. If the attempt fails, it remains a current channel that is not active, until it is time for the next attempt. The number of times that a channel retries, and how often, is determined by the retry count and retry interval channel attributes. There are short and long values for both these attributes. See Channel attributes for more information.

When a channel has to become an active channel (because a START command has been issued, or because it has been triggered, or because it is time for another retry attempt), but is unable to do so because the number of active channels is already at the maximum value, the channel waits until one of the active slots is freed by another channel instance ceasing to be active. If, however, a channel is starting because it is being initiated remotely, and there are no active slots available for it at that time, the remote initiation is rejected.

Whenever a channel, other than a requester channel, is attempting to become active, it goes into the STARTING state. This state occurs even if there is an active slot immediately available, although it is only in the STARTING state for a short time. However, if the channel has to wait for an active slot, it is in STARTING state while it is waiting.

Requester channels do not go into STARTING state. If a requester channel cannot start because the number of active channels is already at the limit, the channel ends abnormally.

Whenever a channel, other than a requester channel, is unable to get an active slot, and so waits for one, a message is written to the log and an event is generated. When a slot is later freed and the channel is able to acquire it, another message and event are generated. Neither of these events and messages are generated if the channel is able to acquire a slot straight away.

If a STOP CHANNEL command is issued while the channel is waiting to become active, the channel goes to STOPPED state. A Channel-Stopped event is raised.

Server-connection channels are included in the maximum number of active channels.

For more information about specifying the maximum number of active channels, see Administering IBM WebSphere MQ for WebSphere MQ for UNIX and Linux systems, and Windows systems.

Channel errors
Errors on channels cause the channel to stop further transmissions. If the channel is a sender or server, it goes to RETRY state because it is possible that the problem might clear itself. If it cannot go to RETRY state, the channel goes to STOPPED state.

For sending channels, the associated transmission queue is set to GET(DISABLED) and triggering is turned off. (A STOP command with STATUS(STOPPED) takes the side that issued it to STOPPED state; only expiry of the disconnect interval or a STOP command with STATUS(INACTIVE) makes it end normally and become inactive.) Channels that are in STOPPED state need operator intervention before they can restart (see "Restarting stopped channels" on page 62).

Note: For UNIX, Linux and Windows systems, a channel initiator must be running for retry to be attempted. If the channel initiator is not available, the channel becomes inactive and must be manually restarted. If you are using a script to start the channel, ensure that the channel initiator is running before you try to run the script.

Long retry count (LONGRTY) describes how retrying works. If the error clears, the channel restarts automatically, and the transmission queue is re-enabled. If the retry limit is reached without the error clearing, the channel goes to STOPPED state. A stopped channel must be restarted manually by the operator. If the error is still present, it does not retry again. When it does start successfully, the transmission queue is re-enabled.

If the queue manager stops while a channel is in RETRYING or STOPPED status, the channel status is remembered when the queue manager is restarted. However, the channel status for the SVRCONN channel type is reset if the queue manager stops while the channel is in STOPPED status.

If a channel is unable to put a message to the target queue because that queue is full or put inhibited, the channel can retry the operation a number of times (specified in the message-retry count attribute) at a time interval (specified in the message-retry interval attribute). Alternatively, you can write your own message-retry exit that determines which circumstances cause a retry, and the number of attempts made. The channel goes to PAUSED state while waiting for the message-retry interval to finish.

See Channel attributes for information about the channel attributes, and Channel-exit programs for messaging channels for information about the message-retry exit.

Server-connection channel limits
You can set server-connection channel limits to prevent client applications from exhausting queue manager channel resources, **MAXINST**, and to prevent a single client application from exhausting server-connection channel capacity, **MAXINSTC**.

A maximum total number of channels can be active at any time on an individual queue manager. The total number of server-connection channel instances are included in the maximum number of active channels.

If you do not specify the maximum number of simultaneous instances of a server-connection channel that can be started, then it is possible for a single client application, connecting to a single server-connection channel, to exhaust the maximum number of active channels that are available. When the maximum number of active channels is reached, it prevents any other channels from being started on the queue manager. To avoid this situation, you must limit the number of simultaneous instances of an individual server-connection channel that can be started, regardless of which client started them.

If the value of the limit is reduced to below the currently running number of instances of the server connection channel, even to zero, then the running channels are not affected. New instances cannot be started until sufficient existing instances have ceased to run so that the number of currently running instances is less than the value of the limit.

Also, many different client-connection channels can connect to an individual server-connection channel. The limit on the number of simultaneous instances of an individual server-connection channel that can be started, regardless of which client started them, prevents any client from exhausting the maximum active channel capacity of the queue manager. If you do not also limit the number of simultaneous instances of an individual server-connection channel that can be started from an individual client, then it is possible for a single, faulty client application to open so many connections that it exhausts the channel capacity allocated to an individual server-connection channel, and therefore prevents other clients that need to

use the channel from connecting to it. To avoid this situation, you must limit the number of simultaneous instances of an individual server-connection channel that can be started from an individual client.

If the value of the individual client limit is reduced below the number of instances of the server-connection channel that are currently running from individual clients, even to zero, then the running channels are not affected. However, new instances of the server-connection channel cannot be started from an individual client that exceeds the new limit until sufficient existing instances from that client have ceased to run so that the number of currently running instances is less than the value of this parameter.

Checking that the other end of the channel is still available

You can use the heartbeat interval, the keep alive interval, and the receive timeout, to check that the other end of the channel is available.

Heartbeats

You can use the heartbeat interval channel attribute to specify that flows are to be passed from the sending MCA when there are no messages on the transmission queue, as is described in Heartbeat interval (HBINT).

Keep alive

In WebSphere MQ for UNIX, Linux, and Windows systems, if you are using TCP as your transport protocol, you can set `keepalive=yes`. If you specify this option, TCP periodically checks that the other end of the connection is still available. It is not, the channel is terminated. This option is described in Keepalive Interval (KAINT).

If you have unreliable channels that report TCP errors, use of the **Keepalive** option means that your channels are more likely to recover.

You can specify time intervals to control the behavior of the **Keepalive** option. When you change the time interval, only TCP/IP channels started after the change are affected. Ensure that the value that you choose for the time interval is less than the value of the disconnect interval for the channel.

For more information about using the **Keepalive** option, see the KAINT parameter in the DEFINE CHANNEL command.

Receive timeout

If you are using TCP as your transport protocol, the receiving end of an idle non-MQI channel connection is also closed if no data is received for a period. This period, the *receive time-out* value, is determined according to the HBINT (heartbeat interval) value.

In WebSphere MQ for UNIX, Linux, and Windows systems, the *receive time-out* value is set as follows:

1. For an initial number of flows, before any negotiation takes place, the *receive time-out* value is twice the HBINT value from the channel definition.
2. After the channels negotiate an HBINT value, if HBINT is set to less than 60 seconds, the *receive time-out* value is set to twice this value. If HBINT is set to 60 seconds or more, the *receive time-out* value is set to 60 seconds greater than the value of HBINT.

Note:

1. If either of the values is zero, there is no timeout.
2. For connections that do not support heartbeats, the HBINT value is negotiated to zero in step 2 and hence there is no timeout, so you must use TCP/IP KEEPALIVE.
3. For client connections that use sharing conversations, heartbeats can flow across the channel (from both ends) all the time, not just when an MQGET is outstanding.
4. For client connections where sharing conversations are not in use, heartbeats are flowed from the server only when the client issues an MQGET call with wait. Therefore, you are not recommended to set the heartbeat interval too small for client channels. For example, if the heartbeat is set to 10

seconds, an MQCMIT call fails (with MQRC_CONNECTION_BROKEN) if it takes longer than 20 seconds to commit because no data flowed during this time. This can happen with large units of work. However, it does not happen if appropriate values are chosen for the heartbeat interval because only MQGET with wait takes significant periods of time.

Provided SHARECNV is not zero, the client uses a full duplex connection, which means that the client can (and does) heartbeat during all MQI calls

5. In WebSphere MQ Version 7 Client channels, heartbeats can flow from both the server as well as the client side. The timeout at either end is based upon 2*HBINT for HBINTs of less than 60 seconds and HBINT+60 for HBINTs of over 60 seconds.
6. Canceling the connection after twice the heartbeat interval is valid because a data or heartbeat flow is expected at least at every heartbeat interval. Setting the heartbeat interval too small, however, can cause problems, especially if you are using channel exits. For example, if the HBINT value is one second, and a send or receive exit is used, the receiving end waits for only 2 seconds before canceling the channel. If the MCA is performing a task such as encrypting the message, this value might be too short.

Adopting an MCA

The Adopt MCA function enables IBM WebSphere MQ Explorer to cancel a receiver channel and start a new one in its place.

If a channel suffers a communications failure, the receiver channel could be left in a 'communications receive' state. When communications are re-established the sender channel attempts to reconnect. If the remote queue manager finds that the receiver channel is already running it does not allow another version of the same receiver channel to be started. This problem requires user intervention to rectify the problem or the use of system keepalive.

The Adopt MCA function solves the problem automatically. It enables IBM WebSphere MQ Explorer to cancel a receiver channel and to start a new one in its place.

The function can be set up with various options. distributed For distributed platforms, see Administering.

Stopping and quiescing channels

This topic explains how you can stop and quiesce a channel before the disconnect time interval expires.

Message channels are designed to be long-running connections between queue managers with orderly termination controlled only by the disconnect interval channel attribute. This mechanism works well unless the operator needs to terminate the channel before the disconnect time interval expires. This need can occur in the following situations:

- System quiesce
- Resource conservation
- Unilateral action at one end of a channel

In this case, you can stop the channel. You can do this using:

- the STOP CHANNEL MQSC command
- the Stop Channel PCF command
- the IBM WebSphere MQ Explorer

There are three options for stopping channels using these commands:

QUIESCE
The QUIESCE option attempts to end the current batch of messages before stopping the channel.

FORCE
The FORCE option attempts to stop the channel immediately and might require the channel to resynchronize when it restarts because the channel might be left in doubt.

TERMINATE
The TERMINATE option attempts to stop the channel immediately, and terminates the thread or process of the channel.

All these options leave the channel in a STOPPED state, requiring operator intervention to restart it.

Stopping the channel at the sending end is effective but does require operator intervention to restart. At the receiving end of the channel, things are much more difficult because the MCA is waiting for data from the sending side, and there is no way to initiate an *orderly* termination of the channel from the receiving side; the stop command is pending until the MCA returns from its wait for data.

Consequently there are three recommended ways of using channels, depending upon the operational characteristics required:

- If you want your channels to be long running, note that there can be orderly termination only from the sending end. When channels are interrupted, that is, stopped, operator intervention (a START CHANNEL command) is required in order to restart them.
- If you want your channels to be active only when there are messages for them to transmit, set the disconnect interval to a fairly low value. The default setting is high and so is not recommended for channels where this level of control is required. Because it is difficult to interrupt the receiving channel, the most economical option is to have the channel automatically disconnect and reconnect as the workload demands. For most channels, the appropriate setting of the disconnect interval can be established heuristically.
- You can use the heartbeat-interval attribute to cause the sending MCA to send a heartbeat flow to the receiving MCA during periods in which it has no messages to send. This action releases the receiving MCA from its wait state and gives it an opportunity to quiesce the channel without waiting for the disconnect interval to expire. Give the heartbeat interval a lower value than the value of the disconnect interval.

  Note:

  1. You are advised to set the disconnect interval to a low value, or to use heartbeats, for server channels. This low value is to allow for the case where the requester channel ends abnormally (for example, because the channel was canceled) when there are no messages for the server channel to send. If the disconnect interval is set high and heartbeats are not in use, the server does not detect that the requester has ended (which it will only do the next time it tries to send a message to the requester). While the server is still running, it holds the transmission queue open for exclusive input in order to get any more messages that arrive on the queue. If an attempt is made to restart the channel from the requester, the start request receives an error because the server still has the transmission queue open for exclusive input. It is necessary to stop the server channel, and then restart the channel from the requester again.

Restarting stopped channels

When a channel goes into STOPPED state, you have to restart the channel manually.

To do restart the channel, issue one of the following commands:

- The START CHANNEL MQSC command
- The Start Channel PCF command
- the IBM WebSphere MQ Explorer

For sender or server channels, when the channel entered the STOPPED state, the associated transmission queue was set to GET(DISABLED) and triggering was set off. When the start request is received, these attributes are reset automatically.

If the queue manager (on distributed platforms) stops while a channel is in RETRYING or STOPPED status, the channel status is remembered when the queue manager is restarted. However, the channel status for the SVRCONN channel type is reset if the queue manager stops while the channel is in STOPPED status.

In-doubt channels

An in-doubt channel is a channel that is in doubt with a remote channel about which messages have been sent and received.

Note the distinction between this and a queue manager being in doubt about which messages should be committed to a queue.

You can reduce the opportunity for a channel to be placed in doubt by using the Batch Heartbeat channel parameter (BATCHHB). When a value for this parameter is specified, a sender channel checks that the remote channel is still active before taking any further action. If no response is received the receiver channel is considered to be no longer active. The messages can be rolled-back, and re-routed, and the sender-channel is not put in doubt. This reduces the time when the channel could be placed in doubt to the period between the sender channel verifying that the receiver channel is still active, and verifying that the receiver channel has received the sent messages. See Channel attributes for more information about the batch heartbeat parameter.

In-doubt channel problems are typically resolved automatically. Even when communication is lost, and a channel is placed in doubt with a message batch at the sender with receipt status unknown, the situation is resolved when communication is re-established. Sequence number and LUWID records are kept for this purpose. The channel is in doubt until LUWID information has been exchanged, and only one batch of messages can be in doubt for the channel.

You can, when necessary, resynchronize the channel manually. The term *manual* includes use of operators or programs that contain WebSphere MQ system management commands. The manual resynchronization process works as follows. This description uses MQSC commands, but you can also use the PCF equivalents.

1. Use the DISPLAY CHSTATUS command to find the last-committed logical unit of work ID (LUWID) for ***each*** side of the channel. Do this using the following commands:
   - For the in-doubt side of the channel:

     ```
     DISPLAY CHSTATUS(name) SAVED CURLUWID
     ```

     You can use the CONNAME and XMITQ parameters to further identify the channel.
   - For the receiving side of the channel:

     ```
     DISPLAY CHSTATUS(name) SAVED LSTLUWID
     ```

     You can use the CONNAME parameter to further identify the channel.

   The commands are different because only the sending side of the channel can be in doubt. The receiving side is never in doubt.

   On WebSphere MQ for IBM i, the DISPLAY CHSTATUS command can be executed from a file using the STRMQMMQSC command or the Work with MQM Channel Status CL command, WRKMQMCHST
2. If the two LUWIDs are the same, the receiving side has committed the unit of work that the sender considers to be in doubt. The sending side can now remove the in-doubt messages from the transmission queue and re-enable it. This is done with the following channel RESOLVE command:

   ```
   RESOLVE CHANNEL(name) ACTION(COMMIT)
   ```
3. If the two LUWIDs are different, the receiving side has not committed the unit of work that the sender considers to be in doubt. The sending side needs to retain the in-doubt messages on the transmission queue and re-send them. This is done with the following channel RESOLVE command:

   ```
   RESOLVE CHANNEL(name) ACTION(BACKOUT)
   ```

Once this process is complete the channel is no longer in doubt. The transmission queue can now be used by another channel, if required.

Problem determination

There are two distinct aspects to problem determination - problems discovered when a command is being submitted, and problems discovered during operation of the channels.

Command validation

Commands and panel data must be free from errors before they are accepted for processing. Any errors found by the validation are immediately notified to the user by error messages.

Problem diagnosis begins with the interpretation of these error messages and taking corrective action.

Processing problems

Problems found during normal operation of the channels are notified to the system console or the system log. Problem diagnosis begins with the collection of all relevant information from the log, and continues with analysis to identify the problem.

Confirmation and error messages are returned to the terminal that initiated the commands, when possible.

WebSphere MQ produces accounting and statistical data, which you can use to identify trends in utilization and performance. distributed On distributed platforms, this information is produced as PCF records, see Structure data types for details.

Messages and codes

For messages and codes to help with the primary diagnosis of the problem, see Diagnostic messages and reason codes.

Safety of messages

In addition to the typical recovery features of WebSphere MQ, distributed queue management ensures that messages are delivered properly by using a sync point procedure coordinated between the two ends of the message channel. If this procedure detects an error, it closes the channel so that you can investigate the problem, and keeps the messages safely in the transmission queue until the channel is restarted.

The sync point procedure has an added benefit in that it attempts to recover an *in-doubt* situation when the channel starts. (*In-doubt* is the status of a unit of recovery for which a sync point has been requested but the outcome of the request is not yet known.) Also associated with this facility are the two functions:

1. Resolve with commit or backout
2. Reset the sequence number

The use of these functions occurs only in exceptional circumstances because the channel recovers automatically in most cases.

Fast, nonpersistent messages

The nonpersistent message speed (NPMSPEED) channel attribute can be used to specify that any nonpersistent messages on the channel are to be delivered more quickly. For more information about this attribute, see Nonpersistent message speed (NPMSPEED).

If a channel terminates while fast, nonpersistent messages are in transit, the messages might be lost and it is up to the application to arrange for their recovery if required.

If the receiving channel cannot put the message to its destination queue then it is placed on the dead letter queue, if one has been defined. If not, the message is discarded.

Note: If the other end of the channel does not support the option, the channel runs at normal speed.

Undelivered Messages

For information about what happens when a message cannot be delivered, see "What happens when a message cannot be delivered?" on page 65.

What happens when a message cannot be delivered?

When a message cannot be delivered, the MCA can process it in several ways. It can try again, it can return-to-sender, or it can put it on the dead-letter queue.

Figure 15 on page 65 shows the processing that occurs when an MCA is unable to put a message to the destination queue. (The options shown do not apply on all platforms.)

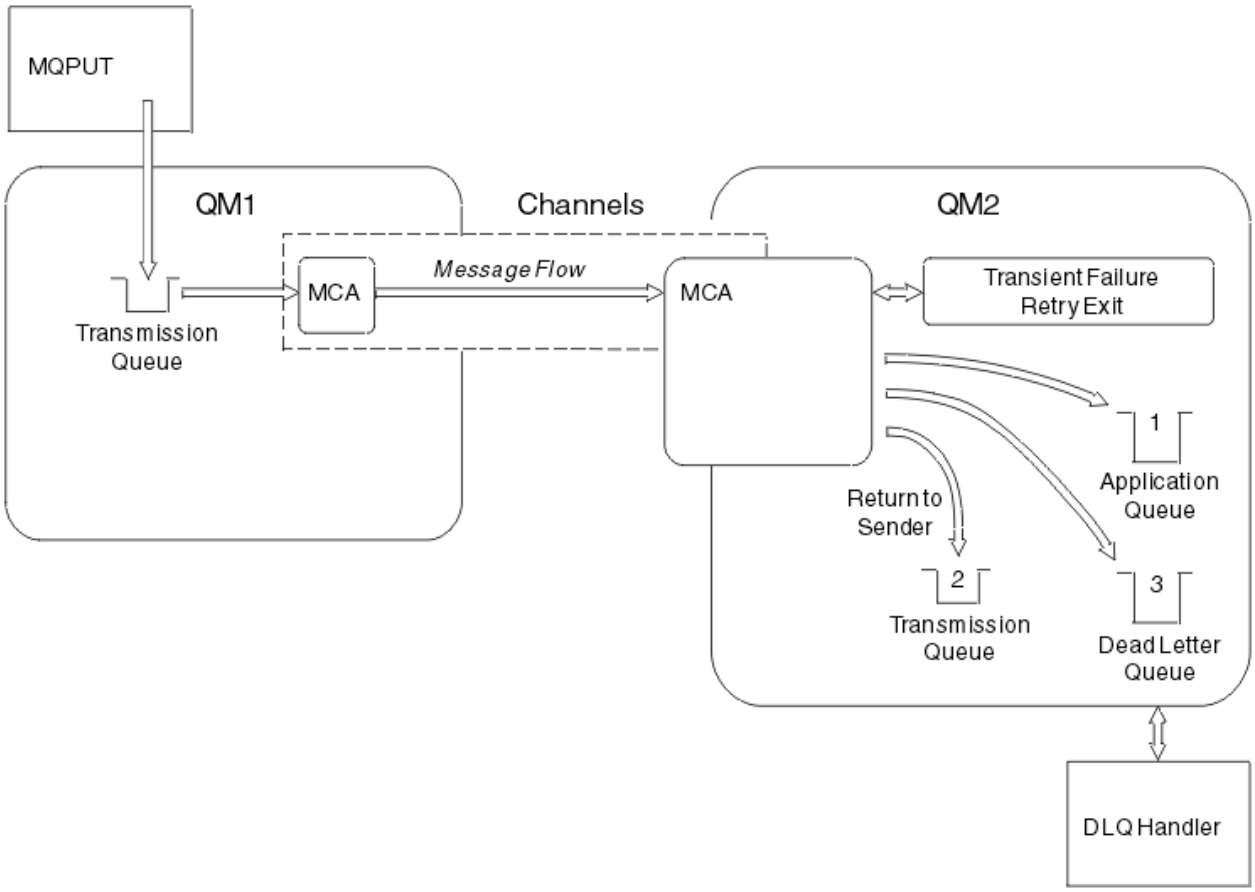


Figure 15. What happens when a message cannot be delivered

As shown in the figure, the MCA can do several things with a message that it cannot deliver. The action taken is determined by options specified when the channel is defined and on the MQPUT report options for the message.

1. Message-retry

   If the MCA is unable to put a message to the target queue for a reason that could be transitory (for example, because the queue is full), the MCA can wait and try the operation again later. You can determine if the MCA waits, for how long, and how many times it tries.

   - You can specify a message-retry time and interval for MQPUT errors when you define your channel. If the message cannot be put to the destination queue because the queue is full, or is inhibited for puts, the MCA tries the operation the number of times specified, at the time interval specified.
   - You can write your own message-retry exit. The exit enables you to specify under what conditions you want the MCA to try the MQPUT or MQOPEN operation again. Specify the name of the exit when you define the channel.

2. Return-to-sender

If message-retry was unsuccessful, or a different type of error was encountered, the MCA can send the message back to the originator. To enable return-to-sender, you need to specify the following options in the message descriptor when you put the message to the original queue:

- The MQRO_EXCEPTION_WITH_FULL_DATA report option
- The MQRO_DISCARD_MSG report option
- The name of the reply-to queue and reply-to queue manager

If the MCA is unable to put the message to the destination queue, it generates an exception report containing the original message, and puts it on a transmission queue to be sent to the reply-to queue specified in the original message. (If the reply-to queue is on the same queue manager as the MCA, the message is put directly to that queue, not to a transmission queue.)

3. Dead-letter queue

   If a message cannot be delivered or returned, it is put on to the dead-letter queue (DLQ). You can use the DLQ handler to process the message. This processing is described in Handling undelivered messages with the WebSphere MQ dead-letter queue handler for IBM WebSphere MQ for UNIX, Linux. If the dead-letter queue is not available, the sending MCA leaves the message on the transmission queue, and the channel stops. On a fast channel, nonpersistent messages that cannot be written to a dead-letter queue are lost.

   On IBM WebSphere MQ Version 7.0, if no local dead-letter queue is defined, the remote queue is not available or defined, and there is no remote dead-letter queue, then the sender channel goes into RETRY and messages are automatically rolled back to the transmission queue.

Related reference

Use Dead-Letter Queue (USEDLQ)

Triggering channels

WebSphere MQ provides a facility for starting an application automatically when certain conditions on a queue are met. This facility is called triggering.

This explanation is intended as an overview of triggering concepts. For a complete description, see Starting WebSphere MQ applications using triggers.

For platform-specific information see the following:

- For Windows, see UNIX and Linux systems, “Triggering channels on UNIX, Linux and Windows systems.” on page 68

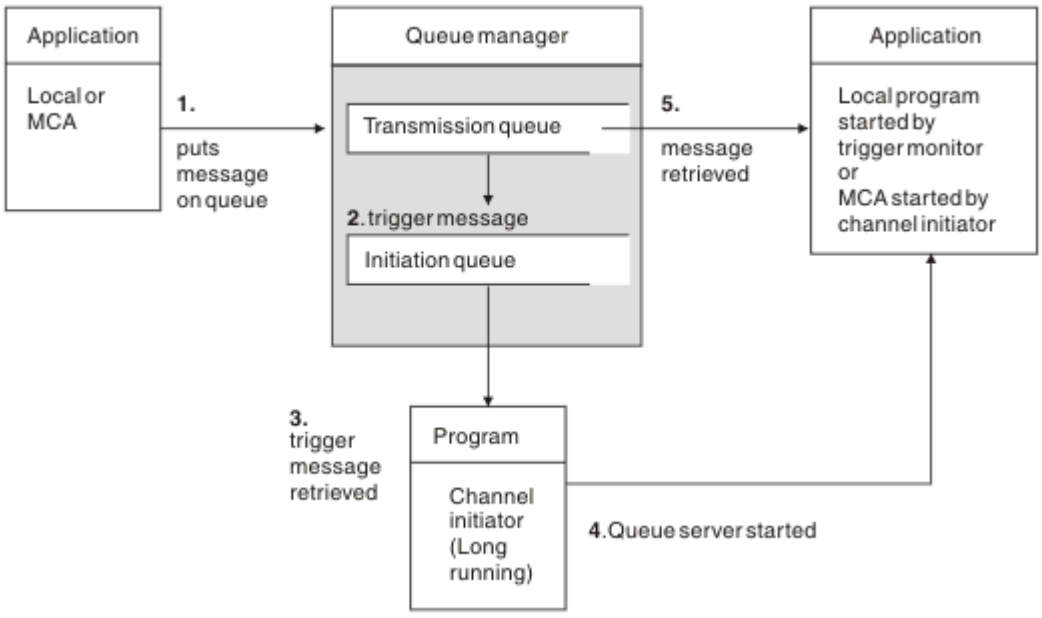


Figure 16. The concepts of triggering

The objects required for triggering are shown in Figure 16 on page 67. It shows the following sequence of events:

1. The local queue manager places a message from an application or from a message channel agent (MCA) on the transmission queue.
2. When the triggering conditions are fulfilled, the local queue manager places a trigger message on the initiation queue.
3. The long-running channel initiator program monitors the initiation queue, and retrieves messages as they arrive.
4. The channel initiator processes the trigger messages according to information contained in them. This information might include the channel name, in which case the corresponding MCA is started.
5. The local application or the MCA, having been triggered, retrieves the messages from the transmission queue.

To set up this scenario, you need to:

- Create the transmission queue with the name of the initiation queue (that is, SYSTEM.CHANNEL.INITQ) in the corresponding attribute.
- Ensure that the initiation queue (SYSTEM.CHANNEL.INITQ) exists.
- Ensure that the channel initiator program is available and running. The channel initiator program must be provided with the name of the initiation queue in its start command.
- Optionally, create the process definition for the triggering, if it does not exist, and ensure that the *UserData* field contains the name of the channel it serves. Instead of creating a process definition, you can specify the channel name in the *TriggerData* attribute of the transmission queue. WebSphere MQ for UNIX, Linux and Windows systems, allow the channel name to be specified as blank, in which case the first available channel definition with this transmission queue is used.
- Ensure that the transmission queue definition contains the name of the process definition to serve it, (if applicable), the initiation queue name, and the triggering characteristics you feel are most suitable. The trigger control attribute allows triggering to be enabled, or not, as necessary.

Note:

1. The channel initiator program acts as a 'trigger monitor' monitoring the initiation queue used to start channels.
2. An initiation queue and trigger process can be used to trigger any number of channels.
3. Any number of initiation queues and trigger processes can be defined.
4. A trigger type of FIRST is recommended, to avoid flooding the system with channel starts.

Triggering channels on UNIX, Linux and Windows systems.

You can create a process definition in WebSphere MQ, defining processes to be triggered. Use the MQSC command DEFINE PROCESS to create a process definition naming the process to be triggered when messages arrive on a transmission queue. The USERDATA attribute of the process definition contains the name of the channel being served by the transmission queue.

Define the local queue (QM4), specifying that trigger messages are to be written to the initiation queue (IQ) to trigger the application that starts channel (QM3.TO.QM4):

```
DEFINE QLOCAL(QM4) TRIGGER INITQ(SYSTEM.CHANNEL.INITQ) PROCESS(P1) USAGE(XMITQ)
```

Define the application (process P1) to be started:

```
DEFINE PROCESS(P1) USERDATA(QM3.TO.QM4)
```

Alternatively, for WebSphere MQ for UNIX, Linux and Windows systems, you can eliminate the need for a process definition by specifying the channel name in the TRIGDATA attribute of the transmission queue.

Define the local queue (QM4). Specify that trigger messages are to be written to the default initiation queue SYSTEM.CHANNEL.INITQ, to trigger the application (process P1) that starts channel (QM3.TO.QM4):

```
DEFINE QLOCAL(QM4) TRIGGER INITQ(SYSTEM.CHANNEL.INITQ)
USAGE(XMITQ) TRIGDATA(QM3.TO.QM4)
```

If you do not specify a channel name, the channel initiator searches the channel definition files until it finds a channel that is associated with the named transmission queue.

Related concepts

“Starting and stopping the channel initiator” on page 68
Triggering is implemented using the channel initiator process.

“Connecting applications using distributed queuing” on page 27
This section provides more detailed information about intercommunication between WebSphere MQ installations, including queue definition, channel definition, triggering, and sync point procedures

Related reference

Channel programs on UNIX, Linux, and Windows systems

Starting and stopping the channel initiator

Triggering is implemented using the channel initiator process.

This channel initiator process is started with the MQSC command START CHINIT. Unless you are using the default initiation queue, specify the name of the initiation queue on the command. For example, to use the START CHINIT command to start queue IQ for the default queue manager, enter:

```
START CHINIT INITQ(IQ)
```

By default, a channel initiator is started automatically using the default initiation queue, SYSTEM.CHANNEL.INITQ. If you want to start all your channel initiators manually, follow these steps:

1. Create and start the queue manager.
2. Alter the queue manager's SCHINIT property to MANUAL
3. End and restart the queue manager

In Linux and Windows systems, a channel initiator is started automatically. The number of channel initiators that you can start is limited. The default and maximum value is 3. You can change this using MAXINITIATORS in the qm.ini file for UNIX and Linux systems, and in the registry for Windows systems.

See WebSphere MQ Control commands for details of the run channel initiator command **runmqchi**, and the other control commands.

Stopping the channel initiator

The default channel initiator is started automatically when you start a queue manager. All channel initiators are stopped automatically when a queue manager is stopped.

Initialization and configuration files

The handling of channel initialization data depends on your WebSphere MQ platform.

Windows, UNIX and Linux systems

In WebSphere MQ for Windows, UNIX and Linux systems, there are *configuration files* to hold basic configuration information about the WebSphere MQ installation.

There are two configuration files: one applies to the machine, the other applies to an individual queue manager.

WebSphere MQ configuration file
This file holds information relevant to all the queue managers on the WebSphere MQ system. The file is called mqs.ini. It is fully described in the Administering for WebSphere MQ for Windows, UNIX and Linux systems.

Queue manager configuration file

This file holds configuration information relating to one particular queue manager. The file is called qm.ini.

It is created during queue manager creation and can hold configuration information relevant to any aspect of the queue manager. Information held in the file includes details of how the configuration of the log differs from the default in WebSphere MQ configuration file.

The queue manager configuration file is held in the root of the directory tree occupied by the queue manager. For example, for the DefaultPath attributes, the queue manager configuration files for a queue manager called QMNAME would be:

For UNIX and Linux systems:

```
/var/mqm/qmgrs/QMNAME/qm.ini
```

An excerpt of a qm.ini file follows. It specifies that the TCP/IP listener is to listen on port 2500, the maximum number of current channels is to be 200, and the maximum number of active channels is to be 100.

```
     TCP:
       Port=2500
     CHANNELS:
       MaxChannels=200
       MaxActiveChannels=100
```

You can specify a range of TCP/IP ports to be used by an outbound channel. One method is to use the qm.ini file to specify the start and end of a range of port values. The following example shows a qm.ini file specifying a range of channels:

```
TCP:
  StrPort=2500
  EndPort=3000
CHANNELS:
  MaxChannels=200
  MaxActiveChannels=100
```

If you specify a value for StrPort or EndPort then you must specify a value for both. The value of EndPort must always be greater than the value of StrPort.

The channel tries to use each of the port values in the range specified. When the connection is successful, the port value is the port that the channel then uses.

For Windows systems:

```
C:\Program Files\IBM\WebSphere MQ\qmgrs\QMNAME\qm.ini
```

For more information about qm.ini files, see Configuration file stanzas for distributed queuing.

Data conversion for messages

WebSphere MQ messages might require data conversion when sent between queues on different queue managers.

A WebSphere MQ message consists of two parts:

- Control information in a message descriptor
- Application data

Either of the two parts might require data conversion when sent between queues on different queue managers. For information about application data conversion, see Application data conversion.

Writing your own message channel agents

WebSphere MQ allows you to write your own message channel agent (MCA) programs or to install one from an independent software vendor.

You might want to write your own MCA programs to make WebSphere MQ interoperate over your own proprietary communications protocol, or to send messages over a protocol that WebSphere MQ does not support. (You cannot write your own MCA to interoperate with a WebSphere MQ-supplied MCA at the other end.)

If you decide to use an MCA that was not supplied by WebSphere MQ, you must consider the following points.

Message sending and receiving
You must write a sending application that gets messages from wherever your application puts them, for example from a transmission queue, and sends them out on a protocol with which you want to communicate. You must also write a receiving application that takes messages from this protocol and puts them onto destination queues. The sending and receiving applications use the message queue interface (MQI) calls, not any special interfaces.

You must ensure that messages are only delivered once. Sync point coordination can be used to help with this delivery.

Channel control function
You must provide your own administration functions to control channels. You cannot use WebSphere MQ channel administration functions either for configuring (for example, the DEFINE CHANNEL command) or monitoring (for example, DISPLAY CHSTATUS) your channels.

Initialization file
You must provide your own initialization file, if you require one.

Application data conversion
You probably want to allow for data conversion for messages you send to a different system. If so, use the MQGMO_CONVERT option on the MQGET call when retrieving messages from wherever your application puts them, for example the transmission queue.

User exits
Consider whether you need user exits. If so, you can use the same interface definitions that WebSphere MQ uses.

Triggering
If your application puts messages to a transmission queue, you can set up the transmission queue attributes so that your sending MCA is triggered when messages arrive on the queue.

Channel initiator
You might must provide your own channel initiator.

Other things to consider for distributed queue management

Other topics to consider when preparing WebSphere MQ for distributed queue management. This topic covers Undelivered-message queue, Queues in use, System extensions and user-exit programs, and Running channels and listeners as trusted applications.

Undelivered-message queue

To ensure that messages arriving on the undelivered-message queue (also known as the dead-letter queue or DLQ) are processed, create a program that can be triggered or run at regular intervals to handle these messages. A DLQ handler is provided with WebSphere MQ on UNIX and Linux systems; for more information, see The sample DLQ handler, amqsdlq.

Queues in use

MCAs for receiver channels can keep the destination queues open even when messages are not being transmitted. This results in the queues appearing to be "in use".

Maximum number of channels

See Configuration file stanzas for distributed queuing .

System extensions and user-exit programs

A facility is provided in the channel definition to enable extra programs to be run at defined times during the processing of messages. These programs are not supplied with WebSphere MQ, but can be provided by each installation according to local requirements.

In order to run, these user-exit programs must have predefined names and be available on call to the channel programs. The names of the user-exit programs are included in the message channel definitions.

There is a defined control block interface for handing over control to these programs, and for handling the return of control from these programs.

The precise places where these programs are called, and details of control blocks and names, are to be found in Channel-exit programs for messaging channels .

Running channels and listeners as trusted applications

If performance is an important consideration in your environment and your environment is stable, you can run your channels and listeners as trusted, using the FASTPATH binding. There are two factors that influence whether channels and listeners run as trusted:

- The environment variable MQ_CONNECT_TYPE=FASTPATH or MQ_CONNECT_TYPE=STANDARD. This is case-sensitive. If you specify a value that is not valid it is ignored.
- MQIBindType in the Channels stanza of the `qm.ini` or registry file. You can set this to FASTPATH or STANDARD and it is not case-sensitive. The default is STANDARD.

You can use MQIBindType in association with the environment variable to achieve the required effect as follows:

MQIBindType	Environment variable	Result
STANDARD	UNDEFINED	STANDARD
FASTPATH	UNDEFINED	FASTPATH
STANDARD	STANDARD	STANDARD
FASTPATH	STANDARD	STANDARD
STANDARD	FASTPATH	STANDARD
FASTPATH	FASTPATH	FASTPATH
STANDARD	CLIENT	CLIENT
FASTPATH	CLIENT	STANDARD
STANDARD	LOCAL	STANDARD
FASTPATH	LOCAL	STANDARD

In summary, there are only two ways of actually making channels and listeners run as trusted:

1. By specifying MQIBindType=FASTPATH in `qm.ini` or registry and not specifying the environment variable.
2. By specifying MQIBindType=FASTPATH in `qm.ini` or registry and setting the environment variable to FASTPATH.

Consider running listeners as trusted, because listeners are stable processes. Consider running channels as trusted, unless you are using unstable channel exits or the command STOP CHANNEL MODE(TERMINATE).

Monitoring and controlling channels on UNIX, Linux, and Windows

For DQM you need to create, monitor, and control the channels to remote queue managers. You can control channels using commands, programs, IBM WebSphere MQ Explorer, files for the channel definitions, and a storage area for synchronization information.

You can use the following types of command:

The IBM WebSphere MQ commands (MQSC)

You can use the MQSC as single commands in an MQSC session in Windows, UNIX and Linux systems. To issue more complicated, or multiple, commands the MQSC can be built into a file that you then run from the command line. For details, see MQSC commands. This section gives some simple examples of using MQSC for distributed queuing.

The channel commands are a subset of the IBM WebSphere MQ Commands (MQSC). You use MQSC and the control commands to:

- Create, copy, display, change, and delete channel definitions
- Start and stop channels, ping, reset channel sequence numbers, and resolve in-doubt messages when links cannot be re-established
- Display status information about channels

Control commands
You can also issue *control commands* at the command line for some of these functions. For details, see control commands.

Programmable command format commands
For details, see PCF commands.

IBM WebSphere MQ Explorer
On UNIX, Linux and Windows systems, you can use the IBM WebSphere MQ Explorer. This provides a graphical administration interface to perform administrative tasks as an alternative to using control commands or MQSC commands. Channel definitions are held as queue manager objects.

Each queue manager has a DQM component for controlling interconnections to compatible remote queue managers. A storage area holds sequence numbers and *logical unit of work (LUW)* identifiers. These are used for channel synchronization purposes.

For a list of the functions available to you when setting up and controlling message channels, using the different types of command, see Table 8 on page 74.

Related concepts

"Getting started with objects" on page 75
Channels must be defined, and their associated objects must exist and be available for use, before a channel can be started. This section shows you how.

"Setting up communication for Windows" on page 81
When a distributed-queuing management channel is started, it tries to use the connection specified in the channel definition. For this to succeed, it is necessary for the connection to be defined and available. This section explains how to do this using one of the four forms of communication for WebSphere MQ for Windows systems.

"Setting up communication on UNIX and Linux systems" on page 90
DQM is a remote queuing facility for IBM WebSphere MQ. It provides channel control programs for the queue manager which form the interface to communication links, controllable by the system operator. The channel definitions held by distributed-queuing management use these connections.

Related reference

Channel programs on UNIX, Linux, and Windows systems

Message channel planning example for distributed platforms

Example configuration information

Channel attributes

Functions required for setting up and controlling channels

A number of IBM WebSphere MQ functions might be needed to set up and control channels. The channel functions are explained in this topic.

You can create a channel definition using the default values supplied by IBM WebSphere MQ, specifying the name of the channel, the type of channel you are creating, the communication method to be used, the transmission queue name and the connection name.

The channel name must be the same at both ends of the channel, and unique within the network. However, you must restrict the characters used to those that are valid for IBM WebSphere MQ object names.

For other channel related functions, see the following topics:

- "Getting started with objects" on page 75
- "Creating associated objects" on page 76
- "Creating default objects" on page 76
- "Creating a channel" on page 76
- "Displaying a channel" on page 77
- "Displaying channel status" on page 77

- “Checking links using Ping” on page 78
- “Starting a channel” on page 78
- “Stopping a channel” on page 79
- “Renaming a channel” on page 80
- “Resetting a channel” on page 80
- “Resolving in-doubt messages on a channel” on page 81

Table 8 on page 74 shows the full list of IBM WebSphere MQ functions that you might need.

Table 8. Functions required in UNIX, Linux, and Windows systems

Function	**Control commands**	**MQSC**	**WebSphere MQ Explorer equivalent?**
Queue manager functions			
Change queue manager		ALTER QMGR	Yes
Create queue manager	crtmqm		Yes
Delete queue manager	dltmqm		Yes
Display queue manager		DISPLAY QMGR	Yes
End queue manager	endmqm		Yes
Ping queue manager		PING QMGR	No
Start queue manager	strmqm		Yes
Command server functions			
Display command server	dspmqcsv		No
End command server	endmqcsv		No
Start command server	strmqcsv		No
Queue functions			
Change queue		ALTER QALIAS ALTER QLOCAL ALTER QMODEL ALTER QREMOTE See, ALTER queues .	Yes
Clear queue		CLEAR QLOCAL	Yes
Create queue		DEFINE QALIAS DEFINE QLOCAL DEFINE QMODEL DEFINE QREMOTE See, DEFINE queues .	Yes
Delete queue		DELETE QALIAS DELETE QLOCAL DELETE QMODEL DELETE QREMOTE See, DELETE queues .	Yes

Table 8. Functions required in UNIX, Linux, and Windows systems (continued)

Function	Control commands	MQSC	WebSphere MQ Explorer equivalent?
Display queue		DISPLAY QUEUE	Yes
Process functions			
Change process		ALTER PROCESS	Yes
Create process		DEFINE PROCESS	Yes
Delete process		DELETE PROCESS	Yes
Display process		DISPLAY PROCESS	Yes
Channel functions			
Change channel		ALTER CHANNEL	Yes
Create channel		DEFINE CHANNEL	Yes
Delete channel		DELETE CHANNEL	Yes
Display channel		DISPLAY CHANNEL	Yes
Display channel status		DISPLAY CHSTATUS	Yes
End channel		STOP CHANNEL	Yes
Ping channel		PING CHANNEL	Yes
Reset channel		RESET CHANNEL	Yes
Resolve channel		RESOLVE CHANNEL	Yes
Run channel	runmqchl	START CHANNEL	Yes
Run channel initiator	runmqchi	START CHINIT	No
Run listener[1]	runmqlsr	START LISTENER	No
End listener	endmqlsr (Windows systems, AIX, HP-UX, and Solaris only)		No

Note:

1. A listener might be started automatically when the queue manager starts.

Getting started with objects

Channels must be defined, and their associated objects must exist and be available for use, before a channel can be started. This section shows you how.

Use the WebSphere MQ commands (MQSC) or the IBM WebSphere MQ Explorer to:

1. Define message channels and associated objects
2. Monitor and control message channels

The associated objects you might need to define are:

- Transmission queues
- Remote queue definitions
- Queue manager alias definitions

- Reply-to queue alias definitions
- Reply-to local queues
- Processes for triggering (MCAs)
- Message channel definitions

The particular communication link for each channel must be defined and available before a channel can be run. For a description of how LU 6.2, TCP/IP, NetBIOS, SPX, and DECnet links are defined, see the particular communication guide for your installation. See also Example configuration information.

For more information about creating and working with objects, see the following subtopics:

Creating associated objects

MQSC is used to create associated objects.

Use MQSC to create the queue and alias objects: transmission queues, remote queue definitions, queue manager alias definitions, reply-to queue alias definitions, and reply-to local queues.

Also create the definitions of processes for triggering (MCAs) in a similar way.

For an example showing how to create all the required objects see Message channel planning example for distributed platforms .

Creating default objects

Default objects are created automatically when a queue manager is created. These objects are queues, channels, a process definition, and administration queues. After the default objects have been created, you can replace them at any time by running the strmqm command with the -c option.

When you use the crtmqm command to create a queue manager, the command also initiates a program to create a set of default objects.

1. Each default object is created in turn. The program keeps a count of how many objects are successfully defined, how many existed and were replaced, and how many unsuccessful attempts there were.
2. The program displays the results to you and if any errors occurred, directs you to the appropriate error log for details.

When the program has finished running, you can use the strmqm command to start the queue manager.

See The control commands for more information about the crtmqm and strmqm commands.

Changing the default objects

When you specify the -c option, the queue manager is started temporarily while the objects are created and is then shut down again. Issuing strmqm with the -c option refreshes existing system objects with the default values (for example, the MCAUSER attribute of a channel definition is set to blanks). You must use the strmqm command again, without the -c option, if you want to start the queue manager.

If you want to change the default objects, you can create your own version of the old amqscoma.tst file and edit it.

Creating a channel

Create ***two*** channel definitions, one at each end of the connection. You create the first channel definition at the first queue manager. Then you create the second channel definition at the second queue manager, on the other end of the link.

Both ends must be defined using the ***same*** channel name. The two ends must have **compatible** channel types, for example: Sender and Receiver.

To create a channel definition for one end of the link use the MQSC command DEFINE CHANNEL. Include the name of the channel, the channel type for this end of the connection, a connection name, a description (if required), the name of the transmission queue (if required), and the transmission protocol. Also include any other attributes that you want to be different from the system default values for the required channel type, using the information you have gathered previously.

You are provided with help in deciding on the values of the channel attributes in Channel attributes.

Note: You are recommended to name all the channels in your network uniquely. Including the source and target queue manager names in the channel name is a good way to do this.

Create channel example

```
DEFINE CHANNEL(QM1.TO.QM2) CHLTYPE(SDR) +
DESCR('Sender channel to QM2') +
CONNAME(QM2) TRPTYPE(TCP) XMITQ(QM2) CONVERT(YES)
```

In all the examples of MQSC the command is shown as it appears in a file of commands, and as it is typed in Windows or UNIX or Linux systems. The two methods look identical, except that to issue a command interactively, you must first start an MQSC session. Type `runmqsc`, for the default queue manager, or `runmqsc` *qmname* where *qmname* is the name of the required queue manager. Then type any number of commands, as shown in the examples.

For portability, restrict the line length of your commands to 72 characters. Use the concatenation character, +, as shown to continue over more than one line. On Windows use Ctrl-z to end the entry at the command line. On UNIX and Linux systems, use Ctrl-d. Alternatively, on UNIX, Linux or Windows systems, use the **end** command.

Displaying a channel

Use the MQSC command DISPLAY CHANNEL to display the attributes of a channel.

The ALL parameter of the DISPLAY CHANNEL command is assumed by default if no specific attributes are requested and the channel name specified is not generic.

The attributes are described in Channel attributes.

Display channel examples

```
DISPLAY CHANNEL(QM1.TO.QM2) TRPTYPE,CONVERT

DISPLAY CHANNEL(QM1.TO.*) TRPTYPE,CONVERT

DISPLAY CHANNEL(*) TRPTYPE,CONVERT

DISPLAY CHANNEL(QM1.TO.QMR34) ALL
```

Displaying channel status

Use the MQSC command DISPLAY CHSTATUS, specifying the channel name and whether you want the current status of channels or the status of saved information.

DISPLAY CHSTATUS applies to all message channels. It does not apply to MQI channels other than server-connection channels.

Information displayed includes:

- Channel name
- Communication connection name
- In-doubt status of channel (where appropriate)
- Last sequence number
- Transmission queue name (where appropriate)
- The in-doubt identifier (where appropriate)
- The last committed sequence number
- Logical unit of work identifier
- Process ID
- Thread ID (Windows only)

Display channel status examples

```
DISPLAY CHSTATUS(*) CURRENT
DISPLAY CHSTATUS(QM1.TO.*) SAVED
```

The saved status does not apply until at least one batch of messages has been transmitted on the channel. Status is also saved when a channel is stopped (using the STOP CHL command) and when the queue manager is ended.

Checking links using Ping

Use the MQSC command PING CHANNEL to exchange a fixed data message with the remote end.

Ping gives some confidence to the system supervisor that the link is available and functioning.

Ping does not involve the use of transmission queues and target queues. It uses channel definitions, the related communication link, and the network setup. It can only be used if the channel is not currently active.

It is available from sender and server channels only. The corresponding channel is started at the far side of the link, and performs the startup parameter negotiation. Errors are notified normally.

The result of the message exchange is presented as `Ping complete` or an error message.

Ping with LU 6.2

When Ping is invoked, by default no user ID or password flows to the receiving end. If user ID and password are required, they can be created at the initiating end in the channel definition. If a password is entered into the channel definition, it is encrypted by WebSphere MQ before being saved. It is then decrypted before flowing across the conversation.

Starting a channel

Use the MQSC command START CHANNEL for sender, server, and requester channels. For applications to be able to exchange messages, you must start a listener program for inbound connections.

START CHANNEL is not necessary where a channel has been set up with queue manager triggering.

When started, the sending MCA reads the channel definitions and opens the transmission queue. A channel start-up sequence is issued, which remotely starts the corresponding MCA of the receiver or server channel. When they have been started, the sender and server processes await messages arriving on the transmission queue and transmit them as they arrive.

When you use triggering or run channels as threads, ensure that the channel initiator is available to monitor the initiation queue. The channel initiator is started by default as part of the queue manager.

However, TCP and LU 6.2 do provide other capabilities:

- For TCP on UNIX and Linux systems, inetd can be configured to start a channel. inetd is started as a separate process.
- For LU 6.2 in UNIX and Linux systems, configure your SNA product to start the LU 6.2 responder process.
- For LU 6.2 in Windows systems, using SNA Server you can use TpStart (a utility provided with SNA Server) to start a channel. TpStart is started as a separate process.

Use of the Start option always causes the channel to resynchronize, where necessary.

For the start to succeed:

- Channel definitions, local and remote, must exist. If there is no appropriate channel definition for a receiver or server-connection channel, a default one is created automatically if the channel is auto-defined. See Channel auto-definition exit program.

- Transmission queue must exist, and have no other channels using it.
- MCAs, local and remote, must exist.
- Communication link must be available.
- Queue managers must be running, local and remote.
- Message channel must not be already running.

A message is returned to the screen confirming that the request to start a channel has been accepted. For confirmation that the start command has succeeded, check the error log, or use DISPLAY CHSTATUS. The error logs are:

Windows

MQ_INSTALLATION_PATH\qmgrs\qmname\errors\AMQERR01.LOG (for each queue manager called qmname)

MQ_INSTALLATION_PATH\qmgrs\@SYSTEM\errors\AMQERR01.LOG (for general errors)

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

Note: On Windows systems, you still also get a message in the Windows systems application event log.

UNIX and Linux systems

/var/mqm/qmgrs/qmname/errors/AMQERR01.LOG (for each queue manager called qmname)

/var/mqm/qmgrs/@SYSTEM/errors/AMQERR01.LOG (for general errors)

On Windows, UNIX and Linux systems, use the runmqlsr command to start the WebSphere MQ listener process. By default, any inbound requests for channel attachment causes the listener process to start MCAs as threads of the amqrmppa process.

```
runmqlsr -t tcp -m QM2
```

For outbound connections, you must start the channel in one of the following three ways:

1. Use the MQSC command START CHANNEL, specifying the channel name, to start the channel as a process or a thread, depending on the MCATYPE parameter. (If channels are started as threads, they are threads of a channel initiator.)

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

2. Use the control command runmqchl to start the channel as a process.

   ```
   runmqchl -c QM1.TO.QM2 -m QM1
   ```

3. Use the channel initiator to trigger the channel.

Stopping a channel

Use the MQSC command STOP CHANNEL to request the channel to stop activity. The channel does not start a new batch of messages until the operator starts the channel again.

For information about restarting stopped channels, see “Restarting stopped channels” on page 62.

This command can be issued to a channel of any type except MQCHT_CLNTCONN.

You can select the type of stop you require:

Stop quiesce example

```
STOP CHANNEL(QM1.TO.QM2) MODE(QUIESCE)
```

This command requests the channel to close down in an orderly way. The current batch of messages is completed and the sync point procedure is carried out with the other end of the channel. If the channel is idle this command does not terminate a receiving channel.

Stop force example

```
STOP CHANNEL(QM1.TO.QM2) MODE(FORCE)
```

This option stops the channel immediately, but does not terminate the channel's thread or process. The channel does not complete processing the current batch of messages, and can, therefore, leave the channel in doubt. In general, consider using the quiesce stop option.

Stop terminate example

```
STOP CHANNEL(QM1.TO.QM2) MODE(TERMINATE)
```

This option stops the channel immediately, and terminates the channel's thread or process.

Stop (quiesce) stopped example

```
STOP CHANNEL(QM1.TO.QM2) STATUS(STOPPED)
```

This command does not specify a MODE, so defaults to MODE(QUIESCE). It requests that the channel is stopped so that it cannot be restarted automatically but must be started manually.

Stop (quiesce) inactive example

```
STOP CHANNEL(QM1.TO.QM2) STATUS(INACTIVE)
```

This command does not specify a MODE, so defaults to MODE(QUIESCE). It requests that the channel is made inactive so that it restarts automatically when required.

Renaming a channel

Use MQSC to rename a message channel.

Use MQSC to carry out the following steps:

1. Use STOP CHANNEL to stop the channel.
2. Use DEFINE CHANNEL to create a duplicate channel definition with the new name.
3. Use DISPLAY CHANNEL to check that it has been created correctly.
4. Use DELETE CHANNEL to delete the original channel definition.

If you decide to rename a message channel, remember that a channel has ***two*** channel definitions, one at each end. Make sure that you rename the channel at both ends at the same time.

Resetting a channel

Use the MQSC command RESET CHANNEL to change the message sequence number.

The RESET CHANNEL command is available for any message channel, but not for MQI channels (client-connection or server-connection). The first message starts the new sequence the next time the channel is started.

If the command is issued on a sender or server channel, it informs the other side of the change when the channel is restarted.

Related concepts

"Getting started with objects" on page 75
Channels must be defined, and their associated objects must exist and be available for use, before a channel can be started. This section shows you how.

"Channel control function" on page 52
The channel control function provides facilities for you to define, monitor, and control channels.

"Connecting applications using distributed queuing" on page 27
This section provides more detailed information about intercommunication between WebSphere MQ installations, including queue definition, channel definition, triggering, and sync point procedures

Related reference

RESET CHANNEL

Resolving in-doubt messages on a channel

Use the MQSC command RESOLVE CHANNEL when messages are held in-doubt by a sender or server. For example because one end of the link has terminated, and there is no prospect of it recovering.

The RESOLVE CHANNEL command accepts one of two parameters: BACKOUT or COMMIT. Backout restores messages to the transmission queue, while Commit discards them.

The channel program does not try to establish a session with a partner. Instead, it determines the logical unit of work identifier (LUWID) which represents the in-doubt messages. It then issues, as requested, either:

- BACKOUT to restore the messages to the transmission queue; or
- COMMIT to delete the messages from the transmission queue.

For the resolution to succeed:

- The channel must be inactive
- The channel must be in doubt
- The channel type must be sender or server
- A local channel definition must exist
- The local queue manager must be running

Related concepts

"Getting started with objects" on page 75
Channels must be defined, and their associated objects must exist and be available for use, before a channel can be started. This section shows you how.

"Channel control function" on page 52
The channel control function provides facilities for you to define, monitor, and control channels.

"Connecting applications using distributed queuing" on page 27
This section provides more detailed information about intercommunication between WebSphere MQ installations, including queue definition, channel definition, triggering, and sync point procedures

Related reference

RESOLVE CHANNEL

Setting up communication for Windows

When a distributed-queuing management channel is started, it tries to use the connection specified in the channel definition. For this to succeed, it is necessary for the connection to be defined and available. This section explains how to do this using one of the four forms of communication for WebSphere MQ for Windows systems.

You might find it helpful to refer to Example configuration - IBM WebSphere MQ for Windows.

For UNIX and Linux systems see “Setting up communication on UNIX and Linux systems” on page 90.

Deciding on a connection

Choose from the following four forms of communication for WebSphere MQ for Windows systems:

- “Defining a TCP connection on Windows ” on page 82
- “Defining an LU 6.2 connection on Windows” on page 84
- “Defining a NetBIOS connection on Windows” on page 85
- “Defining an SPX connection on Windows” on page 88 (Windows XP and Windows 2003 Server only)

Each channel definition must specify only one protocol as the Transmission protocol (Transport Type) attribute. One or more protocols can be used by a queue manager.

For WebSphere MQ clients, it might be useful to have alternative channels using different transmission protocols. For more information about WebSphere MQ clients, see Overview of clients.

Related concepts

“Connecting applications using distributed queuing” on page 27
This section provides more detailed information about intercommunication between WebSphere MQ installations, including queue definition, channel definition, triggering, and sync point procedures

“Monitoring and controlling channels on UNIX, Linux, and Windows” on page 72
For DQM you need to create, monitor, and control the channels to remote queue managers. You can control channels using commands, programs, IBM WebSphere MQ Explorer, files for the channel definitions, and a storage area for synchronization information.

“Configuring connections between the client and server” on page 96
To configure the communication links between WebSphere MQ MQI clients and servers, decide on your communication protocol, define the connections at both ends of the link, start a listener, and define channels.

Defining a TCP connection on Windows

Define a TCP connection by configuring a channel at the sending end to specify the address of the target, and by running a listener program at the receiving end.

Sending end

Specify the host name, or the TCP address of the target machine, in the Connection name field of the channel definition.

The port to connect to defaults to 1414. Port number 1414 is assigned by the Internet Assigned Numbers Authority to IBM WebSphere MQ.

To use a port number other than the default, specify it in the connection name field of the channel object definition thus:

```
DEFINE CHANNEL('channel name') CHLTYPE(SDR) +
          TRPTYPE(TCP) +
          CONNAME('OS2ROG3(1822)') +
          XMITQ('XMitQ name') +
          REPLACE
```

where 0S2R0G3 is the DNS name of the remote queue manager and 1822 is the port required. (This must be the port that the listener at the receiving end is listening on.)

A running channel must be stopped and restarted to pick up any change to the channel object definition.

You can change the default port number by specifying it in the `.ini` file for IBM WebSphere MQ for Windows:

```
TCP:
```

```
Port=1822
```

Note: To select which TCP/IP port number to use, IBM WebSphere MQ uses the first port number it finds in the following sequence:

1. The port number explicitly specified in the channel definition or command line. This number allows the default port number to be overridden for a channel.
2. The port attribute specified in the TCP stanza of the `.ini` file. This number allows the default port number to be overridden for a queue manager.
3. The default value of 1414. This is the number assigned to IBM WebSphere MQ by the Internet Assigned Numbers Authority for both inbound and outbound connections.

For more information about the values you set using qm.ini, see Configuration file stanzas for distributed queuing.

Receiving on TCP

To start a receiving channel program, a listener program must be started to detect incoming network requests and start the associated channel. You can use the IBM WebSphere MQ listener.

Receiving channel programs are started in response to a startup request from the sending channel.

To start a receiving channel program, a listener program must be started to detect incoming network requests and start the associated channel. You can use the IBM WebSphere MQ listener.

To run the Listener supplied with IBM WebSphere MQ, that starts new channels as threads, use the runmqlsr command.

A basic example of using the **runmqlsr** command:

```
runmqlsr -t tcp [-m QMNAME] [-p 1822]
```

The square brackets indicate optional parameters; QMNAME is not required for the default queue manager, and the port number is not required if you are using the default (1414). The port number must not exceed 65535.

Note: To select which TCP/IP port number to use, IBM WebSphere MQ uses the first port number it finds in the following sequence:

1. The port number explicitly specified in the channel definition or command line. This number allows the default port number to be overridden for a channel.
2. The port attribute specified in the TCP stanza of the `.ini` file. This number allows the default port number to be overridden for a queue manager.
3. The default value of 1414. This is the number assigned to IBM WebSphere MQ by the Internet Assigned Numbers Authority for both inbound and outbound connections.

For the best performance, run the IBM WebSphere MQ listener as a trusted application as described in “Running channels and listeners as trusted applications” on page 71. See Restrictions for trusted applications for information about trusted applications

Using the TCP/IP SO_KEEPALIVE option

If you want to use the Windows SO_KEEPALIVE option you must add the following entry to your registry:

```
TCP:
KeepAlive=yes
```

For more information about the SO_KEEPALIVE option, see “Checking that the other end of the channel is still available” on page 60.

On Windows, the HKLM\SYSTEM\CurrentControlSet\Services\Tcpip\Parameters registry value for the Windows KeepAliveTime option controls the interval that elapses before the connection is checked. The default is two hours.

Defining an LU 6.2 connection on Windows

SNA must be configured so that an LU 6.2 conversation can be established between the two machines.

Once the SNA is configured, proceed as follows.

See the following table for information.

Table 9. Settings on the local Windows system for a remote queue manager platform

Remote platform	TPNAME	TPPATH
z/OS or MVS/ESA without CICS	The same as in the corresponding side information about the remote queue manager.	-
z/OS or MVS/ESA using CICS	CKRC (sender) CKSV (requester) CKRC (server)	-
IBM i	The same as the compare value in the routing entry on the IBM i system.	-
UNIX and Linux systems	The same as in the corresponding side information about the remote queue manager.	*MQ_INSTALLATION_PATH*/bin/amqcrs6a
Windows	As specified in the Windows Run Listener command, or the invokable Transaction Program that was defined using TpSetup on Windows.	*MQ_INSTALLATION_PATH*\bin\amqcrs6a

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

If you have more than one queue manager on the same machine, ensure that the TPnames in the channel definitions are unique.

For the latest information about configuring AnyNet® SNA over TCP/IP, see the following online IBM documentation: AnyNet SNA over TCP/IP and SNA Node Operations.

Related concepts

"Sending end on LU 6.2" on page 84
Create a CPI-C side object (symbolic destination) from the administration application of the LU 6.2 product you are using. Enter this name in the Connection name field in the channel definition. Also create an LU 6.2 link to the partner.

"Receiving on LU 6.2" on page 85
Receiving channel programs are started in response to a startup request from the sending channel.

Sending end on LU 6.2

Create a CPI-C side object (symbolic destination) from the administration application of the LU 6.2 product you are using. Enter this name in the Connection name field in the channel definition. Also create an LU 6.2 link to the partner.

In the CPI-C side object enter the partner LU Name at the receiving machine, the TP Name and the Mode Name. For example:

```
Partner LU Name              OS2ROG2
Partner TP Name              recv
Mode Name                    #INTER
```

Receiving on LU 6.2
Receiving channel programs are started in response to a startup request from the sending channel.

To start a receiving channel program, a listener program has to be started to detect incoming network requests and start the associated channel. You start this listener program with the RUNMQLSR command, giving the TpName to listen on. Alternatively, you can use TpStart under SNA Server for Windows.

Using the RUNMQLSR command

Example of the command to start the listener:

```
RUNMQLSR -t LU62 -n RECV [-m QMNAME]
```

where RECV is the TpName that is specified at the other (sending) end as the "TpName to start on the remote side". The last part in square brackets is optional and is not required for the default queue manager.

It is possible to have more than one queue manager running on one machine. You must assign a different TpName to each queue manager, and then start a listener program for each one. For example:

```
RUNMQLSR -t LU62 -m QM1 -n TpName1
RUNMQLSR -t LU62 -m QM2 -n TpName2
```

For the best performance, run the WebSphere MQ listener as a trusted application as described in Running channels and listeners as trusted applications. See Restrictions for trusted applications for information about trusted applications.

You can stop all WebSphere MQ listeners running on a queue manager that is inactive, using the command:

```
ENDMQLSR [-m QMNAME]
```

Using Microsoft SNA Server on Windows

You can use TpSetup (from the SNA Server SDK) to define an invokable TP that then drives amqcrs6a.exe, or you can set various registry values manually. The parameters that should be passed to amqcrs6a.exe are:

```
   -m QM -n TpName
```

where *QM* is the Queue Manager name and *TpName* is the TP Name. See the *Microsoft SNA Server APPC Programmers Guide* or the *Microsoft SNA Server CPI-C Programmers Guide* for more information.

If you do not specify a queue manager name, the default queue manager is assumed.

Defining a NetBIOS connection on Windows

WebSphere MQ uses three types of NetBIOS resource when establishing a NetBIOS connection to another WebSphere MQ product: sessions, commands, and names. Each of these resources has a limit, which is established either by default or by choice during the installation of NetBIOS.

Each running channel, regardless of type, uses one NetBIOS session and one NetBIOS command.
The IBM NetBIOS implementation allows multiple processes to use the same local NetBIOS name. Therefore, only one NetBIOS name needs to be available for use by WebSphere MQ. Other vendors' implementations, for example Novell's NetBIOS emulation, require a different local name per process. Verify your requirements from the documentation for the NetBIOS product you are using.

In all cases, ensure that sufficient resources of each type are already available, or increase the maximums specified in the configuration. Any changes to the values require a system restart.

During system startup, the NetBIOS device driver displays the number of sessions, commands, and names available for use by applications. These resources are available to any NetBIOS-based application that is running on the same system. Therefore, it is possible for other applications to consume these resources before WebSphere MQ needs to acquire them. Your LAN network administrator should be able to clarify this for you.

Related concepts

“Defining the IBM WebSphere MQ local NetBIOS name” on page 86
The local NetBIOS name used by IBM WebSphere MQ channel processes can be specified in three ways.

“Establishing the queue manager NetBIOS session, command, and name limits” on page 87
The queue manager limits for NetBIOS sessions, commands, and names can be specified in two ways.

“Establishing the LAN adapter number” on page 87
For channels to work successfully across NetBIOS, the adapter support at each end must be compatible. IBM WebSphere MQ allows you to control the choice of LAN adapter (LANA) number by using the AdapterNum value in the NETBIOS stanza of your qm.ini file and by specifying the `-a` parameter on the runmqlsr command.

“Initiating the NetBIOS connection” on page 87
Defining the steps needed to initiate a connection.

“Target listener for the NetBIOS connection” on page 88
Defining the steps to be undertaken at the receiving end of the NetBIOS connection.

Defining the IBM WebSphere MQ local NetBIOS name

The local NetBIOS name used by IBM WebSphere MQ channel processes can be specified in three ways.

In order of precedence the three ways are:

1. The value specified in the `-l` parameter of the RUNMQLSR command, for example:

   ```
   RUNMQLSR -t NETBIOS -l my_station
   ```

2. The MQNAME environment variable with a value that is established by the command:

   ```
   SET MQNAME=my_station
   ```

   You can set the MQNAME value for each process. Alternatively, you can set it at a system level in the Windows registry.

   If you are using a NetBIOS implementation that requires unique names, you must issue a SET MQNAME command in each window in which an IBM WebSphere MQ process is started. The MQNAME value is arbitrary but it must be unique for each process.

3. The NETBIOS stanza in the queue manager configuration file qm.ini. For example:

   ```
   NETBIOS:
     LocalName=my_station
   ```

Note:

1. Due to the variations in implementation of the NetBIOS products supported, you are advised to make each NetBIOS name unique in the network. If you do not, unpredictable results might occur. If you have problems establishing a NetBIOS channel and there are error messages in the queue-manager error log showing a NetBIOS return code of X'15', review your use of NetBIOS names.
2. On Windows, you cannot use your machine name as the NetBIOS name because Windows already uses it.
3. Sender channel initiation requires that a NetBIOS name be specified either by using the MQNAME environment variable or the LocalName in the qm.ini file.

Establishing the queue manager NetBIOS session, command, and name limits
The queue manager limits for NetBIOS sessions, commands, and names can be specified in two ways.

In order of precedence these ways are:

1. The values specified in the RUNMQLSR command:

```
-s  Sessions
-e  Names
-o  Commands
```

   If the -m operand is not specified in the command, the values apply only to the default queue manager.
2. The NETBIOS stanza in the queue manager configuration file qm.ini. For example:

```
NETBIOS:

  NumSess=Qmgr_max_sess
  NumCmds=Qmgr_max_cmds
  NumNames=Qmgr_max_names
```

Establishing the LAN adapter number
For channels to work successfully across NetBIOS, the adapter support at each end must be compatible. IBM WebSphere MQ allows you to control the choice of LAN adapter (LANA) number by using the AdapterNum value in the NETBIOS stanza of your qm.ini file and by specifying the -a parameter on the runmqlsr command.

The default LAN adapter number used by IBM WebSphere MQ for NetBIOS connections is 0. Verify the number being used on your system as follows:

On Windows, it is not possible to query the LAN adapter number directly through the operating system. Instead, you use the LANACFG.EXE command-line utility, available from Microsoft. The output of the tool shows the virtual LAN adapter numbers and their effective bindings. For further information about LAN adapter numbers, see the Microsoft Knowledge Base article 138037 *HOWTO: Use LANA Numbers in a 32-bit Environment*.

Specify the correct value in the NETBIOS stanza of the queue manager configuration file, qm.ini:

```
NETBIOS:
  AdapterNum=n
```

where n is the correct LAN adapter number for this system.

Initiating the NetBIOS connection
Defining the steps needed to initiate a connection.

To initiate the connection, follow these steps at the sending end:

1. Define the NetBIOS station name using the MQNAME or LocalName value.
2. Verify the LAN adapter number being used on your system and specify the correct file using the AdapterNum.
3. In the ConnectionName field of the channel definition, specify the NetBIOS name being used by the target listener program. On Windows, NetBIOS channels ***must*** be run as threads. Do this by specifying MCATYPE(THREAD) in the channel definition.

```
DEFINE CHANNEL (chname) CHLTYPE(SDR) +
       TRPTYPE(NETBIOS) +
       CONNAME(your_station) +
       XMITQ(xmitq) +
       MCATYPE(THREAD) +
       REPLACE
```

Target listener for the NetBIOS connection
Defining the steps to be undertaken at the receiving end of the NetBIOS connection.

At the receiving end, follow these steps:

1. Define the NetBIOS station name using the MQNAME or LocalName value.
2. Verify the LAN adapter number being used on your system and specify the correct file using the AdapterNum.
3. Define the receiver channel:

```
DEFINE CHANNEL (chname) CHLTYPE(RCVR) +
       TRPTYPE(NETBIOS) +
       REPLACE
```

4. Start the WebSphere MQ listener program to establish the station and make it possible to contact it. For example:

```
RUNMQLSR -t NETBIOS -l your_station [-m qmgr]
```

   This command establishes `your_station` as a NetBIOS station waiting to be contacted. The NetBIOS station name must be unique throughout your NetBIOS network.

For the best performance, run the WebSphere MQ listener as a trusted application as described in “Running channels and listeners as trusted applications” on page 71. See Restrictions for trusted applications for information about trusted applications.

You can stop all WebSphere MQ listeners running on a queue manager that is inactive, using the command:

```
ENDMQLSR [-m QMNAME]
```

If you do not specify a queue manager name, the default queue manager is assumed.

Defining an SPX connection on Windows

An SPX connection applies only to a client and server running Windows XP and Windows 2003 Server.

The channel definition at the sending end specifies the address of the target. A listener program must be run at the receiving end.

Related concepts

“Sending end on SPX” on page 88
If the target machine is remote, specify the SPX address of the target machine in the Connection name field of the channel definition.

“Receiving on SPX” on page 89
Receiving channel programs are started in response to a startup request from the sending channel.

“IPX/SPX parameters” on page 90
In most cases, the default settings for the IPX/SPX parameters will suit your needs. However, you might need to modify some of them in your environment to tune its use for WebSphere MQ.

Sending end on SPX
If the target machine is remote, specify the SPX address of the target machine in the Connection name field of the channel definition.

The SPX address is specified in the following form:

```
        network.node(socket)
```

where:

network
Is the 4-byte network address of the network on which the remote machine resides,

node
Is the 6-byte node address, which is the LAN address of the LAN adapter in the remote machine

socket
Is the 2-byte socket number on which the remote machine listens.

If the local and remote machines are on the same network then the network address need not be specified. If the remote end is listening on the default socket (5E86) then the socket need not be specified.

An example of a fully specified SPX address specified in the CONNAME parameter of an MQSC command is:

```
      CONNAME('00000001.08005A7161E5(5E87)')
```

In the default case, where the machines are both on the same network, this becomes:

```
      CONNAME(08005A7161E5)
```

The default socket number can be changed by specifying it in the queue manager configuration file (qm.ini):

```
SPX:
  Socket=5E87
```

For more information about the values you set using qm.ini, see Configuration file stanzas for distributed queuing .

Receiving on SPX
Receiving channel programs are started in response to a startup request from the sending channel.

To start a receiving channel program, a listener program must be started to detect incoming network requests and start the associated channel.

Use the WebSphere MQ listener.

Using the SPX listener backlog option

When receiving on SPX, a maximum number of outstanding connection requests is set. This can be considered a *backlog* of requests waiting on the SPX port for the listener to accept the request. The default listener backlog values are shown in Table 10 on page 89.

Table 10. Default outstanding connection requests on Windows

Platform	**Default listener backlog value**
Windows Server	5
Windows Workstation	5

If the backlog reaches the values in Table 10 on page 89, the reason code, MQRC_Q_MGR_NOT_AVAILABLE is received when trying to connect to the queue manager using MQCONN or MQCONNX. If this happens, it is possible to try to connect again.

However, to avoid this error, you can add an entry in the qm.ini file or in the registry for Windows:

```
SPX:
ListenerBacklog = n
```

This overrides the default maximum number of outstanding requests (see Table 10 on page 89) for the SPX listener.

Note: Some operating systems support a larger value than the default. If necessary, this can be used to avoid reaching the connection limit.

To run the listener with the `backlog` option switched on either:

- Use the RUNMQLSR -b command, or
- Use the MQSC command **DEFINE LISTENER** with the BACKLOG attribute set to the required value.

For information about the **RUNMQLSR** command, see runmqlsr. For information about the DEFINE LISTENER command, see DEFINE LISTENER.

Using the WebSphere MQ listener

To run the Listener supplied with WebSphere MQ, that starts new channels as threads, use the RUNMQLSR command. For example:

```
RUNMQLSR -t spx [-m QMNAME] [-x 5E87]
```

The square brackets indicate optional parameters; QMNAME is not required for the default queue manager, and the socket number is not required if you are using the default (5E86).

For the best performance, run the WebSphere MQ listener as a trusted application as described in “Running channels and listeners as trusted applications” on page 71. See Restrictions for trusted applications for more information about trusted applications.

You can stop all WebSphere MQ listeners running on a queue manager that is inactive, using the command:

```
ENDMQLSR [-m QMNAME]
```

If you do not specify a queue manager name, the default queue manager is assumed.

IPX/SPX parameters
In most cases, the default settings for the IPX/SPX parameters will suit your needs. However, you might need to modify some of them in your environment to tune its use for WebSphere MQ.

The actual parameters and the method of changing them varies according to the platform and provider of SPX communications support. The example section describes some of these parameters, particularly those that might influence the operation of WebSphere MQ channels and client connections.

Windows systems

Refer to the Microsoft documentation for full details of the use and setting of the NWLink IPX and SPX parameters. The IPX/SPX parameters are in the following paths in the registry:

```
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Service\NWLinkSPX\Parameters
HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Service\NWLinkIPX\Parameters
```

Setting up communication on UNIX and Linux systems

DQM is a remote queuing facility for IBM WebSphere MQ. It provides channel control programs for the queue manager which form the interface to communication links, controllable by the system operator. The channel definitions held by distributed-queuing management use these connections.

When a distributed-queuing management channel is started, it tries to use the connection specified in the channel definition. To succeed, it is necessary for the connection to be defined and available. This section explains how to do this. You might also find it helpful to refer to the following sections:

- Example configuration - IBM WebSphere MQ for AIX
- Example configuration - IBM WebSphere MQ for HP-UX
- Example configuration - IBM WebSphere MQ for Solaris
- Example configuration - IBM WebSphere MQ for Linux

For Windows, see “Setting up communication for Windows” on page 81.

You can choose between two forms of communication for WebSphere MQ on UNIX and Linux systems:

- “Defining a TCP connection on UNIX and Linux” on page 91
- “Defining an LU 6.2 connection on UNIX and Linux” on page 94

Each channel definition must specify one only as the transmission protocol (Transport Type) attribute. One or more protocols can be used by a queue manager.

For IBM WebSphere MQ Explorer MQI clients, it might be useful to have alternative channels using different transmission protocols. For more information about IBM WebSphere MQ Explorer MQI clients, see Overview of IBM WebSphere MQ MQI clients .

Related concepts

“Connecting applications using distributed queuing” on page 27
This section provides more detailed information about intercommunication between WebSphere MQ installations, including queue definition, channel definition, triggering, and sync point procedures

“Monitoring and controlling channels on UNIX, Linux, and Windows” on page 72
For DQM you need to create, monitor, and control the channels to remote queue managers. You can control channels using commands, programs, IBM WebSphere MQ Explorer, files for the channel definitions, and a storage area for synchronization information.

“Configuring connections between the client and server” on page 96
To configure the communication links between WebSphere MQ MQI clients and servers, decide on your communication protocol, define the connections at both ends of the link, start a listener, and define channels.

Defining a TCP connection on UNIX and Linux

The channel definition at the sending end specifies the address of the target. The listener or inet daemon is configured for the connection at the receiving end.

Sending end

Specify the host name, or the TCP address of the target machine, in the Connection Name field of the channel definition. The port to connect to defaults to 1414. Port number 1414 is assigned by the Internet Assigned Numbers Authority to WebSphere MQ.

To use a port number other than the default, change the connection name field thus:

```
Connection Name REMHOST(1822)
```

where REMHOST is the host name of the remote machine and 1822 is the port number required. (This must be the port that the listener at the receiving end is listening on.)

Alternatively you can change the port number by specifying it in the queue manager configuration file (qm.ini):

```
TCP:
  Port=1822
```

For more information about the values you set using qm.ini, see Configuration file stanzas for distributed queuing.

Receiving on TCP

You can use either the TCP/IP listener, which is the inet daemon (inetd), or the WebSphere MQ listener.

Some Linux distributions now use the extended inet daemon (xinetd) instead of the inet daemon. For information about how to use the extended inet daemon on a Linux system, see Establishing a TCP connection on Linux.

Related concepts

"Using the TCP/IP listener" on page 92
To start channels on UNIX and Linux, the /etc/services file and the inetd.conf file must be edited

"Using the TCP listener backlog option" on page 93
In TCP, connections are treated incomplete unless three-way handshake takes place between the server and the client. These connections are called outstanding connection requests. A maximum value is set for these outstanding connection requests and can be considered a backlog of requests waiting on the TCP port for the listener to accept the request.

"Using the WebSphere MQ listener" on page 94
To run the listener supplied with WebSphere MQ, which starts new channels as threads, use the `runmqlsr` command.

"Using the TCP/IP SO_KEEPALIVE option" on page 94
On some UNIX and Linux systems, you can define how long TCP waits before checking that the connection is still available, and how frequently it tries the connection again if the first check fails. This is either a kernel tunable parameter, or can be entered at the command line.

Using the TCP/IP listener

To start channels on UNIX and Linux, the /etc/services file and the inetd.conf file must be edited

Follow these instructions:

1. Edit the /etc/services file:

   Note: To edit the /etc/services file, you must be logged in as a superuser or root. You can change this, but it must match the port number specified at the sending end.

   Add the following line to the file:

   ```
   MQSeries 1414/tcp
   ```

   where 1414 is the port number required by WebSphere MQ. The port number must not exceed 65535.
2. Add a line in the inetd.conf file to call the program amqcrsta, where *MQ_INSTALLATION_PATH* represents the high-level directory in which WebSphere MQ is installed:

   ```
   MQSeries stream tcp nowait mqm MQ_INSTALLATION_PATH/bin/amqcrsta amqcrsta
   [-m Queue_Man_Name]
   ```

The updates are active after inetd has reread the configuration files. To do this, issue the following commands from the root user ID:

- On AIX:

  ```
  refresh -s inetd
  ```

- On HP-UX, from the mqm user ID:

  ```
  inetd -c
  ```

- On Solaris 10 or later:

  ```
  inetconv
  ```

- On other UNIX and Linux systems (including Solaris 9):

```
kill -1 <process number>
```

When the listener program started by inetd inherits the locale from inetd, it is possible that the MQMDE is not honored (merged) and is placed on the queue as message data. To ensure that the MQMDE is honored, you must set the locale correctly. The locale set by inetd might not match that chosen for other locales used by WebSphere MQ processes. To set the locale:

1. Create a shell script which sets the locale environment variables LANG, LC_COLLATE, LC_CTYPE, LC_MONETARY, LC_NUMERIC, LC_TIME, and LC_MESSAGES to the locale used for other WebSphere MQ process.
2. In the same shell script, call the listener program.
3. Modify the inetd.conf file to call your shell script in place of the listener program.

It is possible to have more than one queue manager on the server. You must add a line to each of the two files, for each of the queue managers. For example:

```
MQSeries1     1414/tcp
MQSeries2     1822/tcp
```

```
MQSeries2 stream tcp nowait mqm MQ_INSTALLATION_PATH/bin/amqcrsta amqcrsta -m QM2
```

Where *MQ_INSTALLATION_PATH* represents the high-level directory in which WebSphere MQ is installed.

This avoids error messages being generated if there is a limitation on the number of outstanding connection requests queued at a single TCP port. For information about the number of outstanding connection requests, see “Using the TCP listener backlog option” on page 93.

Using the TCP listener backlog option
In TCP, connections are treated incomplete unless three-way handshake takes place between the server and the client. These connections are called outstanding connection requests. A maximum value is set for these outstanding connection requests and can be considered a backlog of requests waiting on the TCP port for the listener to accept the request.

The default listener backlog values are shown in Table 11 on page 93.

Table 11. Maximum outstanding connection requests queued at a TCP/IP port

Server platform	**Maximum connection requests**
AIX	100
HP-UX	20
Linux	100
IBM i	255
Solaris	100
Windows Server	100
Windows Workstation	100
z/OS	255

If the backlog reaches the values shown in Table 11 on page 93, the TCP/IP connection is rejected and the channel is not able to start.

For MCA channels, this results in the channel going into a RETRY state and trying the connection again at a later time.

However, to avoid this error, you can add an entry in the qm.ini file:

```
TCP:
ListenerBacklog = n
```

This overrides the default maximum number of outstanding requests (see Table 11 on page 93) for the TCP/IP listener.

Note: Some operating systems support a larger value than the default. If necessary, this value can be used to avoid reaching the connection limit.

To run the listener with the `backlog` option switched on either:

- Use the `runmqlsr -b` command, or
- Use the MQSC command **`DEFINE LISTENER`** with the BACKLOG attribute set to the required value.

For information about the **`runmqlsr`** command, see runmqlsr. For information about the DEFINE LISTENER command, see the DEFINE LISTENER.

Using the WebSphere MQ listener
To run the listener supplied with WebSphere MQ, which starts new channels as threads, use the `runmqlsr` command.

For example:

```
runmqlsr -t tcp [-m QMNAME] [-p 1822]
```

The square brackets indicate optional parameters; QMNAME is not required for the default queue manager, and the port number is not required if you are using the default (1414). The port number must not exceed 65535.

For the best performance, run the WebSphere MQ listener as a trusted application as described in “Running channels and listeners as trusted applications” on page 71. See Restrictions for trusted applications for information about trusted applications.

You can stop all WebSphere MQ listeners running on a queue manager that is inactive, using the command:

```
endmqlsr [-m QMNAME]
```

If you do not specify a queue manager name, the default queue manager is assumed.

Using the TCP/IP SO_KEEPALIVE option
On some UNIX and Linux systems, you can define how long TCP waits before checking that the connection is still available, and how frequently it tries the connection again if the first check fails. This is either a kernel tunable parameter, or can be entered at the command line.

If you want to use the SO_KEEPALIVE option (for more information, see “Checking that the other end of the channel is still available” on page 60) you must add the following entry to your queue manager configuration file (qm.ini):

```
TCP:
   KeepAlive=yes
```

See the documentation for your UNIX and Linux system for more information.

Defining an LU 6.2 connection on UNIX and Linux

SNA must be configured so that an LU 6.2 conversation can be established between the two machines.

For the latest information about configuring SNA over TCP/IP, see the following online IBM documentation: Communications Server.

SNA must be configured so that an LU 6.2 conversation can be established between the two systems.

See the *Multiplatform APPC Configuration Guide* and the following table for information.

Table 12. Settings on the local UNIX and Linux system for a remote queue manager platform

Remote platform	TPNAME	TPPATH
z/OS without CICS	The same as the corresponding TPName in the side information about the remote queue manager.	-
z/OS using CICS	CKRC (sender) CKSV (requester) CKRC (server)	-
IBM i	The same as the compare value in the routing entry on the IBM i system.	-
UNIX and Linux systems	The same as the corresponding TPName in the side information about the remote queue manager.	*MQ_INSTALLATION_PATH*/bin/amqcrs6a
Windows	As specified in the Windows Run Listener command, or the invokable Transaction Program that was defined using TpSetup on Windows.	*MQ_INSTALLATION_PATH*\bin\amqcrs6a

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

If you have more than one queue manager on the same machine, ensure that the TPnames in the channel definitions are unique.

Related concepts

"Sending end" on page 95
On UNIX and Linux systems, create a CPI-C side object (symbolic destination) and enter this name in the Connection name field in the channel definition. Also create an LU 6.2 link to the partner.

"Receiving on LU 6.2" on page 95
On UNIX and Linux systems, create a listening attachment at the receiving end, an LU 6.2 logical connection profile, and a TPN profile.

Sending end

On UNIX and Linux systems, create a CPI-C side object (symbolic destination) and enter this name in the Connection name field in the channel definition. Also create an LU 6.2 link to the partner.

In the CPI-C side object enter the partner LU name at the receiving machine, the transaction program name and the mode name. For example:

```
Partner LU Name              REMHOST
Remote TP Name               recv
Service Transaction Program  no
Mode Name                    #INTER
```

On HP-UX, use the APPCLLU environment variable to name the local LU that the sender should use. On Solaris, set the APPC_LOCAL_LU environment variable to be the local LU name.

SECURITY PROGRAM is used, where supported by CPI-C, when WebSphere MQ attempts to establish an SNA session.

Receiving on LU 6.2

On UNIX and Linux systems, create a listening attachment at the receiving end, an LU 6.2 logical connection profile, and a TPN profile.

In the TPN profile, enter the full path to the executable file and the Transaction Program name:

```
Full path to TPN executable   MQ_INSTALLATION_PATH/bin/amqcrs6a
Transaction Program name      recv
User ID                       0
```

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

On systems where you can set the user ID, specify a user who is a member of the mqm group. On AIX, Solaris, and HP-UX, set the APPCTPN (transaction name) and APPCLLU (local LU name) environment variables (you can use the configuration panels for the invoked transaction program).

You might need to use a queue manager other than the default queue manager. If so, define a command file that calls:

```
amqcrs6a -m Queue_Man_Name
```

then call the command file.

Configuring connections between the client and server

To configure the communication links between WebSphere MQ MQI clients and servers, decide on your communication protocol, define the connections at both ends of the link, start a listener, and define channels.

In WebSphere MQ, the logical communication links between objects are called *channels*. The channels used to connect WebSphere MQ MQI clients to servers are called MQI channels. You set up channel definitions at each end of your link so that your WebSphere MQ application on the WebSphere MQ MQI client can communicate with the queue manager on the server. For a detailed description of how to do this, see User defined channels.

Before you define your MQI channels, you must:

1. Decide on the form of communication you are going to use. See “Which communication type to use” on page 96.
2. Define the connection at each end of the channel:

   To define the connection, you must:

   - Configure the connection.
   - Record the values of the parameters that you need for the channel definitions.
   - Enable the server to detect incoming network requests from your WebSphere MQ MQI client, by starting a *listener*.

Which communication type to use

Different platforms support different transmission protocols. Your choice of transmission protocol depends on your combination of WebSphere MQ MQI client and server platforms.

There are up to four types of transmission protocol for MQI channels depending on your client and server platforms:

- LU 6.2
- NetBIOS
- SPX
- TCP/IP

When you define your MQI channels, each channel definition must specify a transmission protocol (transport type) attribute. A server is not restricted to one protocol, so different channel definitions can specify different protocols. For WebSphere MQ MQI clients, it might be useful to have alternative MQI channels using different transmission protocols.

Your choice of transmission protocol might be restricted by your particular combination of WebSphere MQ MQI client and server platforms. The possible combinations are shown in the following table.

Table 13. Transmission protocols - combination of WebSphere MQ MQI client and server platforms		
Transmission protocol	**WebSphere MQ MQI client**	**WebSphere MQ server**
TCP/IP	UNIX systems Windows	UNIX systems Windows z/OS
LU 6.2	UNIX systems[1] Windows	UNIX systems[1] Windows
NetBIOS	Windows	Windows
SPX	Windows	Windows
Note: 1. Except Linux for Power Systems		

For more about setting up different types of connections, see the following links:

- “Defining a TCP connection on Windows ” on page 82
- “Defining a TCP connection on UNIX and Linux” on page 91
- “TCP/IP connection limits” on page 99
- “Defining an LU 6.2 connection on Windows” on page 84
- “Defining an LU 6.2 connection on UNIX and Linux” on page 94
- “Defining a NetBIOS connection on Windows” on page 85
- “Defining an SPX connection on Windows” on page 88

Related concepts

“Configuring an extended transactional client” on page 100
This collection of topics describes how to configure the extended transactional function for each category of transaction manager.

“Defining MQI channels” on page 109
To create a new channel, you have to create **two** channel definitions, one for each end of the connection, using the same channel name and compatible channel types. In this case, the channel types are *server-connection* and *client-connection*.

“Creating server-connection and client-connection definitions on different platforms” on page 110
You can create each channel definition on the computer to which it applies. There are restrictions on how you can create channel definitions on a client computer.

“Creating server-connection and client-connection definitions on the server” on page 113
You can create both definitions on the server, then make the client-connection definition available to the client.

“Channel-exit programs for MQI channels” on page 118
Three types of channel exit are available to the WebSphere MQ MQI client environment on UNIX, Linux and Windows systems.

“Connecting a client to a queue-sharing group” on page 122
You can connect a client to a queue-sharing group by creating an MQI channel between a client and a queue manager on a server that is a member of a queue-sharing group.

“Configuring a client using a configuration file” on page 123

Configure your clients using attributes in a text file. These attributes can be overridden by environment variables or in other platform-specific ways.

Related tasks

Connecting IBM MQ MQI client applications to queue managers

Related reference

DISPLAY CHLAUTH

SET CHLAUTH

Which communication type to use

Different platforms support different communication protocols. Your choice of transmission protocol depends on your combination of WebSphere MQ MQI client and server platforms.

There are four types of communication for MQI channels on different platforms:

- LU 6.2
- NetBIOS
- SPX
- TCP/IP

When you define your MQI channels, each channel definition must specify a transmission protocol (transport type) attribute. A server is not restricted to one protocol, so different channel definitions can specify different protocols. For WebSphere MQ MQI clients, it might be useful to have alternative MQI channels using different transmission protocols.

Your choice of transmission protocol also depends on your particular combination of WebSphere MQ client and server platforms. The possible combinations are shown in the following table.

Table 14. Transmission protocols - combination of WebSphere MQ client and server platforms

Transmission protocol	WebSphere MQ MQI client	WebSphere MQ server
TCP/IP	UNIX systems Windows	UNIX systems Windows
LU 6.2	UNIX systems[1] Windows	UNIX systems[1] Windows
NetBIOS	Windows	Windows
SPX	Windows	Windows

Note:

1. Except Linux (POWER® platform)

Related concepts

“Defining a TCP connection on Windows ” on page 82

Define a TCP connection by configuring a channel at the sending end to specify the address of the target, and by running a listener program at the receiving end.

“Defining a TCP connection on UNIX and Linux” on page 91

The channel definition at the sending end specifies the address of the target. The listener or inet daemon is configured for the connection at the receiving end.

“Defining an LU 6.2 connection on Windows” on page 84

SNA must be configured so that an LU 6.2 conversation can be established between the two machines.

"Defining an LU 6.2 connection on UNIX and Linux" on page 94
SNA must be configured so that an LU 6.2 conversation can be established between the two machines.

"Defining a NetBIOS connection on Windows" on page 85
WebSphere MQ uses three types of NetBIOS resource when establishing a NetBIOS connection to another WebSphere MQ product: sessions, commands, and names. Each of these resources has a limit, which is established either by default or by choice during the installation of NetBIOS.

"Defining an SPX connection on Windows" on page 88
An SPX connection applies only to a client and server running Windows XP and Windows 2003 Server.

Related reference

"TCP/IP connection limits" on page 99
The number of outstanding connection requests that can be queued at a single TCP/IP port depends on the platform. An error occurs if the limit is reached.

Defining a TCP/IP connection

Specifying a transport type of TCP on the channel definition at the client end. Start a listener program on the server.

Specify a TCP/IP connection at the client by specifying a transport type of TCP on the channel definition.

Receiving channel programs are started in response to a startup request from the sending channel. To do this, a listener program has to be started to detect incoming network requests and start the associated channel. The procedure for starting a listener program depends on the server platform.

See the related topics for your client and server platforms.

TCP/IP connection limits

The number of outstanding connection requests that can be queued at a single TCP/IP port depends on the platform. An error occurs if the limit is reached.

This connection limit is not the same as the maximum number of clients you can attach to an IBM WebSphere MQ server. You can connect more clients to a server, up to the level determined by the server system resources. The backlog values for connection requests are shown in the following table:

Table 15. Maximum outstanding connection requests queued at a TCP/IP port

Server platform	**Maximum connection requests**
AIX	100
HP-UX	20
Linux	100
IBM	255
Solaris	100
Windows Server	100
Windows Workstation	100
z/OS	255

If the connection limit is reached, the client receives a return code of MQRC_HOST_NOT_AVAILABLE from the `MQCONN` call, and an AMQ9202 error in the client error log (`/var/mqm/errors/AMQERR0n.LOG` on UNIX and Linux systems or `amqerr0n.log` in the errors subdirectory of the IBM WebSphere MQ client installation on Windows). If the client retries the `MQCONN` request, it might be successful.

To increase the number of connection requests you can make, and avoid error messages being generated by this limitation, you can have multiple listeners each listening on a different port, or have more than one queue manager.

Defining a NetBIOS or SPX connection

NetBIOS and SPX connections apply only to Windows systems.

A NetBIOS connection applies only to a client and server running Windows. See Defining a NetBIOS connection.

An SPX connection applies only to a client and server running Windows XP or Windows 2003 Server. See Defining an SPX connection.

Configuring an extended transactional client

This collection of topics describes how to configure the extended transactional function for each category of transaction manager.

For each platform, the extended transactional client provides support for the following external transaction managers:

XA-compliant transaction managers
The extended transactional client provides the XA resource manager interface to support XA-compliant transaction managers such as CICS and Tuxedo.

Microsoft Transaction Server (Windows systems only)
On Windows systems only, the XA resource manager interface also supports Microsoft Transaction Server (MTS). The WebSphere MQ MTS support supplied with the extended transactional client provides the bridge between MTS and the XA resource manager interface.

WebSphere Application Server
Earlier versions of WebSphere MQ supported WebSphere Application Server Version 4 or Version 5, and required you to carry out certain configuration tasks to use the extended transactional client. WebSphere Application Server Version 6 and later includes a WebSphere MQ messaging provider, so you do not need to use the extended transactional client.

Related concepts

“Configuring XA-compliant transaction managers” on page 100
First configure the WebSphere MQ base client, then configure the extended transactional function using the information in these topics.

“ Microsoft Transaction Server” on page 108
No additional configuration is required before you can use MTS as a transaction manager. However, there are some points to note.

Configuring XA-compliant transaction managers

First configure the WebSphere MQ base client, then configure the extended transactional function using the information in these topics.

Note: This section assumes that you have a basic understanding of the XA interface as published by The Open Group in *Distributed Transaction Processing: The XA Specification*.

To configure an extended transactional client, you must first configure the WebSphere MQ base client as described in Installing an IBM WebSphere MQ client . Using the information in this section, you can then configure the extended transactional function for an XA-compliant transaction manager such as CICS and Tuxedo.

A transaction manager communicates with a queue manager as a resource manager using the same MQI channel as that used by the client application that is connected to the queue manager. When the transaction manager issues a resource manager (xa_) function call, the MQI channel is used to forward the call to the queue manager, and to receive the output back from the queue manager.

Either the transaction manager can start the MQI channel by issuing an xa_open call to open the queue manager as a resource manager, or the client application can start the MQI channel by issuing an MQCONN or MQCONNX call.

- If the transaction manager starts the MQI channel, and the client application later calls MQCONN or MQCONNX on the same thread, the MQCONN or MQCONNX call completes successfully and a connection handle is returned to the application. The application does not receive a MQCC_WARNING completion code with an MQRC_ALREADY_CONNECTED reason code.
- If the client application starts the MQI channel, and the transaction manager later calls xa_open on the same thread, the xa_open call is forwarded to the queue manager using the MQI channel.

In a recovery situation following a failure, when no client applications are running, the transaction manager can use a dedicated MQI channel to recover any incomplete units of work in which the queue manager was participating at the time of the failure.

Note the following conditions when using an extended transactional client with an XA-compliant transaction manager:

- Within a single thread, a client application can be connected to only one queue manager at a time. This restriction applies only when using an extended transactional client; a client application that is using a WebSphere MQ base client can be connected to more than one queue manager concurrently within a single thread.
- Each thread of a client application can connect to a different queue manager.
- A client application cannot use shared connection handles.

To configure the extended transactional function, you must provide the following information to the transaction manager for each queue manager that acts as a resource manager:

- An xa_open string
- A pointer to an XA switch structure

When the transaction manager calls xa_open to open the queue manager as a resource manager, it passes the xa_open string to the extended transactional client as the argument, *xa_info*, on the call. The extended transactional client uses the information in the xa_open string in the following ways:

- To start an MQI channel to the server queue manager, if the client application has not already started one
- To check that the queue manager that the transaction manager opens as a resource manager is the same as the queue manager to which the client application connects
- To locate the transaction manager's ax_reg and ax_unreg functions, if the queue manager uses dynamic registration

For the format of an xa_open string, and for more details about how the information in the xa_open string is used by an extended transactional client, see “The format of an xa_open string” on page 102.

An XA switch structure enables the transaction manager to locate the xa_ functions provided by the extended transactional client, and specifies whether the queue manager uses dynamic registration. For information about the XA switch structures supplied with an extended transactional client, see “The XA switch structures” on page 105.

For information about how to configure the extended transactional function for a particular transaction manager, and for any other information about using the transaction manager with an extended transactional client, see the following sections:

- “Configuring an extended transactional client for CICS” on page 107
- “Configuring an extended transactional client for Tuxedo” on page 108

Related concepts

“The CHANNEL, TRPTYPE, CONNAME, and QMNAME parameters of the xa_open string” on page 104

Use this information to understand how the extended transactional client uses these parameters to determine the queue manager to connect to.

“Additional error processing for xa_open” on page 105
The xa_open call fails in certain circumstances.

Related tasks

“Using the extended transactional client with SSL channels” on page 106
You cannot set up an SSL channel using the xa_open string. Follow these instructions to use the client channel definition table (ccdt).

Related reference

“The TPM and AXLIB parameters” on page 104
An extended transactional client uses the TPM and AXLIB parameters to locate the transaction manager's ax_reg and ax_unreg functions. These functions are used only if the queue manager uses dynamic registration.

“Recovery following a failure in extended transactional processing” on page 105
Following a failure, a transaction manager must be able to recover any incomplete units of work. To do this, the transaction manager must be able to open as a resource manager any queue manager that was participating in an incomplete unit of work at the time of the failure.

The format of an xa_open string

An xa_open string contains pairs of defined parameter names and values.

An xa_open string has the following format:

```
parm_name1=parm_value1,parm_name2=parm_value2, ...
```

where *parm_name* is the name of a parameter and *parm_value* is the value of a parameter. The names of the parameters are not case-sensitive but, unless stated otherwise, the values of the parameters are case-sensitive. You can specify the parameters in any order.

The names, meanings, and valid values of the parameters are as follows:

Name
Meaning and valid values

CHANNEL
The name of an MQI channel.

This is an optional parameter. If this parameter is supplied, the CONNAME parameter must also be supplied.

TRPTYPE
The communications protocol for the MQI channel. The following are valid values:

LU62
SNA LU 6.2

NETBIOS
NetBIOS

SPX
IPX/SPX

TCP
TCP/IP

This is an optional parameter. If it is omitted, the default value of TCP is assumed. The values of the parameter are not case-sensitive.

CONNAME
The network address of the queue manager at the server end of the MQI channel. The valid values of this parameter depend on the value of the TRPTYPE parameter:

LU62
A symbolic destination name, which identifies a CPI-C side information entry.

The network qualified name of a partner LU is not a valid value, nor is a partner LU alias. This is because there are no additional parameters to specify a transaction program (TP) name and a mode name.

NETBIOS
A NetBIOS name.

SPX
A 4 byte network address, a 6 byte node address, and an optional 2 byte socket number. These values must be specified in hexadecimal notation. A period must separate the network and node addresses, and the socket number, if supplied, must be enclosed in parentheses. For example:

```
0a0b0c0d.804abcde23a1(5e86)
```

If the socket number is omitted, the default value of 5e86 is assumed.

TCP
A host name or an IP address, optionally followed by a port number in parentheses. If the port number is omitted, the default value of 1414 is assumed.

This is an optional parameter. If this parameter is supplied, the CHANNEL parameter must also be supplied.

QMNAME
The name of the queue manager at the server end of the MQI channel. The name cannot be blank or a single asterisk (*), nor can the name start with an asterisk. This means that the parameter must identify a specific queue manager by name.

This is a mandatory parameter.

When a client application is connected to a specific queue manager any transaction recovery must be processed by the same queue manager.

If the application is connecting to a z/OS queue manager then the application can specify either the name of a specific queue manager or the name of a queue-sharing group (QSG). By using the queue manager name or QSG name, the application controls whether it partakes in a transaction with a QMGR unit of recovery disposition or a GROUP unit of recovery disposition. The GROUP unit of recovery disposition enables the recovery of the transaction to be processed on any member of the QSG. To use GROUP units of recovery the **GROUPUR** queue manager attribute must be enabled.

TPM
The transaction manager being used. The valid values are CICS and TUXEDO.

An extended transactional client uses this parameter and the AXLIB parameter for the same purpose. For more information these parameters, see The TPM and AXLIB parameters.

This is an optional parameter. The values of the parameter are not case-sensitive.

AXLIB
The name of the library that contains the transaction manager's ax_reg and ax_unreg functions.

This is an optional parameter.

Here is an example of an xa_open string:

```
channel=MARS.SVR,trptype=tcp,conname=MARS(1415),qmname=MARS,tpm=cics
```

The CHANNEL, TRPTYPE, CONNAME, and QMNAME parameters of the xa_open string

Use this information to understand how the extended transactional client uses these parameters to determine the queue manager to connect to.

If the CHANNEL and CONNAME parameters are supplied in the xa_open string, the extended transactional client uses these parameters and the TRPTYPE parameter to start an MQI channel to the server queue manager.

If the CHANNEL and CONNAME parameters are not supplied in the xa_open string, the extended transactional client uses the value of the MQSERVER environment variable to start an MQI channel. If the MQSERVER environment variable is not defined, the extended transactional client uses the entry in the client channel definition identified by the QMNAME parameter.

In each of these cases, the extended transactional client checks that the value of the QMNAME parameter is the name of the queue manager at the server end of the MQI channel. If it is not, the xa_open call fails and the transaction manager reports the failure to the application.

If the application client is connecting to a z/OS queue manager at V7.0.1 or later it can specify a queue-sharing group (QSG) name for the QMNAME parameter. This allows the application client to participate in a transaction with a GROUP unit of recovery disposition.

If the application uses a QSG name in QMNAME parameter field and the GROUPUR property is disabled on the queue manager to which it connects then the xa_open call fails.

If the application connects to a queue manager at an earlier version than V7.0.1, the xa_open call succeeds but the transaction has a QMGR unit of recovery disposition. Ensure that applications that require the GROUP unit of recovery disposition connect only to queue managers at V7.0.1 or later.

When the client application later calls MQCONN or MQCONNX on the same thread that the transaction manager used to issue the xa_open call, the application receives a connection handle for the MQI channel that was started by the xa_open call. A second MQI channel is not started. The extended transactional client checks that the value of the *QMgrName* parameter on the MQCONN or MQCONNX call is the name of the queue manager at the server end of the MQI channel. If it is not, the MQCONN or MQCONNX call fails with a reason code of MQRC_ANOTHER_Q_MGR_CONNECTED. If the value of the *QMgrName* parameter is blank or a single asterisk (*), or starts with an asterisk, the MQCONN or MQCONNX call fails with a reason code of MQRC_Q_MGR_NAME_ERROR.

If the client application has already started an MQI channel by calling MQCONN or MQCONNX before the transaction manager calls xa_open on the same thread, the transaction manager uses this MQI channel instead. A second MQI channel is not started. The extended transactional client checks that the value of the QMNAME parameter in the xa_open string is the name of the server queue manager. If it is not, the xa_open call fails.

If a client application starts an MQI channel first, the value of the *QMgrName* parameter on the MQCONN or MQCONNX call can be blank or a single asterisk (*), or it can start with an asterisk. Under these circumstances, however, you must ensure that the queue manager to which the application connects is the same as the queue manager that the transaction manager intends to open as a resource manager when it later calls xa_open on the same thread. You might encounter fewer problems, therefore, if the value of the *QMgrName* parameter identifies the queue manager explicitly by name.

The TPM and AXLIB parameters

An extended transactional client uses the TPM and AXLIB parameters to locate the transaction manager's ax_reg and ax_unreg functions. These functions are used only if the queue manager uses dynamic registration.

If the TPM parameter is supplied in an xa_open string, but the AXLIB parameter is not supplied, the extended transactional client assumes a value for the AXLIB parameter based on the value of the TPM parameter. See Table 16 on page 105 for the assumed values of the AXLIB parameter.

Table 16. Assumed values of the AXLIB parameter		
Value of TPM	**Platform**	**Assumed value of AXLIB**
CICS	AIX	/usr/lpp/encina/lib/libEncServer.a(EncServer_shr.o)
CICS	HP-UX	/opt/encina/lib/libEncServer.sl
CICS	Solaris	/opt/encina/lib/libEncServer.so
CICS	Windows systems	libEncServer
Tuxedo	AIX	/usr/lpp/tuxedo/lib/libtux.a(libtux.so.60)
Tuxedo	HP-UX	/opt/tuxedo/lib/libtux.sl
Tuxedo	Solaris	/opt/tuxedo/lib/libtux.so.60
Tuxedo	Windows systems	libtux

If the AXLIB parameter is supplied in an xa_open string, the extended transactional client uses its value to override any assumed value based on the value of the TPM parameter. The AXLIB parameter can also be used for a transaction manager for which the TPM parameter does not have a specified value.

Additional error processing for xa_open

The xa_open call fails in certain circumstances.

Topics in this section describe situations in which the xa_open call fails. It also fails if any of the following situations occur:

- There are errors in the xa_open string.
- There is insufficient information to start an MQI channel.
- There is a problem while trying to start an MQI channel (the server queue manager is not running, for example).

Recovery following a failure in extended transactional processing

Following a failure, a transaction manager must be able to recover any incomplete units of work. To do this, the transaction manager must be able to open as a resource manager any queue manager that was participating in an incomplete unit of work at the time of the failure.

If you ever need to change any configuration information, therefore, you must ensure that all incomplete units of work have been resolved before making the changes. Alternatively, you must ensure that the configuration changes do not affect the ability of the transaction manager to open the queue managers it needs to open. The following are examples of such configuration changes:

- Changing the contents of an xa_open string
- Changing the value of the MQSERVER environment variable
- Changing entries in the client channel definition table (CCDT)
- Deleting a server connection channel definition

The XA switch structures

Two XA switch structures are supplied with the extended transactional client on each platform.

These switch structures are:

MQRMIXASwitch
: This switch structure is used by a transaction manager when a queue manager, acting as a resource manager, is not using dynamic registration.

MQRMIXASwitchDynamic
: This switch structure is used by a transaction manager when a queue manager, acting as a resource manager, uses dynamic registration.

These switch structures are located in the libraries shown in Table 17 on page 106.

Table 17. WebSphere MQ libraries containing the XA switch structures

Platform	Library containing the XA switch structures
AIX HP-UX Linux Solaris	*MQ_INSTALLATION_PATH*/lib/libmqcxa
Windows systems	*MQ_INSTALLATION_PATH*\bin\mqcxa.dll [1]
MQ_INSTALLATION_PATH represents the high-level directory in which WebSphere MQ is installed.	

The name of the WebSphere MQ resource manager in each switch structure is MQSeries_XA_RMI, but many queue managers can share the same switch structure.

Related concepts

“Dynamic registration and extended transactional processing” on page 106
Using dynamic registration is a form of optimization because it can reduce the number of xa_ function calls issued by the transaction manager.

Dynamic registration and extended transactional processing
Using dynamic registration is a form of optimization because it can reduce the number of xa_ function calls issued by the transaction manager.

If a queue manager does not use dynamic registration, a transaction manager involves the queue manager in every unit of work. The transaction manager does this by calling xa_start, xa_end, and xa_prepare, even if the queue manager has no resources that are updated within the unit of work.

If a queue manager uses dynamic registration, a transaction manager starts by assuming that the queue manager is not involved in a unit of work, and does not call xa_start. The queue manager then becomes involved in the unit of work only if its resources are updated within sync point control. If this occurs, the extended transactional client calls ax_reg to register the queue manager's involvement.

Using the extended transactional client with SSL channels

You cannot set up an SSL channel using the xa_open string. Follow these instructions to use the client channel definition table (ccdt).

About this task

Because of the limited size of the xa_open xa_info string, it is not possible to pass all the information required to set up an SSL channel using the xa_open string method of connecting to a queue manager. Therefore you must either use the client channel definition table or, if your transaction manager allows, create the channel with MQCONNX before issuing the xa_open call.

To use the client channel definition table, follow these steps:

Procedure

1. Specify an xa_open string containing only the mandatory qmname (queue manager name) parameter, for example: `XA_Open_String=qmname=MYQM`
2. Use a queue manager to define a CLNTCONN (client-connection) channel with the required SSL parameters. Include the queue manager name in the QMNAME attribute on the CLNTCONN definition. This will be matched up with the qmname in the xa_open string.
3. Make the CLNTCONN definition available to the client system in a client channel definition table (CCDT) or, on Windows, in the active directory.

4. If you are using a CCDT, identify the CCDT containing the definition of the CLNTCONN channel using environment variables MQCHLLIB and MQCHLTAB. Set these variables in the environments used by both the client application and the transaction manager.

Results

This gives the transaction manager a channel definition to the appropriate queue manager with the SSL attributes needed to authenticate correctly, including SSLCIPH, the CipherSpec.

Configuring an extended transactional client for CICS

You configure an extended transactional client for use by CICS by adding an XAD resource definition to a CICS region.

Add the XAD resource definition by using the CICS resource definition online (RDO) command, **cicsadd**. The XAD resource definition specifies the following information:

- An xa_open string
- The fully qualified path name of a switch load file

One switch load file is supplied for use by CICS on each of the following platforms: AIX, HP-UX, Solaris, and Windows systems. Each switch load file contains a function that returns a pointer to the XA switch structure that is used for dynamic registration, MQRMIXASwitchDynamic. See Table 18 on page 107 for the fully qualified path name of each switch load file.

Table 18. The switch load files

Platform	**Switch load file**
AIX HP-UX Linux Solaris	*MQ_INSTALLATION_PATH*/lib/amqczsc
Windows systems	*MQ_INSTALLATION_PATH*\bin\mqcc4swi.dll [1]
MQ_INSTALLATION_PATH represents the high-level directory in which WebSphere MQ is installed.	

Here is an example of an XAD resource definition for Windows systems:

```
cicsadd -c xad -r REGION1 WMQXA \
    ResourceDescription="WebSphere MQ queue manager MARS" \
    XAOpen="channel=MARS.SVR,trptype=tcp,conname=MARS(1415),qmname=MARS,tpm=cics" \
    SwitchLoadFile="C:\Program Files\IBM\WebSphere MQ\bin\mqcc4swi.dll"
```

For more information about adding an XAD resource definition to a CICS region, see the *CICS Administration Reference* and the *CICS Administration Guide* for your platform.

Note the following information about using CICS with an extended transactional client:

- You can add only one XAD resource definition for WebSphere MQ to a CICS region. This means that only one queue manager can be associated with a region, and all CICS applications that run in the region can connect only to that queue manager. If you want to run CICS applications that connect to a different queue manager, you must run the applications in a different region.
- Each application server in a region calls xa_open while it is initializing and starts an MQI channel to the queue manager associated with the region. This means that the queue manager must be started before an application server starts, otherwise the xa_open call fails. All WebSphere MQ MQI client applications later processed by the application server use the same MQI channel.
- When an MQI channel starts, and there is no security exit at the client end of the channel, the user ID that flows from the client system to the server connection MCA is `cics`. Under certain circumstances, the queue manager uses this user ID for authority checks when the server connection

MCA subsequently attempts to access the queue manager resources on behalf of a client application. If this user ID is used for authority checks, you must ensure that it has the authority to access all the resources it needs to access.

For information about when the queue manager uses this user ID for authority checks, see Security.

- The CICS task termination exits that are supplied for use on WebSphere MQ client systems are listed in Table 19 on page 108. You configure these exits in the same way that you configure the corresponding exits for WebSphere MQ server systems. For this information, therefore, see the Enabling CICS user exits.

Table 19. CICS task termination exits

Platform	Source	Library
AIX HP-UX Linux Solaris	amqzscgx.c	amqczscg
Windows systems	amqzscgn.c	mqcc1415.dll

Configuring an extended transactional client for Tuxedo

To configure XAD resource definition for use by Tuxedo, update the UBBCONFIG file and resource manager table.

To configure XAD resource definition for use by Tuxedo, perform the following actions:

- In the GROUPS section of the Tuxedo UBBCONFIG file for an application, use the OPENINFO parameter to specify an xa_open string.

  For an example of how to do this, see the sample UBBCONFIG file, which is supplied for use with the Tuxedo sample programs. On AIX, HP-UX, and Solaris, the name of the file is ubbstxcx.cfg and, on Windows systems, the name of the file is `ubbstxcn.cfg`.
- In the entry for a queue manager in the Tuxedo resource manager table:
  - udataobj/RM (AIX, HP-UX, and Solaris)
  - udataobj\rm (Windows systems)

  specify the name of an XA switch structure and the fully qualified path name of the library that contains the structure. For an example of how to do this for each platform, see TUXEDO samples. Tuxedo supports dynamic registration of a resource manager, and so you can use either MQRMIXASwitch or MQRMIXASwitchDynamic.

Microsoft Transaction Server

No additional configuration is required before you can use MTS as a transaction manager. However, there are some points to note.

Note the following information about using MTS with the extended transactional client:

- An MTS application always starts an MQI channel when it connects to a server queue manager. MTS, in its role as a transaction manager, then uses the same MQI channel to communicate with the queue manager.
- Following a failure, MTS must be able to recover any incomplete units of work. To do this, MTS must be able to communicate with any queue manager that was participating in an incomplete unit of work at the time of the failure.

  When an MTS application connects to a server queue manager and starts an MQI channel, the extended transactional client extracts sufficient information from the parameters of the MQCONN or MQCONNX call to enable the channel to be restarted following a failure, if required. The extended transactional client passes the information to MTS, and MTS records the information in its log.

If the MTS application issues an MQCONN call, this information is simply the name of the queue manager. If the MTS application issues an MQCONNX call and provides a channel definition structure, MQCD, the information also includes the name of the MQI channel, the network address of the server queue manager, and the communications protocol for the channel.

In a recovery situation, MTS passes this information back to the extended transactional client, and the extended transactional client uses it to restart the MQI channel.

If you ever need to change any configuration information, therefore, ensure that all incomplete units of work have been resolved before making the changes. Alternatively, ensure that the configuration changes do not affect the ability of the extended transactional client to restart an MQI channel using the information recorded by MTS. The following are examples of such configuration changes:

- Changing the value of the MQSERVER environment variable
- Changing entries in the client channel definition table (CCDT)
- Deleting a server connection channel definition

- Note the following conditions when using an extended transactional client with MTS:
  - Within a single thread, a client application can be connected to only one queue manager at a time.
  - Each thread of a client application can connect to a different queue manager.
  - A client application cannot use shared connection handles.

Defining MQI channels

To create a new channel, you have to create **two** channel definitions, one for each end of the connection, using the same channel name and compatible channel types. In this case, the channel types are *server-connection* and *client-connection*.

User defined channels

When the server does not automatically define channels there are two ways of creating the channel definitions and giving the WebSphere MQ application on the WebSphere MQ MQI client machine access to the channel.

These two methods are described in detail:

1. Create one channel definition on the WebSphere MQ client and the other on the server.

   This applies to any combination of WebSphere MQ MQI client and server platforms. Use it when you are getting started on the system, or to test your setup.

   See “Creating server-connection and client-connection definitions on different platforms” on page 110 for details on how to use this method.

2. Create both channel definitions on the server machine.

   Use this method when you are setting up multiple channels and WebSphere MQ MQI client machines at the same time.

   See “Creating server-connection and client-connection definitions on the server” on page 113 for details on how to use this method.

Automatically defined channels

WebSphere MQ products on platforms other than z/OS include a feature that can automatically create a channel definition on the server if one does not exist.

If an inbound attach request is received from a client and an appropriate server-connection definition cannot be found on that queue manager, WebSphere MQ creates a definition automatically and adds it to the queue manager. The automatic definition is based on the definition of the default server-connection channel SYSTEM.AUTO.SVRCONN. You enable automatic definition of server-connection definitions by updating the queue manager object using the ALTER QMGR command with the CHAD parameter (or the PCF command Change Queue Manager with the ChannelAutoDef parameter).

For more information about the automatic creation of channel definitions, see Auto-definition of receiver and server-connection channels.

Related concepts

"Automatically defined channels" on page 110
WebSphere MQ products on platforms other than z/OS include a feature that can automatically create a channel definition on the server if one does not exist.

"User defined channels" on page 110
When the server does not automatically define channels there are two ways of creating the channel definitions and giving the WebSphere MQ application on the WebSphere MQ MQI client machine access to the channel.

"Channel control function" on page 52
The channel control function provides facilities for you to define, monitor, and control channels.

Automatically defined channels

WebSphere MQ products on platforms other than z/OS include a feature that can automatically create a channel definition on the server if one does not exist.

If an inbound attach request is received from a client and an appropriate server-connection definition cannot be found on that queue manager, WebSphere MQ creates a definition automatically and adds it to the queue manager. The automatic definition is based on the definition of the default server-connection channel SYSTEM.AUTO.SVRCONN. You enable automatic definition of server-connection definitions by updating the queue manager object using the ALTER QMGR command with the CHAD parameter (or the PCF command Change Queue Manager with the ChannelAutoDef parameter).

User defined channels

When the server does not automatically define channels there are two ways of creating the channel definitions and giving the WebSphere MQ application on the WebSphere MQ MQI client machine access to the channel.

These two methods are described in detail:

1. Create one channel definition on the WebSphere MQ client and the other on the server.

   This applies to any combination of WebSphere MQ MQI client and server platforms. Use it when you are getting started on the system, or to test your setup.

   See "Creating server-connection and client-connection definitions on different platforms" on page 110 for details on how to use this method.
2. Create both channel definitions on the server machine.

   Use this method when you are setting up multiple channels and WebSphere MQ MQI client machines at the same time.

   See "Creating server-connection and client-connection definitions on the server" on page 113 for details on how to use this method.

Creating server-connection and client-connection definitions on different platforms

You can create each channel definition on the computer to which it applies. There are restrictions on how you can create channel definitions on a client computer.

On all platforms, you can use WebSphere MQ Script (MQSC) commands, programmable command format (PCF) commands, or the IBM WebSphere MQ Explorer to define a server-connection channel on the server machine.

Because MQSC commands are not available on a machine where WebSphere MQ has been installed as a WebSphere MQ MQI client only, you must use different ways of defining a client-connection channel on the client machine.

Related concepts

"Creating a client-connection channel on the IBM WebSphere MQ MQI client" on page 111
You can define a client-connection channel on the client workstation using MQSERVER or using the MQCNO structure on an MQCONNX call.

Related tasks

"Defining a server-connection channel on the server" on page 111
Start MQSC if necessary, then define the server-connection channel.

Defining a server-connection channel on the server

Start MQSC if necessary, then define the server-connection channel.

Procedure

1. Optional: If your server platform is not z/OS, first create and start a queue manager and then start MQSC commands.
   a) Create a queue manager, called QM1 for example:

   ```
   crtmqm QM1
   ```

   b) Start the queue manager:

   ```
   strmqm QM1
   ```

   c) Start MQSC commands:

   ```
   runmqsc QM1
   ```

2. Define a channel with your chosen name and a channel type of *server-connection*.

   ```
   DEFINE CHANNEL(CHAN1) CHLTYPE(SVRCONN) TRPTYPE(TCP) +
   DESCR('Server-connection to Client_1')
   ```

   This channel definition is associated with the queue manager running on the server.
3. Use the following command to allow the inbound connect access to your queue manager:

   ```
   SET CHLAUTH(CHAN1) TYPE(ADDRESSMAP)  ADDRESS('IP address') MCAUSER('userid')
   ```

   - Where SET CHLAUTH uses the name of the channel defined in the previous step.
   - Where *'IP address'* is the IP address of the client.
   - Where *'userid'* is the ID you want to provide to the channel for access control to the target queues. This field is case-sensitive.

   You can choose to identify your inbound connection using a number of different attributes. The example uses IP address. Alternative attributes include client user ID and SSL or TLS Subject Distinguished Name. For more information, see Channel authentication records

Creating a client-connection channel on the IBM WebSphere MQ MQI client

You can define a client-connection channel on the client workstation using MQSERVER or using the MQCNO structure on an MQCONNX call.

Using MQSERVER

You can use the MQSERVER environment variable to specify a simple definition of a client-connection channel. It is simple in the sense that you can specify only a few attributes of the channel using this method.

- Specify a simple channel definition on Windows as follows:

  ```
  SET MQSERVER=ChannelName/TransportType/ConnectionName
  ```

- Specify a simple channel definition on UNIX and Linux systems as follows:

```
export MQSERVER=ChannelName/TransportType/ConnectionName
```

where:

- `ChannelName` must be the same name as defined on the server. It cannot contain a forward slash.
- `TransportType` can be one of the following values, depending on your IBM WebSphere MQ MQI client platform:
  - LU62
  - TCP
  - NETBIOS
  - SPX

  Note: On UNIX and Linux systems, the TransportType is case-sensitive and must be uppercase. An `MQCONN` or `MQCONNX` call returns 2058 if the TransportType is not recognized
- `ConnectionName` is the name of the server as defined to the communications protocol (TransportType).

For example, on Windows:

```
SET MQSERVER=CHANNEL1/TCP/MCID66499
```

or, on UNIX and Linux systems:

```
export MQSERVER=CHANNEL1/TCP/'MCID66499'
```

Note: To change the TCP/IP port number, see “MQSERVER” on page 144.

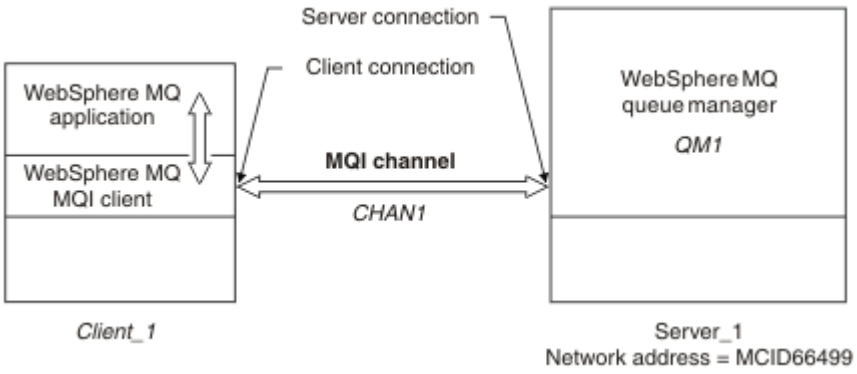


Figure 17. Simple channel definition

Some more examples of simple channel definitions are:

- On Windows:

```
SET MQSERVER=CHANNEL1/TCP/9.20.4.56
SET MQSERVER=CHANNEL1/NETBIOS/BOX643
```

- On UNIX and Linux systems:

```
export MQSERVER=CHANNEL1/TCP/'9.20.4.56'
export MQSERVER=CHANNEL1/LU62/BOX99
```

where B0X99 is the LU 6.2 ConnectionName.

On the IBM WebSphere MQ MQI client, all **MQCONN** or **MQCONNX** requests then attempt to use the channel you have defined, unless the channel is overridden in an MQCD structure referenced from the MQCNO structure supplied to **MQCONNX**.

Note: For more information about the *MQSERVER* environment variable, see "MQSERVER" on page 144.

Using the MQCNO structure on an MQCONNX call

A IBM WebSphere MQ MQI client application can use the connect options structure, MQCNO, on an **MQCONNX** call to reference a channel definition structure, MQCD, that contains the definition of a client-connection channel.

In this way, the client application can specify the **ChannelName**, **TransportType**, and **ConnectionName** attributes of a channel at run time, enabling the client application to connect to multiple server queue managers simultaneously.

Note that if you define a channel using the *MQSERVER* environment variable, it is not possible to specify the **ChannelName**, **TransportType**, and **ConnectionName** attributes at run time.

A client application can also specify attributes of a channel such as **MaxMsgLength** and **SecurityExit**. Specifying such attributes enables the client application to specify values for the attributes that are not the default values, and enables channel exit programs to be called at the client end of an MQI channel.

If a channel uses the Secure Sockets Layer (SSL) or Transport Layer Security (TLS), a client application can also provide information relating to SSL or TLS in the MQCD structure. Additional information relating to SSL or TLS can be provided in the SSL or TLS configuration options structure, MQSCO, which is also referenced by the MQCNO structure on an **MQCONNX** call.

For more information about the MQCNO, MQCD, and MQSCO structures, see MQCNO, MQCD, and MQSCO.

Note: The sample program for MQCONNX is called **amqscnxc**. Another sample program called **amqssslc** demonstrates use of the MQSCO structure.

Creating server-connection and client-connection definitions on the server

You can create both definitions on the server, then make the client-connection definition available to the client.

First define a server-connection channel and then define a client-connection channel. On all platforms, you can use WebSphere MQ Script (MQSC) commands, programmable command format (PCF) commands or the IBM WebSphere MQ Explorer to define a server-connection channel on the server machine.

Client-connection channel definitions created on the server are made available to clients using a client channel definition table (CCDT).

Related concepts

"Client channel definition table" on page 114
The client channel definition table (CCDT) determines the channel definitions and authentication information used by client applications to connect to the queue manager. On platforms other than z/OS a CCDT is created automatically. You must then make it available to the client application.

Related tasks

"Defining the server-connection channel on the server" on page 116
Create a server-connection channel definition for the queue manager.

"Defining the client-connection channel on the server" on page 116
Having defined the server-connection channel, you now define the corresponding client-connection channel.

"Accessing client-connection channel definitions" on page 117

Make the client channel definition table (CCDT) available to client applications by copying or sharing it, then specify its location and name on the client computer.

Client channel definition table

The client channel definition table (CCDT) determines the channel definitions and authentication information used by client applications to connect to the queue manager. On platforms other than z/OS a CCDT is created automatically. You must then make it available to the client application.

The purpose of the client channel definition table (CCDT) is to determine the channel definitions used by client applications to connect to the queue manager. The channel definition also specifies the authentication information that applies to the connections.

The CCDT is a binary file. It is generated by a queue manager. The queue manager does not read the CCDT file.

On platforms other than z/OS, the CCDT is created when the queue manager is created. Client connection channels are added to the table when you use the **DEFINE CHANNEL** command, and their definitions altered when you issue the **ALTER CHANNEL** command.

You can use the CCDT to provide clients with the authentication information to check for SSL certificate revocation. Define a namelist containing authentication information objects and set the queue manager attribute **SSLCRLNameList** to the name of the namelist.

There are a number of ways for a client application to use a CCDT. The CCDT can be copied to the client computer. You can copy the CCDT to a location shared by more than one client. You can make the CCDT accessible to the client as a shared file, while it remains located on the server.

If you use FTP to copy the file, use the `bin` option to set binary mode; do not use the default `ASCII` mode. Whichever method you choose to make the CCDT available, the location must be secure to prevent unauthorized changes to the channels.

Platforms other than z/OS

A default CCDT called `AMQCLCHL.TAB` is created when you create a queue manager.

By default, AMQCLCHL.TAB is located in the following directory on a server:

- UNIX Linux On UNIX and Linux systems:

  ```
  /prefix/qmgrs/QUEUEMANAGERNAME/@ipcc
  ```

  The name of the directory referenced by *QUEUEMANAGERNAME* is case-sensitive on UNIX and Linux systems. The directory name might not be the same as the queue manager name, if the queue manager name has special characters in it.

- Windows On Windows:

  ```
  MQ_INSTALLATION_PATH\data\qmgrs\QUEUEMANAGERNAME\@ipcc
  ```

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

However, you might have chosen to use a different directory for queue manager data. You can specify the parameter **-md** *DataPath* when you used the **crtmqm** command. If you do, `AMQCLCHL.TAB` is located in the `@ipcc` directory of the *DataPath* you specified.

The path to the CCDT can be changed by setting `MQCHLLIB`. If you do set `MQCHLLIB`, be aware, if you have multiple queue managers on the same server, they share the same CCDT location.

The CCDT is created when the queue manager is created. Each entry of a CCDT represents a client connection to a specific queue manager. A new entry is added when you define a client-connection channel using the **DEFINE CHANNEL** command, and the entry is updated when you alter the client-connection channels by using the **ALTER CHANNEL** command.

How to specify the location of the CCDT on the client

On a client system, you can specify the location of the CCDT in two ways:

- Using the environment variables `MQCHLLIB` to specify the directory where the table is located, and `MQCHLTAB` to specify the file name of the table.
- Using the client configuration file. In the `CHANNELS` stanza, use the attributes `ChannelDefinitionDirectory` to specify the directory where the table is located, and `ChannelDefinitionFile` to specify the file name.

If the location is specified both in the client configuration file and by using environment variables, the environment variables take priority. You can use this feature to specify a standard location in the client configuration file and override it using environment variables when necessary.

Related reference

"MQCHLLIB" on page 142
`MQCHLLIB` specifies the directory path to the file containing the client channel definition table (CCDT). The file is created on the server, but can be copied across to the WebSphere MQ MQI client workstation.

Related information

Working with revoked certificates

Migration and client channel definition tables (CCDT)

In general, the internal format of the client channel definition table might change from one release level of IBM WebSphere MQ to the next. As a result, an IBM WebSphere MQ MQI client can use a client channel definition table only when it has been prepared by a server queue manager that is at the same release level as the client, or at an earlier release level.

A Version 7.1 IBM WebSphere MQ MQI client can use a client channel definition table that has been prepared by a Version 6.0 queue manager. But a Version 6.0 client cannot use a client channel definition table that has been prepared by a Version 7.1 queue manager.

Client connection channels in the Active Directory

On Windows systems that support the Active Directory, IBM WebSphere MQ publishes client connection channels in the Active Directory to provide dynamic client-server binding.

When client connection channel objects are defined, they are written into a client channel definition file, called AMQCLCHL.TAB by default. If the client connection channels use the TCP/IP protocol, the IBM WebSphere MQ server also publishes them in the Active Directory. When the IBM WebSphere MQ client determines how to connect to the server, it looks for a relevant client connection channel object definition using the following search order:

1. MQCONNX MQCD data structure
2. MQSERVER environment variable
3. client channel definition file
4. Active Directory

This order means that any current applications are not affected by any change. You can think of these entries in the Active Directory as records in the client channel definition file, and the IBM WebSphere MQ client processes them in the same way. To configure and administer support for publishing client connection channel definitions in the Active Directory, use the `setmqscp` command, as described in setmqscp.

Defining the server-connection channel on the server

Create a server-connection channel definition for the queue manager.

Procedure

1. On the server machine, define a channel with your chosen name and a channel type of *server-connection*.
   For example:

```
DEFINE CHANNEL(CHAN2) CHLTYPE(SVRCONN) TRPTYPE(TCP) +
DESCR('Server-connection to Client_2')
```

2. Use the following command to allow the inbound connect access to your queue manager:

```
SET CHLAUTH(CHAN2) TYPE(ADDRESSMAP)  ADDRESS('IP address') MCAUSER('userid')
```

   - Where SET CHLAUTH uses the name of the channel defined in the previous step.
   - Where *'IP address'* IP address is the IP address of the client.
   - Where *'userid'* is the ID you want to provide to the channel for access control to the target queues. This field is case-sensitive.

   You can choose to identify your inbound connection using a number of different attributes. The example uses IP address. Alternative attributes include client user ID and SSL or TLS Subject Distinguished Name. For more information, see Channel authentication records

   This channel definition is associated with the queue manager running on the server.

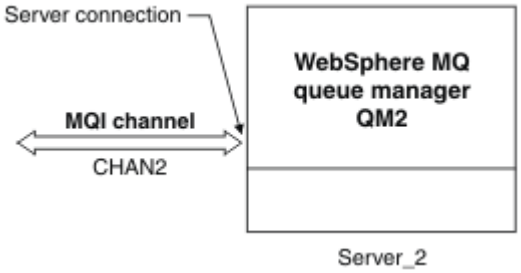


Figure 18. Defining the server-connection channel

Defining the client-connection channel on the server

Having defined the server-connection channel, you now define the corresponding client-connection channel.

Before you begin

Define the server-connection channel.

Procedure

1. Define a channel with the same name as the server-connection channel, but a channel type of *client-connection*. You must state the connection name (CONNAME). For TCP/IP, the connection name is the network address or host name of the server machine. It is also advisable to specify the queue manager name (QMNAME) to which you want your IBM WebSphere MQ application, running in the client environment, to connect. By varying the queue manager name, you can define a set of channels to connect to different queue managers.

```
DEFINE CHANNEL(CHAN2) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
CONNAME(9.20.4.26) QMNAME(QM2) DESCR('Client-connection to Server_2')
```

2. Use the following command to allow the inbound connect access to your queue manager:

```
SET CHLAUTH(CHAN2) TYPE(ADDRESSMAP)  ADDRESS('IP-address') MCAUSER('userid')
```

- Where SET CHLAUTH uses the name of the channel defined in the previous step.
- Where *'IP address'* is the IP address of the client.
- Where *'userid'* is the ID you want to provide to the channel for access control to the target queues. This field is case-sensitive.

You can choose to identify your inbound connection using a number of different attributes. The example uses IP address. Alternative attributes include client user ID and SSL or TLS Subject Distinguished Name. For more information, see Channel authentication records

Results

On platforms other than z/OS, this channel definition is stored in a file called the client channel definition table (CCDT), which is associated with the queue manager. The client channel definition table can contain more than one client-connection channel definition. For more information about the client channel definition table, and for the corresponding information about how client-connection channel definitions are stored on z/OS, see "Client channel definition table" on page 114.

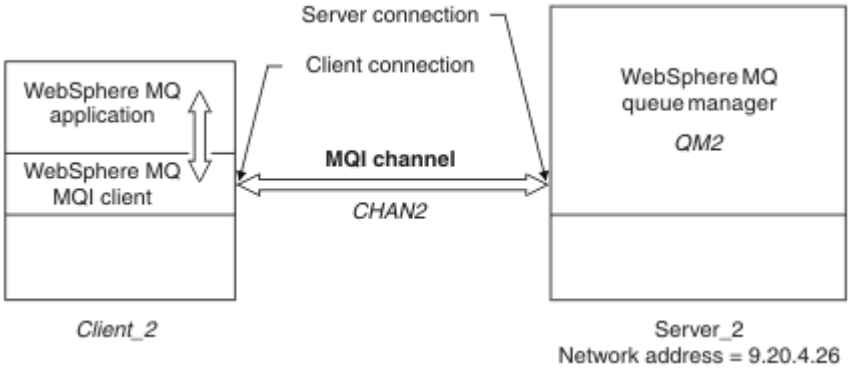


Figure 19. Defining the client-connection channel

Accessing client-connection channel definitions

Make the client channel definition table (CCDT) available to client applications by copying or sharing it, then specify its location and name on the client computer.

Before you begin

You have defined the client-connection channels you need.

On z/OS, you have created a CCDT. On other platforms, the CCDT is automatically created and updated.

About this task

For a client application to use the client channel definition table (CCDT), you must make the CCDT available to it and specify its location and name

Procedure

1. Make the CCDT available to the client applications in one of three ways:
   a) Optional: Copy the CCDT to the client computer.
   b) Optional: Copy the CCDT to a location shared by more than one client.
   c) Optional: Leave the CCDT on the server but make it shareable by the client.

   Whichever location you choose for the CCDT, the location must be secure to prevent unauthorized changes to the channels.

2. On the client, specify the location and name of the file containing the CCDT in one of three ways:
   a) Optional: Use the CHANNELS stanza of the client configuration file. For more information, see "CHANNELS stanza of the client configuration file" on page 132.
   b) Optional: Use the environment variables MQCHLLIB and MQCHLTAB.

      For example, you can set the environment variables by typing:

      - On HP Integrity NonStop Server, and UNIX and Linux systems:

        ```
        export MQCHLLIB=MQ_INSTALLATION_PATH/qmgrs/QUEUEMANAGERNAME/@ipcc
        export MQCHLTAB=AMQCLCHL.TAB
        ```

      where *MQ_INSTALLATION_PATH* represents the high-level directory in which WebSphere MQ is installed.
   c) Optional: On Windows only, use the **setmqscp** control command to publish the client-connection channel definitions in Active Directory

   If the MQSERVER environment variable is set, a WebSphere MQ client uses the client-connection channel definition specified by MQSERVER in preference to any definitions in the client channel definition table.

Channel-exit programs for MQI channels

Three types of channel exit are available to the WebSphere MQ MQI client environment on UNIX, Linux and Windows systems.

These are:

- Send exit
- Receive exit
- Security exit

These exits are available at both the client and the server end of the channel. Exits are not available to your application if you are using the MQSERVER environment variable. Channel exits are explained in Channel exit programs for messaging channels.

The send and receive exits work together. There are several possible ways in which you can use them:

- Splitting and reassembling a message
- Compressing and decompressing data in a message (this functionality is provided as part of WebSphere MQ, but you might want to use a different compression technique)
- Encrypting and decrypting user data (this functionality is provided as part of WebSphere MQ, but you might want to use a different encryption technique)
- Journaling each message sent and received

You can use the security exit to ensure that the WebSphere MQ client and server are correctly identified, and to control access.

If send or receive exits on the server-connection side of the channel instance need to perform MQI calls on the connection with which they are associated, they use the connection handle provided in the MQCXP Hconn field. You must be aware that client-connection send and receive exits cannot make MQI calls.

Related concepts

"Security exits on a client connection" on page 119
You can use security exit programs to verify that the partner at the other end of a channel is genuine. Special considerations apply when a security exit is applied to a client connection.

User exits, API exits, and WebSphere MQ installable services

Related tasks

Extending queue manager facilities

Related reference

"Path to exits" on page 119
A default path for location of the channel exits is defined in the client configuration file. Channel exits are loaded when a channel is initialized.

"Identifying the API call in a send or receive exit program" on page 120
When you use MQI channels for clients, byte 10 of the agent buffer identifies the API call in use when a send or receive exit is called. This is useful for identifying which channel flows include user data and might require processing such as encryption or digital signing.

Path to exits

A default path for location of the channel exits is defined in the client configuration file. Channel exits are loaded when a channel is initialized.

On UNIX, Linux and Windows systems, a client configuration file is added to your system during installation of the WebSphere MQ MQI client. A default path for location of the channel exits on the client is defined in this file, using the stanza:

```
ClientExitPath:
  ExitsDefaultPath=string
  ExitsDefaultPath64=string
```

where *string* is a file location in a format appropriate to the platform

When a channel is initialized, after an MQCONN or MQCONNX call, the client configuration file is searched. The ClientExitPath stanza is read and any channel exits that are specified in the channel definition are loaded.

Security exits on a client connection

You can use security exit programs to verify that the partner at the other end of a channel is genuine. Special considerations apply when a security exit is applied to a client connection.

Figure 20 on page 120 illustrates the use of security exits in a client connection, using the WebSphere MQ object authority manager to authenticate a user. Either SecurityParmsPtr or SecurityParmsOffset is set in the MQCNO structure on the client and there are security exits at both ends of the channel. After the normal security message exchange has ended, and the channel is ready to run, the MQCSP structure accessed from the MQCXP SecurityParms field is passed to the security exit on the client. The exit type is set to MQXR_SEC_PARMS. The security exit can elect to do nothing to the user identifier and password, or it can alter either or both of them. The data returned from the exit is then sent to the server-connection end of the channel. The MQCSP structure is rebuilt on the server-connection end of the channel and is passed to the server-connection security exit accessed from the MQCXP SecurityParms field. The security exit receives and processes this data. This processing is typically to reverse any change made to the user ID and password fields in the client exit, which are then used to authorize the queue manager connection. The resulting MQCSP structure is referenced using SecurityParmsPtr in the MQCNO structure on the queue manager system.

If SecurityParmsPtr or SecurityParmsOffset are set in the MQCNO structure and there is a security exit at only one end of the channel, the security exit receives and processes the MQCSP structure. Actions such as encryption are inappropriate for a single user exit, as there is no exit to perform the complementary action.

If SecurityParmsPtr and SecurityParmsOffset are not set in the MQCNO structure and there is a security exit at either or both ends of the channel, the security exit or exits are called. Either security exit can return its own MQCSP structure, addressed through the SecurityParmsPtr; the security exit is not called again until it is terminated (ExitReason of MQXR_TERM). The exit writer can free the memory used for the MQCSP at that stage.

When a server-connection channel instance is sharing more than one conversation, the pattern of calls to the security exit is restricted on the second and subsequent conversations.

For the first conversation, the pattern is the same as if the channel instance is not sharing conversations. For the second and subsequent conversations, the security exit is never called with MQXR_INIT, MQXR_INIT_SEC, or MQXR_SEC_MSG. It is called with MQXR_SEC_PARMS.

In a channel instance with sharing conversations, MQXR_TERM is called only for the last conversation running.

Each conversation has the opportunity in the MQXR_SEC_PARMS invocation of the exit to alter the MQCD; on the server-connection end of the channel this feature can be useful to vary, for example, the MCAUserIdentifier or LongMCAUserIdPtr values before the connection is made to the queue manager.

Server-connection exit	Client-connection exit
	Invoked with MQXR_INIT Responds with MQXCC_OK
Invoked with MQXR_INIT Responds with MQXCC_OK	
	Invoked with MQXR_INIT_SEC Responds with MQXCC_OK
Invoked with MQXR_INIT_SEC Responds with MQXCC_OK	
	Invoked with MQXR_SEC_PARMS Responds with MQXCC_OK
Invoked with MQXR_SEC_PARMS Responds with MQXCC_OK	
Data transfer begins	
Invoked with MQXR_TERM Responds with MQXCC_OK	Invoked with MQXR_TERM Responds with MQXCC_OK

Figure 20. Client connection-initiated exchange with agreement for client connection using security parameters

Note: Security exit applications constructed prior to the release of WebSphere MQ v7.1 may require updating. For more information see Channel security exit programs.

Identifying the API call in a send or receive exit program

When you use MQI channels for clients, byte 10 of the agent buffer identifies the API call in use when a send or receive exit is called. This is useful for identifying which channel flows include user data and might require processing such as encryption or digital signing.

The following table shows the data that appears in byte 10 of the channel flow when an API call is being processed.

Note: These are not the only values of this byte. There are other **reserved** values.

Table 20. Identifying API calls

API call	Value of byte 10 for request	Value of byte 10 for reply
MQCONN "1" on page 121,"2" on page 121	X'81'	X'91'
MQDISC "1" on page 121	X'82'	X'92'
MQOPEN "3" on page 121	X'83'	X'93'
MQCLOSE	X'84'	X'94'
MQGET "4" on page 121	X'85'	X'95'
MQPUT "4" on page 121	X'86'	X'96'
MQPUT1 request "4" on page 121	X'87'	X'97'
MQSET request	X'88'	X'98'
MQINQ request	X'89'	X'99'
MQCMIT request	X'8A'	X'9A'
MQBACK request	X'8B'	X'9B'
MQSTAT request	X'8D'	X'9D'
MQSUB request	X'8E'	X'9E'
MQSUBRQ request	X'8F'	X'9F'
xa_start request	X'A1'	X'B1'
xa_end request	X'A2'	X'B2'
xa_open request	X'A3'	X'B3'
xa_close request	X'A4'	X'B4'
xa_prepare request	X'A5'	X'B5'
xa_commit request	X'A6'	X'B6'
xa_rollback request	X'A7'	X'B7'
xa_forget request	X'A8'	X'B8'
xa_recover request	X'A9'	X'B9'
xa_complete request	X'AA'	X'BA'

Notes:

1. The connection between the client and server is initiated by the client application using MQCONN. Therefore, for this command in particular, there are several other network flows. The same applies to MQDISC, which terminates the network connection.
2. MQCONNX is treated in the same way as MQCONN for the purposes of the client-server connection.
3. If a large distribution list is opened, there might be more than one network flow per MQOPEN call in order to pass all the required data to the SVRCONN MCA.
4. Large messages can exceed the transmission segment size. If this happens there can be many network flows resulting from a single API call.

Connecting a client to a queue-sharing group

You can connect a client to a queue-sharing group by creating an MQI channel between a client and a queue manager on a server that is a member of a queue-sharing group.

A queue-sharing group is formed by a set of queue-managers that can access the same set of shared queues.

A client putting to a shared queue can connect to any member of the queue-sharing group. The benefits of connecting to a queue-sharing group are possible increases in front-end and back-end availability, and increased capacity. You can connect to a specific queue manager or to the generic interface.

Connecting directly to a queue manager in a queue-sharing group gives the benefit that you can put messages to a shared target queue, which increases back-end availability.

Connecting to the generic interface of a queue-sharing group opens a session with one of the queue managers in the group. This increases front-end availability, because the client queue manager can connect with any queue-manager in the group. You connect to the group using the generic interface when you do not want to connect to a specific queue manager within the queue-sharing group.

The generic interface can be a WLM/DNS group name or a VTAM® generic resource name, or another common interface to the queue-sharing group.

To connect to the generic interface of a queue-sharing group you need to create channel definitions that can be accessed by any queue manager in the group. To do this you need to have the same definitions on each queue manager in the group.

Define the SVRCONN channel as follows:

```
DEFINE CHANNEL(CHANNEL1) CHLTYPE(SVRCONN) TRPTYPE(TCP) +
MCAUSER(' ') QSGDISP(GROUP)
```

Channel definitions on the server are stored in a shared DB2® repository. Each queue manager in the queue-sharing group makes a local copy of the definition, ensuring that you will always connect to the correct server-connection channel when you issue an MQCONN or MQCONNX call.

Define the CLNTCONN channel as follows:

```
DEFINE CHANNEL(CHANNEL1) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
CONNAME(WLM/DNS groupname) QMNAME(QSG1) +
DESCR('Client-connection to Queue Sharing Group QSG1') QSGDISP(GROUP)
```

Because the generic interface of the queue-sharing group is stored in the CONNAME field in the client-connection channel, you can now connect to any queue manager in the group, and put to shared queues owned by that group.

Related concepts

“Creating channel definitions” on page 122

To connect to the generic interface of a queue-sharing group you need to create channel definitions that can be accessed by any queue manager in the group. To do this you need to have the same definitions on each queue manager in the group.

Creating channel definitions

To connect to the generic interface of a queue-sharing group you need to create channel definitions that can be accessed by any queue manager in the group. To do this you need to have the same definitions on each queue manager in the group.

Define the SVRCONN channel as follows:

```
DEFINE CHANNEL(CHANNEL1) CHLTYPE(SVRCONN) TRPTYPE(TCP) +
MCAUSER(' ') QSGDISP(GROUP)
```

Channel definitions on the server are stored in a shared DB2 repository. Each queue manager in the queue-sharing group makes a local copy of the definition, ensuring that you will always connect to the correct server-connection channel when you issue an MQCONN or MQCONNX call.

Define the CLNTCONN channel as follows:

```
DEFINE CHANNEL(CHANNEL1) CHLTYPE(CLNTCONN) TRPTYPE(TCP) +
CONNAME(WLM/DNS groupname) QMNAME(QSG1) +
DESCR('Client-connection to Queue Sharing Group QSG1') QSGDISP(GROUP)
```

Because the generic interface of the queue-sharing group is stored in the CONNAME field in the client-connection channel, you can now connect to any queue manager in the group, and put to shared queues owned by that group.

Configuring a client using a configuration file

Configure your clients using attributes in a text file. These attributes can be overridden by environment variables or in other platform-specific ways.

You configure your IBM WebSphere MQ MQI client using a text file, similar to the queue manager configuration file, qm.ini, used on UNIX and Linux platforms. The file contains a number of stanzas, each of which contains a number of lines of the format **attribute-name**=*value* .

In this documentation, this file is referred to as the *WebSphere MQ MQI client configuration file*; its file name is generally mqclient.ini, but you can choose to give it another name. Configuration information in this file applies to all platforms, and to clients using the MQI, IBM WebSphere MQ classes for Java, IBM WebSphere MQ classes for JMS, IBM WebSphere MQ classes for .NET, and XMS.

Although the attributes in the IBM WebSphere MQ MQI client configuration file apply to most IBM WebSphere MQ clients, there are some attributes that are not read by managed .NET and XMS .NET clients, or by clients that use either the IBM WebSphere MQ classes for Java or the IBM WebSphere MQ classes for JMS. For more information, see “Which IBM WebSphere MQ clients can read each attribute” on page 125.

The configuration features apply to all connections a client application makes to any queue managers, rather than being specific to an individual connection to a queue manager. Attributes relating to a connection to an individual queue manager can be configured programmatically, for example by using an MQCD structure, or by using a Client Channel Definition Table (CCDT).

Environment variables which were supported in releases of IBM WebSphere MQ earlier than Version 7.0 continue to be supported, and where such an environment variable matches an equivalent value in the client configuration file, the environment variable overrides the client configuration file value.

For a client application using IBM WebSphere MQ classes for JMS, you can also override the client configuration file in the following ways:

- setting properties in the JMS configuration file
- setting Java system properties, which also overrides the JMS configuration file

For the .NET client, you can also override the client configuration file and the equivalent environment variables using the .NET application configuration file.

Note that you cannot set up multiple channel connections using the client configuration file.

Example client configuration file

```
#* Module Name: mqclient.ini                                          *#
#* Type       : WebSphere MQ MQI client configuration file             *#
#  Function   : Define the configuration of a client                   *#
#*                                                                     *#
#*********************************************************************#
#* Notes      :                                                        *#
#* 1) This file defines the configuration of a client                  *#
#*                                                                     *#
```

```
#*******************************************************************#

ClientExitPath:
   ExitsDefaultPath=/var/mqm/exits
   ExitsDefaultPath64=/var/mqm/exits64

TCP:
   Library1=DLLName1
   KeepAlive = Yes
   ClntSndBuffSize=32768
   ClntRcvBuffSize=32768
   Connect_Timeout=0

MessageBuffer:
   MaximumSize=-1
   Updatepercentage=-1
   PurgeTime=0

LU62:
   TPName
   Library1=DLLName1
   Library2=DLLName2

PreConnect:
    Module=amqldapi
    Function=myFunc
    Data=ldap://myLDAPServer.com:389/cn=wmq,ou=ibm,ou=com
    Sequence=1

CHANNELS:
 DefRecon=YES
 ServerConnectionParms=SALES.SVRCONN/TCP/hostname.x.com(1414)
```

Related reference

“Location of the client configuration file” on page 124
An IBM WebSphere MQ MQI client configuration file can be held in a number of locations.

“CHANNELS stanza of the client configuration file” on page 132
Use the CHANNELS stanza to specify information about client channels.

“ClientExitPath stanza of the client configuration file” on page 134
Use the ClientExitPath stanza to specify the default locations of channel exits on the client.

“LU62, NETBIOS, and SPX stanzas of the client configuration file” on page 134
On Windows systems only, use these stanzas to specify configuration parameters for the specified network protocols.

“MessageBuffer stanza of the client configuration file” on page 135
Use the MessageBuffer stanza to specify information about message buffers.

“SSL stanza of the client configuration file” on page 137
Use the SSL stanza to specify information about the use of SSL or TLS.

“TCP stanza of the client configuration file” on page 139
Use the TCP stanza to specify TCP network protocol configuration parameters.

“Using WebSphere MQ environment variables” on page 140
This section describes the environment variables that you can use with WebSphere MQ MQI client applications.

“Changing queue manager configuration information” on page 419
The attributes described here modify the configuration of an individual queue manager. They override any settings for WebSphere MQ.

Location of the client configuration file

An IBM WebSphere MQ MQI client configuration file can be held in a number of locations.

A client application uses the following search path to locate the IBM WebSphere MQ MQI client configuration file:

1. The location specified by the environment variable MQCLNTCF.

The format of this environment variable is a full URL. This means the file name might not necessarily be `mqclient.ini` and facilitates placing the file on a network attached file-system.

Note the following:

- C, .NET and XMS clients support only the `file:` protocol; the `file:` protocol is assumed if the URL string does not begin with `protocol:`
- To allow for Java 1.4.2 JREs, which do not support reading environment variables, the MQCLNTCF environment variable can be overridden with an MQCLNTCF Java System Property.

2. A file called `mqclient.ini` in the present working directory of the application.
3. A file called `mqclient.ini` in the IBM WebSphere MQ data directory for Windows, UNIX and Linux systems.

   Note the following:

   - The IBM WebSphere MQ data directory does not exist on certain platforms, for example, IBM i and z/OS, or in cases where the client has been supplied with another product.
   - On UNIX and Linux systems, the directory is `/var/mqm`
   - On Windows platforms you configure the environment variable MQ_FILE_PATH during installation, to point at the data directory. It is normally `C:\Program Files\IBM\WebSphere MQ`
   - To allow for Java 1.4.2 JREs that do not support reading environment variables you can manually override the MQ_FILE_PATH environment variable with an MQ_FILE_PATH Java System Property.
4. A file called `mqclient.ini` in a standard directory appropriate to the platform, and accessible to users:

   - For all Java clients this is the value of the `user.home` Java System Property.
   - For C clients on UNIX and Linux platforms this is the value of the HOME environment variable.
   - For C clients on Windows this is the concatenated values of the HOMEDRIVE and HOMEPATH environment variables.

Note: For the IBM WebSphere MQ client for HP Integrity NonStop Server, the `mqclient.ini` file must be located in the OSS file system. Guardian applications must either place the `mqclient.ini` file in the IBM WebSphere MQ data directory or set the MQCLNTCF environment variable to a location in the OSS file system.

Which IBM WebSphere MQ clients can read each attribute

Most of the attributes in the IBM WebSphere MQ MQI client configuration file can be used by the C client, and the unmanaged .NET clients. However, there are some attributes that are not read by managed .NET and XMS .NET clients, or by clients using either the IBM WebSphere MQ classes for Java or the IBM WebSphere MQ classes for JMS.

Table 21. Which attributes apply to each type of client

`mqclient.ini` stanza name and attributes	**Description**	**C and unmanaged .NET**	**Java**	**JMS**	**Managed .NET**	**Managed XMS .NET**
CHANNELS stanza						
CCSID	The coded character set number to be used.	Yes	No	No	Yes	Yes

Table 21. Which attributes apply to each type of client (continued)

mqclient.ini stanza name and attributes	**Description**	**C and unmanaged .NET**	**Java**	**JMS**	**Managed .NET**	**Managed XMS .NET**
ChannelDefinitionDirectory	The directory path to the file containing the client channel definition table.	Yes	No	No	Yes	Yes
ChannelDefinitionFile	The name of the file containing the client channel definition table.	Yes	No	No	Yes	Yes
ReconDelay	An administrative option to configure reconnect delay for client programs that can auto-reconnect.	Yes	No	Yes	Yes	Yes
DefRecon	An administrative option to enable client programs to automatically reconnect, or to disable the automatic reconnection of a client program that has been written to reconnect automatically.	Yes	No	Yes	Yes	Yes
MQReconnectTimeout	The timeout in seconds to reconnect to a client.	Yes	No	No	Yes	No

Table 21. Which attributes apply to each type of client (continued)

mqclient.ini stanza name and attributes	**Description**	**C and unmanaged .NET**	**Java**	**JMS**	**Managed .NET**	**Managed XMS .NET**
ServerConnectionParms	The location of the IBM WebSphere MQ server and the communication method to be used.	Yes	No	No	Yes	Yes
Put1DefaultAlwaysSync	Controls the behavior of the MQPUT1 function call with the option MQPMO_RESPONSE_AS_Q_DEF.	Yes	Yes	Yes	Yes	Yes
ClientExitPath stanza						
ExitsDefaultPath	Specifies the location of 32-bit channel exits for clients.	Yes	No	No	Yes	Yes
ExitsDefaultPath64	Specifies the location of 64-bit channel exits for clients.	Yes	No	No	Yes	Yes
JavaExitsClassPath	The values to be added to the classpath when a Java exit is run.	No	Yes	Yes	No	No
MessageBuffer stanza						
MaximumSize	Size, in kilobytes, of the read-ahead buffer, in the range 1 through 999 999.	Yes	Yes	Yes	Yes	Yes

Table 21. Which attributes apply to each type of client (continued)

mqclient.ini stanza name and attributes	**Description**	**C and unmanaged .NET**	**Java**	**JMS**	**Managed .NET**	**Managed XMS .NET**
PurgeTime	Interval, in seconds, after which messages left in the read-ahead buffer are purged.	Yes	Yes	Yes	Yes	Yes
UpdatePercentage	The update percentage value, in the range of 1 - 100, used in calculating the threshold value to determine when a client application makes a new request to the server.	Yes	Yes	Yes	Yes	Yes
SSL stanza						
CDPCheckExtensions	Specifies whether SSL or TLS channels on this queue manager try to check CDP servers that are named in CrlDistributionPoint certificate extensions.	Yes	No	No	No	No
CertificateLabel	The certificate label of the channel definition.	Yes	No	No	No	No
CertificateValPolicy	Determines the type of certificate validation used.	Yes	No	No	No	No

Table 21. Which attributes apply to each type of client (continued)

mqclient.ini stanza name and attributes	**Description**	**C and unmanaged .NET**	**Java**	**JMS**	**Managed .NET**	**Managed XMS .NET**
ClientRevocationChecks	Determines how certificate revocation checking is configured if the client connect call uses an SSL/TLS channel.	Yes	No	No	No	No
EncryptionPolicySuiteB	Determines whether a channel uses Suite-B compliant cryptography and what level of strength is to be used.	Yes	No	No	No	No
OCSPAuthentication	Defines the behavior of IBM WebSphere MQ when OCSP is enabled and the OCSP revocation check is unable to determine the certificate revocation status.	Yes	No	No	No	No
OCSPCheckExtensions	Controls whether IBM WebSphere MQ acts on AuthorityInfoAccess certificate extensions.	Yes	No	No	No	No

Table 21. Which attributes apply to each type of client (continued)

mqclient.ini stanza name and attributes	**Description**	**C and unmanaged .NET**	**Java**	**JMS**	**Managed .NET**	**Managed XMS .NET**
SSLCryptoHardware	Sets the parameter string required to configure PKCS #11 cryptographic hardware present on the system.	Yes	No	No	No	No
SSLFipsRequired	Specifies whether only FIPS-certified algorithms are to be used if cryptography is carried out in IBM WebSphere MQ.	Yes	No	No	No	No
SSLHTTPProxyName	The string is either the host name or network address of the HTTP Proxy server that is to be used by GSKit for OCSP checks.	Yes	No	No	No	No
SSLKeyRepository	The location of the key repository that holds the user's digital certificate, in stem format.	Yes	No	No	No	No

Table 21. Which attributes apply to each type of client (continued)

mqclient.ini stanza name and attributes	**Description**	**C and unmanaged .NET**	**Java**	**JMS**	**Managed .NET**	**Managed XMS .NET**
SSLKeyResetCount	The number of unencrypted bytes sent and received on an SSL or TLS channel before the secret key is renegotiated.	Yes	No	No	No	No
TCP stanza						
ClntRcvBuffSize	The size in bytes of the TCP/IP receive buffer used by the client end of a client-connection server-connection channel.	Yes	Yes	Yes	Yes	Yes
ClntSndBuffSize	The size in bytes of the TCP/IP send buffer used by the client end of a client-connection server-connection channel.	Yes	Yes	Yes	Yes	Yes
Connect_Timeout	The number of seconds before an attempt to connect the socket times out.	Yes	Yes	Yes	No	No
IPAddressVersion	Specifies which IP protocol to use for a channel connection.	Yes	No	No	Yes	Yes

Table 21. Which attributes apply to each type of client (continued)						
mqclient.ini stanza name and attributes	**Description**	**C and unmanaged .NET**	**Java**	**JMS**	**Managed .NET**	**Managed XMS .NET**
KeepAlive	Switches the KeepAlive function on or off.	Yes	Yes	Yes	Yes	Yes
Windows Library1	On Windows only, the name of the TCP/IP sockets DLL.	Yes	No	No	No	No

For the HP Integrity NonStop Server, you can use the TMF and TmfGateway stanzas to communicate with the TMF/Gateway.

CHANNELS stanza of the client configuration file

Use the CHANNELS stanza to specify information about client channels.

The following attributes can be included in the CHANNELS stanza:

CCSID=*number*
The coded character set number to be used.

The CCSID number is equivalent to the MQCCSID environment parameter.

ChannelDefinitionDirectory=*path*
The directory path to the file containing the client channel definition table.

On Windows systems, the default is the IBM WebSphere MQ installation directory, typically `C:\Program Files\IBM\WebSphere MQ`. On UNIX and Linux systems, the default is `/var/mqm`.

The ChannelDefinitionDirectory path is equivalent to the MQCHLLIB environment parameter.

ChannelDefinitionFile=*filename*| AMQCLCHL.TAB
The name of the file containing the client channel definition table.

The client channel definition table is equivalent to the MQCHLTAB environment parameter.

ReconDelay=(delay[,rand])(delay[,rand])...
The ReconDelay attribute provides an administrative option to configure reconnect delay for client programs that can auto-reconnect. Here is an example configuration:

```
ReconDelay=(1000,200)(2000,200)(4000,1000)
```

The example shown defines an initial delay of one second, plus a random interval of up to 200 milliseconds. The next delay is two seconds plus a random interval of up to 200 milliseconds. All subsequent delays are four seconds, plus a random interval of up to 1000 milliseconds.

DefRecon=NO|YES|QMGR|DISABLED
The DefRecon attribute provides an administrative option to enable client programs to automatically reconnect, or to disable the automatic reconnection of a client program that has been written to reconnect automatically. You might opt to set the latter if a program uses an option, such as `MQPMO_LOGICAL_ORDER`, that is incompatible with reconnection.

The interpretation of the DefRecon options depends on whether an `MQCNO_RECONNECT_*` value is also set in the client program, and what value is set.

If the client program connects using MQCONN, or sets the MQCNO_RECONNECT_AS_DEF option using MQCONNX, the reconnect value set by DefRecon takes effect. If no reconnect value is set in the program, or by the DefRecon option, the client program is not reconnected automatically.

Automatic client reconnection is not supported by IBM WebSphere MQ classes for Java.

NO
Unless overridden by MQCONNX, the client is not reconnected automatically.

YES
Unless overridden by MQCONNX, the client reconnects automatically.

QMGR
Unless overridden by MQCONNX, the client reconnects automatically, but only to the same queue manager. The QMGR option has the same effect as MQCNO_RECONNECT_Q_MGR.

DISABLED
Reconnection is disabled, even if requested by the client program using the MQCONNX MQI call.

The automatic client reconnection depends on two values:

- The reconnect option set in the application
- DefRecon value in the `mqclient.ini` file

Table 22. Automatic reconnection depends on the values set in the application and in the mqclient.ini file

DefRecon value in the `mqclient.ini`	**Reconnection options set in the application**			
	MQCNO_RECONNECT	MQCNO_RECONNECT_Q_MGR	MQCNO_RECONNECT_AS_DEF	MQCNO_RECONNECT_DISABLED
NO	YES	QMGR	NO	NO
YES	YES	QMGR	YES	NO
QMGR	YES	QMGR	QMGR	NO
DISABLED	NO	NO	NO	NO

MQReconnectTimeout
The timeout in seconds to reconnect to a client. The default value is 1800 seconds (30 minutes).

IBM WebSphere MQ classes for XMS .NET clients can specify a timeout to reconnect using the property XMSC.WMQ_CLIENT_RECONNECT_TIMEOUT. The default value for this property is 1800 seconds (30 minutes).

ServerConnectionParms
ServerConnectionParms is equivalent to the MQSERVER environment parameter and specifies the location of the IBM WebSphere MQ server and the communication method to be used. The ServerConnectionParms attribute defines only a simple channel; you cannot use it to define an SSL channel or a channel with channel exits. It is a string of the format *ChannelName*/*TransportType*/*ConnectionName*, *ConnectionName* must be a fully qualified network name. *ChannelName* cannot contain the forward slash ("/") character because this character is used to separate the channel name, transport type, and connection name.

When ServerConnectionParms is used to define a client channel, a maximum message length of 100 MB is used. Therefore the maximum message size in effect for the channel is the value specified in the SVRCONN channel on the server.

Note that only a single client channel connection can be made. For example, if you have two entries:

```
ServerConnectionParms=R1.SVRCONN/TCP/localhost(1963)
ServerConnectionParms=R2.SVRCONN/TCP/localhost(1863)
```

only the second one is used.

Specify *ConnectionName* as a comma-separated list of names for the stated transport type. Generally, only one name is required. You can provide multiple *hostnames* to configure multiple connections with the same properties. The connections are tried in the order that they are specified in the connection list until a connection is successfully established. If no connection is successful, the client starts to process again. Connection lists are an alternative to queue manager groups to configure connections for reconnectable clients.

Put1DefaultAlwaysSync=NO|YES
Controls the behavior of the MQPUT1 function call with the option MQPMO_RESPONSE_AS_Q_DEF.

NO
If MQPUT1 is set with MQPMO_SYNCPOINT, it behaves as MQPMO_ASYNC_RESPONSE. Similarly, if MQPUT1 is set with MQPMO_NO_SYNCPOINT, it behaves as MQPMO_SYNC_RESPONSE. This is the default value.

YES
MQPUT1 behaves as if MQPMO_SYNC_RESPONSE is set, regardless of whether MQPMO_SYNCPOINT or MQPMO_NO_SYNCPOINT is set.

ClientExitPath stanza of the client configuration file

Use the ClientExitPath stanza to specify the default locations of channel exits on the client.

The following attributes can be included in the ClientExitPath stanza:

ExitsDefaultPath=*string*
Specifies the location of 32-bit channel exits for clients.

ExitsDefaultPath64=*string*
Specifies the location of 64-bit channel exits for clients.

JavaExitsClassPath=*string*
The values to be added to the classpath when a Java exit is run. This is ignored by exits in any other language.

In the JMS configuration file, the JavaExitsClassPath name is given the standard com.ibm.mq.cfg. prefix and this full name is also used on the Websphere MQ V7.0 system property. At Version 6.0 this attribute was specified using system property com.ibm.mq.exitClasspath, which was documented in the Version 6.0 readme. The use of com.ibm.mq.exitClasspath is deprecated. If both JavaExitsClassPath and exitClasspath are present, JavaExitsClassPath is honored. If only exitClasspath usage is present, it is still honored in Websphere MQ V7.0.

LU62, NETBIOS, and SPX stanzas of the client configuration file

On Windows systems only, use these stanzas to specify configuration parameters for the specified network protocols.

LU62

Use the LU62 stanza to specify SNA LU 6.2 protocol configuration parameters. The following attributes can be included in this stanza:

Library1=*DLLName*|WCPIC32
The name of the APPC DLL.

Library2=*DLLName*|WCPIC32
The same as Library1, used if the code is stored in two separate libraries. .

TPName
The TP name to start on the remote site.

NETBIOS

Use the NETBIOS stanza to specify NetBIOS protocol configuration parameters. The following attributes can be included in this stanza:

AdapterNum=*number*|0
The number of the LAN adapter.

Library1=*DLLName*|NETAPI32
The name of the NetBIOS DLL.

LocalName=*name*
The name by which this computer is known on the LAN.

This is equivalent to the MQNAME environment parameter.

NumCmds=*number*|1
How many commands to allocate.

NumSess=*number*|1
How many sessions to allocate.

SPX

Use the SPX stanza to specify SPX protocol configuration parameters. The following attributes can be included in this stanza:

BoardNum=*number*|0
The LAN adapter number.

KeepAlive=YES|NO
Switch the KeepAlive function on or off.

KeepAlive=YES causes SPX to check periodically that the other end of the connection is still available. If it is not, the channel is closed.

Library1=*DLLName*|WSOCK32.DLL
The name of the SPX DLL.

Library2=*DLLName*|WSOCK32.DLL
The same as Library1, used if the code is stored in two separate libraries.

Socket=*number*|5E86
The SPX socket number in hexadecimal notation.

MessageBuffer stanza of the client configuration file

Use the MessageBuffer stanza to specify information about message buffers.

The following attributes can be included in the MessageBuffer stanza:

MaximumSize=*integer*|1
Size, in kilobytes, of the read-ahead buffer, in the range 1 through 999 999.

The following special values exist:

-1
The client determines the appropriate value.

0
Read ahead is disabled for the client.

PurgeTime=*integer*|600
Interval, in seconds, after which messages left in the read-ahead buffer are purged.

If the client application is selecting messages based on MsgId or CorrelId it is possible that the read ahead buffer might contain messages sent to the client with a previously requested MsgId or CorrelId. These messages would then be stranded in the read ahead buffer until an MQGET is issued with an appropriate MsgId or CorrelId. You can purge messages from the read ahead buffer by setting PurgeTime. Any messages that have remained in the read ahead buffer for longer than the purge

interval are automatically purged. These messages have already been removed from the queue on the queue manager, so unless they are being browsed, they are lost.

The valid range is in the range 1 through 999 999 seconds, or the special value 0, meaning that no purge takes place.

UpdatePercentage=*integer*|__-1__

The update percentage value, in the range of 1 - 100, used in calculating the threshold value to determine when a client application makes a new request to the server. The special value -1 indicates that the client determines the appropriate value.

The client periodically sends a request to the server indicating how much data the client application has consumed. A request is sent when the number of bytes, *n*, retrieved by the client by way of MQGET calls exceeds a threshold *T*. *n* is reset to zero each time a new request is sent to the server.

The threshold T is calculated as follows:

```
T = Upper - Lower
```

Upper is the same as the read-ahead buffer size, specified by the *MaximumSize* attribute, in Kilobytes. Its default is 100 Kb.

Lower is lower than Upper, and is specified by the *UpdatePercentage* attribute. This attribute is a number in the range 1 through 100, and has a default of 20. Lower is calculated as follows:

```
Lower = Upper x UpdatePercentage / 100
```

Example 1:

The MaximumSize and UpdatePercentage attributes take their defaults of 100 Kb and 20 Kb.

The client calls MQGET to retrieve a message, and does so repeatedly. This continues until MQGET has consumed n bytes.

Using the calculation

```
T = Upper - Lower
```

T is (100 - 20) = 80 Kb.

So when MQGET calls have removed 80 Kb from a queue, the client makes a new request automatically.

Example 2:

The MaximumSize attributes takes its default of 100 Kb, and a value of 40 is chosen for UpdatePercentage.

The client calls MQGET to retrieve a message, and does so repeatedly. This continues until MQGET has consumed n bytes.

Using the calculation

```
T = Upper - Lower
```

T is (100 - 40) = 60 Kb

So when MQGET calls have removed 60 Kb from a queue, the client makes a new request automatically. This is sooner than in EXAMPLE 1 where the defaults were used.

Therefore choosing a larger threshold *T* tends to decrease the frequency at which requests are sent from client to server. Conversely choosing a smaller threshold *T* tends to increase the frequency of requests that are sent from client to server.

However, choosing a large threshold *T* can mean that the performance gain of read ahead is reduced as the chance of the read ahead buffer becoming empty can increase. When this happens an MQGET call might have to pause, waiting for data to arrive from the server.

SSL stanza of the client configuration file

Use the SSL stanza to specify information about the use of SSL or TLS.

The following attributes can be included in the SSL stanza:

CDPCheckExtensions=YES|NO
CDPCheckExtensions specifies whether SSL or TLS channels on this queue manager try to check CDP servers that are named in CrlDistributionPoint certificate extensions.

This attribute has the following possible values:

- YES: SSL or TLS channels try to check CDP servers to determine whether a digital certificate is revoked.
- NO: SSL or TLS channels do not try to check CDP servers. This value is the default.

CertificateLabel = *string*
The certificate label of the channel definition.

This attribute can be read by C and unmanaged .NET clients.

CertificateValPolicy=*string*
Determines the type of certificate validation used.

ANY
Use any certificate validation policy supported by the underlying secure sockets library. This setting is the default setting.

RFC5280
Use only certificate validation which complies with the RFC 5280 standard.

ClientRevocationChecks = REQUIRED | OPTIONAL | DISABLED
Determines how certificate revocation checking is configured if the client connect call uses an SSL/TLS channel. See also **OCSPAuthentication**.

This attribute can be read by C and unmanaged .NET clients.

This attribute has the following possible values:

REQUIRED (default)
Attempts to load certificate revocation configuration from the CCDT and perform revocation checking as configured. If the CCDT file cannot be opened or it is not possible to validate the certificate (because an OCSP or CRL server is not available, for example) the MQCONN call fails. No revocation checking is performed if the CCDT contains no revocation configuration but this does not cause the channel to fail.

Windows On Windows systems, you can also use Active Directory for CRL revocation checking. You cannot use Active Directory for OCSP revocation checking.

OPTIONAL
As for REQUIRED, but if it is not possible to load the certificate revocation configuration, the channel does not fail.

DISABLED
No attempt is made to load certificate revocation configuration from the CCDT and no certificate revocation checking is done.

Note: If you are using MQCONNX rather than MQCONN calls, you might choose to supply authentication information records (MQAIR) via the MQSCO. The default behavior with MQCONNX is therefore not to fail if the CCDT file cannot be opened but to assume that you are supplying an MQAIR (even if you choose not to do so).

EncryptionPolicySuiteB=*string*
Determines whether a channel uses Suite-B compliant cryptography and what level of strength is to be used. The possible values are:

NONE
Suite-B compliant cryptography is not used. This setting is the default setting.

128_BIT,192_BIT
Sets the security strength to both 128-bit and 192-bit levels.

128_BIT
Sets the security strength to 128-bit level.

192_BIT
Sets the security strength to 192-bit level.

OCSPAuthentication=OPTIONAL|REQUIRED|WARN
Defines the behavior of WebSphere MQ when OCSP is enabled and the OCSP revocation check is unable to determine the certificate revocation status. There are three possible values:

OPTIONAL
Any certificate with a revocation status that cannot be determined by OCSP checking is accepted and no warning or error message is generated. The SSL or TLS connection continues as if no revocation check had been made.

REQUIRED
OCSP checking must yield a definitive revocation result for every SSL or TLS certificate which is checked. Any SSL or TLS certificate with a revocation status that cannot be verified is rejected with an error message. If queue manager SSL event messages are enabled, an MQRC_CHANNEL_SSL_ERROR message with a ReasonQualifier of MQRQ_SSL_HANDSHAKE_ERROR is generated. The connection is closed.

This value is the default value.

WARN
A warning is reported in the queue manager error logs if an OCSP revocation check is unable to determine the revocation status of any SSL or TLS certificate. If queue manager SSL event messages are enabled, an MQRC_CHANNEL_SSL_WARNING message with a ReasonQualifier of MQRQ_SSL_UNKNOWN_REVOCATION is generated. The connection is allowed to continue

OCSPCheckExtensions=YES|NO
Controls whether WebSphere MQ acts on AuthorityInfoAccess certificate extensions. If the value is set to NO, WebSphere MQ ignores AuthorityInfoAccess certificate extensions and does not attempt an OCSP security check. The default value is YES.

SSLCryptoHardware=*string*
Sets the parameter string required to configure PKCS #11 cryptographic hardware present on the system.

Specify a string in the following format: GSK_PKCS11=*driver path and filename*;*token label*;*token password*;*symmetric cipher setting*;

For example: GSK_PKCS11=/usr/lib/pkcs11/
PKCS11_API.so;tokenlabel;passw0rd;SYMMETRIC_CIPHER_ON

The driver path is an absolute path to the shared library providing support for the PKCS #11 card. The driver file name is the name of the shared library. An example of the value required for the PKCS #11 driver path and file name is /usr/lib/pkcs11/PKCS11_API.so. To access symmetric cipher operations through GSKit, specify the symmetric cipher setting parameter. The value of this parameter is either:

SYMMETRIC_CIPHER_OFF
Do not access symmetric cipher operations. This setting is the default setting.

SYMMETRIC_CIPHER_ON
Access symmetric cipher operations.

The maximum length of the string is 256 characters. The default value is blank. If you specify a string that is not in the correct format, an error is generated.

SSLFipsRequired=YES|NO
Specifies whether only FIPS-certified algorithms are to be used if cryptography is carried out in WebSphere MQ. If cryptographic hardware is configured, the cryptographic modules used are those modules provided by the hardware product. These might, or might not, be FIPS-certified to a particular level, depending on the hardware product in use.

SSLHTTPProxyName=*string*
The string is either the host name or network address of the HTTP Proxy server that is to be used by GSKit for OCSP checks. This address can be followed by an optional port number, enclosed in parentheses. If you do not specify the port number, the default HTTP port, 80, is used. On the HP-UX PA-RISC and Sun Solaris SPARC platforms, and for 32-bit clients on AIX, the network address can be only an IPv4 address; on other platforms it can be an IPv4 or IPv6 address.

This attribute might be necessary if, for example, a firewall prevents access to the URL of the OCSP responder.

SSLKeyRepository=*pathname*
The location of the key repository that holds the user's digital certificate, in stem format. That is, it includes the full path and the file name without an extension.

SSLKeyResetCount=*integer*|0
The number of unencrypted bytes sent and received on an SSL or TLS channel before the secret key is renegotiated.

The value must be in the range 0 - 999999999.

The default is 0, which means that secret keys are never renegotiated.

If you specify a value of 1 - 32768, SSL or TLS channels use a secret key reset count of 32768 (32Kb). This is to avoid excessive key resets, which would occur for small secret key reset values.

TCP stanza of the client configuration file

Use the TCP stanza to specify TCP network protocol configuration parameters.

The following attributes can be included in the TCP stanza:

ClntRcvBuffSize=*number*|32768
The size in bytes of the TCP/IP receive buffer used by the client end of a client-connection server-connection channel. A value of zero indicates that the operating system will manage the buffer sizes, as opposed to the buffer sizes being fixed by WebSphere MQ.

ClntSndBuffSize=*number*|32768
The size in bytes of the TCP/IP send buffer used by the client end of a client-connection server-connection channel. A value of zero indicates that the operating system will manage the buffer sizes, as opposed to the buffer sizes being fixed by WebSphere MQ.

Connect_Timeout=*number*
The number of seconds before an attempt to connect the socket times out; the default is 0 unless the channel has been configured with a non-zero client channel weighting in which case the default is 5.

IPAddressVersion=MQIPADDR_IPV4|MQIPADDR_IPV6
Specifies which IP protocol to use for a channel connection.

It has the possible string values of MQIPADDR_IPV4 or MQIPADDR_IPV6. These values have the same meanings as IPV4 and IPV6 in **ALTER QMGR IPADDRV**.

KeepAlive=YES|NO
Switch the KeepAlive function on or off. KeepAlive=YES causes TCP/IP to check periodically that the other end of the connection is still available. If it is not, the channel is closed.

Windows **Library1=*DLLName*|WSOCK32**
(Windows only) The name of the TCP/IP sockets DLL.

TMF and TMF/Gateway stanzas

The IBM WebSphere MQ provided TMF/Gateway runs in a Pathway environment. Use the TMF and TMF/Gateway stanzas to specify the required configuration parameters for the IBM WebSphere MQ client for HP Integrity NonStop Server to communicate with the TMF/Gateway.

If you want to use TMF, then you must define a TMF stanza and one TmfGateway stanza for each queue manager with which you are communicating. All values are derived from your configuration.

TMF stanza

PathMon=*name*
The name of your defined Pathmon process that defines the server classes for the TMF/Gateway.

TmfGateway stanza

The following attributes can be included in this stanza:

QManager=*name*
The name of the queue manager.

Server=*name*
The server class name for the TMF/Gateway configured for that queue manager.

Example

Here is an example of a TMF stanza that is defined with two TmfGateway stanzas for two different queue managers on different servers:

```
TMF:
   PathMon=$PSD1P

TmfGateway:
   QManager=MQ5B
   Server=MQ-MQ5B

TmfGateway:
   QManager=MQ5C
   Server=MQ-MQ5C
```

Using WebSphere MQ environment variables

This section describes the environment variables that you can use with WebSphere MQ MQI client applications.

You can use environment variables in the following ways:

- Set the variables in your system profile to make a permanent change
- Issue a command from the command line to make a change for this session only
- To give one or more variables a particular value dependent on the application that is running, add commands to a command script file used by the application

WebSphere MQ uses default values for those variables that you have not set.

Commands are available on all the WebSphere MQ MQI client platforms unless otherwise stated.

For each environment variable, use the command relevant to your platform to display the current setting or to reset the value of a variable.

For example:

Setting or resetting the value of an environment variable		
Effect	**Command**	
	Windows	**UNIX and Linux systems**
Removes the variable	SET MQSERVER=	unset MQSERVER
Displays the current setting	SET MQSERVER	echo $MQSERVER
Displays all environment variables for the session	set	set

For information about the individual variables, see the following subtopics:

Related concepts

"Configuring a client using a configuration file" on page 123
Configure your clients using attributes in a text file. These attributes can be overridden by environment variables or in other platform-specific ways.

Related reference

Environment variables

MQCCSID

MQCCSID specifies the coded character set number to be used and overrides the CCSID value with which the server has been configured.

See Choosing client or server coded character set identifier (CCSID) for more information.

To set this variable use one of these commands:

- For Windows:

```
SET MQCCSID=number
```

- For UNIX and Linux systems:

```
export MQCCSID=number
```

MQCERTVPOL

MQCERTVPOL specifies the certificate validation policy used.

For more information about certificate validation policies in WebSphere MQ, see Certificate validation policies in WebSphere MQ.

This environment variable overrides the *CertificateValPolicy* setting in the SSL stanza of the client ini file. The variable can be set to one of two values:

ANY
Use any certificate validation policy supported by the underlying secure sockets library.

RFC5280
Use only certificate validation which complies with the RFC 5280 standard.

To set this variable, use one of these commands:

- For Windows:

```
SET MQCERTVPOL=value
```

- For UNIX and Linux systems:

```
export MQCERTVPOL=value
```

MQCHLLIB

MQCHLLIB specifies the directory path to the file containing the client channel definition table (CCDT). The file is created on the server, but can be copied across to the WebSphere MQ MQI client workstation.

If MQCHLLIB is not set, the path for the client defaults to:

- Windows For Windows: *MQ_INSTALLATION_PATH*
- UNIX Linux For UNIX and Linux systems: /var/mqm/

For the **crtmqm** and **strmqm** commands, the path defaults to one of two sets of paths. If *datapath* is set, the path defaults to one of the first set. If *datapath* is not set, the path defaults to one of the second set.

- Windows For Windows: *datapath*\@ipcc
- UNIX Linux For UNIX and Linux systems: *datapath*/@ipcc

Or:

- Windows For Windows: *MQ_INSTALLATION_PATH*\data\qmgrs*qmgrname*\@ipcc
- UNIX Linux For UNIX and Linux systems: /prefix/qmgrs/*qmgrname*/@ipcc

where:

- *MQ_INSTALLATION_PATH* represents the high-level directory in which IBM WebSphere MQ is installed.
- If present, *datapath* is the value of DataPath defined in the queue manager stanza.
- *prefix* is the value of Prefix defined in the queue manager stanza. Prefix is typically /var/mqm on UNIX and Linux platforms.
- *qmgrname* is the value of the Directory attribute defined in the queue manager stanza. The value might be different from the actual queue manager name. The value might have been altered to replace special characters.
- The queue manager stanza is defined in the mqs.ini file on UNIX, and Linux, and in the registry on Windows

Notes:

1. If set, MQCHLLIB overrides the path used to locate the CCDT.
2. Environment variables, such as MQCHLLIB, can be scoped to a process, or a job, or system-wide, in a platform-specific way.
3. If you set MQCHLLIB system-wide on a server, it sets the same path to the CCDT file for all the queue managers on the server. If you do not set the MQCHLLIB environment variable, the path is different for each queue manager. Queue managers read the value of MQCHLLIB, if it is set, on either the **crtmqm** or **strmqm** command.
4. If you create multiple queue managers on one server, the distinction is important, for the following reason. If you set MQCHLLIB system-wide, each queue manager updates the same CCDT file. The file contains the client-connection definitions from all the queue managers on the server. If the same definition exists on multiple queue managers, SYSTEM.DEF.CLNTCONN for example, the file contains the latest definition. When you create a queue manager, if MQCHLLIB is set, SYSTEM.DEF.CLNTCONN is updated in the CCDT. The update overwrites the SYSTEM.DEF.CLNTCONN created by a different queue manager. If you modified the earlier definition, your modifications are lost. For this reason, you must consider finding alternatives to setting MQCHLLIB as a system-wide environment variable on the server.
5. The MQSC and PCF NOREPLACE option on a client-connection definition does not check the contents of the CCDT file. A client-connection channel definition of the same name that was previously created, but not by this queue manager, is replaced, regardless of the NOREPLACE option. If the definition was previously created by the same queue manager, the definition is not replaced.

6. The command, **rcrmqobj** `-t clchltab` deletes and recreates the CCDT file. The file is recreated with only the client-connection definitions created on the queue manager that the command is running against.
7. Other commands that update the CCDT modify only the client-connection channels that have the same channel name. Other client-connection channels in the file are not altered.
8. The path for MQCHLLIB does not need quotations marks.

Examples

To set this variable use one of these commands:

- Windows For Windows:

```
SET MQCHLLIB=pathname
```

  For example:

```
SET MQCHLLIB=C:\wmqtest
```

- UNIX Linux For UNIX and Linux systems:

```
export MQCHLLIB=pathname
```

MQCHLTAB

MQCHLTAB specifies the name of the file containing the client channel definition table (ccdt). The default file name is AMQCLCHL.TAB.

For information about where the client channel definition table is located on a server, see “Client channel definition table” on page 114.

To set this variable use one of these commands:

- On Windows:

```
SET MQCHLTAB=filename
```

- On UNIX and Linux systems:

```
export MQCHLTAB=filename
```

For example:

```
SET MQCHLTAB=ccdf1.tab
```

In the same way as for the client, the MQCHLTAB environment variable on the server specifies the name of the client channel definition table.

MQIPADDRV

MQIPADDRV specifies which IP protocol to use for a channel connection. It has the possible string values of "MQIPADDR_IPV4" or "MQIPADDR_IPV6". These values have the same meanings as IPV4 and IPV6 in ALTER QMGR IPADDRV. If it is not set, "MQIPADDR_IPV4" is assumed.

To set this variable use one of these commands:

- For Windows:

```
SET MQIPADDRV=MQIPADDR_IPV4|MQIPADDR_IPV6
```

- For UNIX and Linux systems:

```
export MQIPADDRV=MQIPADDR_IPV4|MQIPADDR_IPV6
```

MQNAME

MQNAME specifies the local NetBIOS name that the WebSphere MQ processes can use.

See “Defining a NetBIOS connection on Windows” on page 85 for a full description and for the rules of precedence on the client and the server.

To set this variable use this command:

```
SET MQNAME=Your_env_Name
```

For example:

```
SET MQNAME=CLIENT1
```

The NetBIOS on some platforms requires a different name (set by MQNAME) for each application if you are running multiple WebSphere MQ applications simultaneously on the WebSphere MQ MQI client.

MQSERVER

MQSERVER environment variable is used to define a minimal channel. MQSERVER specifies the location of the WebSphere MQ server and the communication method to be used.

You cannot use MQSERVER to define an SSL channel or a channel with channel exits. For details of how to define an SSL channel, see Protecting channels with SSL.

ConnectionName must be a fully-qualified network name. The *ChannelName* cannot contain the forward slash (/) character because this character is used to separate the channel name, transport type, and connection name. When the MQSERVER environment variable is used to define a client channel, a maximum message length (MAXMSGL) of 100 MB is used. Therefore the maximum message size in effect for the channel is the value specified in the SVRCONN channel at the server.

To set this variable use one of these commands:

- For Windows:

```
SET MQSERVER=ChannelName/TransportType/ConnectionName
```

- For UNIX and Linux systems:

```
export MQSERVER='ChannelName/TransportType/ConnectionName'
```

TransportType can be one of the following values, depending on your IBM WebSphere MQ client platform:

- LU62
- TCP
- NETBIOS
- SPX

ConnectionName can be a comma-separated list of connection names. The connection names in the list are used in a similar way to multiple connections in a client connection table. The *ConnectionName* list might be used as an alternative to queue manager groups to specify multiple connections for the client to try. If you are configuring a multi-instance queue manager, you might use a *ConnectionName* list to specify different queue manager instances.

TCP/IP default port

By default, for TCP/IP, WebSphere MQ assumes that the channel will be connected to port 1414.

You can change this by:

- Adding the port number in brackets as the last part of the ConnectionName:
  - For Windows:

```
SET MQSERVER=ChannelName/TransportType/ConnectionName(PortNumber)
```

  - For UNIX and Linux systems:

```
export MQSERVER='ChannelName/TransportType/ConnectionName(PortNumber)'
```

- Changing the `mqclient.ini` file by adding the port number to the protocol name, for example:

```
TCP:
port=2001
```

- Adding WebSphere MQ to the services file as described in “Using the TCP/IP listener” on page 92.

SPX default socket

By default, for SPX, WebSphere MQ assumes that the channel will be connected to socket 5E86.

You can change this by:

- Adding the socket number in brackets as the last part of the ConnectionName:

```
SET MQSERVER=ChannelName/TransportType/ConnectionName(SocketNumber)
```

  For SPX connections, specify the ConnectionName and socket in the form `network.node(socket)`. If the WebSphere MQ client and server are on the same network, the network need not be specified. If you are using the default socket, the socket need not be specified.

- Changing the `qm.ini` file by adding the port number to the protocol name, for example:

```
SPX:
socket=5E87
```

Using MQSERVER

If you use the MQSERVER environment variable to define the channel between your WebSphere MQ MQI client machine and a server machine, this is the only channel available to your application, and no reference is made to the client channel definition table (CCDT).

In this situation, the listener program that you have running on the server machine determines the queue manager to which your application will connect. It will be the same queue manager as the listener program is connected to.

If the MQCONN or MQCONNX request specifies a queue manager other than the one the listener is connected to, or if the MQSERVER parameter *TransportType* is not recognized, the MQCONN or MQCONNX request fails with return code MQRC_Q_MGR_NAME_ERROR.

OnUNIX and Linux systems, you might define MQSERVER as in one of the following examples:

```
export MQSERVER=CHANNEL1/TCP/'9.20.4.56(2002)'
export MQSERVER=CHANNEL1/LU62/BOX99
```

All MQCONN or MQCONNX requests then attempt to use the channel you have defined unless an MQCD structure has been referenced from the MQCNO structure supplied to MQCONNX, in which case the channel specified by the MQCD structure takes priority over any specified by the MQSERVER environment variable.

The MQSERVER environment variable takes priority over any client channel definition pointed to by MQCHLLIB and MQCHLTAB.

Canceling MQSERVER

To cancel MQSERVER and return to the client channel definition table pointed to by MQCHLLIB and MQCHLTAB, enter the following:

- On Windows:

```
SET MQSERVER=
```

- On UNIX and Linux systems:

```
unset MQSERVER
```

MQSSLCRYP

MQSSLCRYP holds a parameter string that allows you to configure the cryptographic hardware present on the system. The permitted values are the same as for the SSLCRYP parameter of the ALTER QMGR command.

To set this variable use one of these commands:

- On Windows systems:

```
SET MQSSLCRYP=string
```

- On UNIX and Linux systems:

```
export MQSSLCRYP=string
```

Related reference
ALTER QMGR command **SSLCRYP** parameter

MQSSLFIPS

MQSSLFIPS specifies whether only FIPS-certified algorithms are to be used if cryptography is carried out in WebSphere MQ. The values are the same as for the SSLFIPS parameter of the ALTER QMGR command.

The use of FIPS-certified algorithms is affected by the use of cryptographic hardware, see Specifying that only FIPS-certified CipherSpecs are used at run time on the MQI client.

To set this variable use one of these commands:

- On Windows systems:

```
SET MQSSLFIPS=YES|NO
```

- On UNIX and Linux systems:

```
export MQSSLFIPS=YES|NO
```

The default is NO.

MQSSLKEYR

MQSSLKEYR specifies the location of the key repository that holds the digital certificate belonging to the user, in stem format. Stem format means that it includes the full path and the file name without an extension. For full details, see the SSLKEYR parameter of the ALTER QMGR command.

To set this variable use one of these commands:

- On Windows systems:

```
SET MQSSLKEYR=pathname
```

- On UNIX and Linux systems:

```
export MQSSLKEYR=pathname
```

There is no default value.

MQSSLPROXY

MQSSLPROXY specifies the host name and port number of the HTTP proxy server to be used by GSKit for OCSP checks.

To set this variable use one of these commands:

- On Windows systems:

```
SET MQSSLPROXY=string
```

- On UNIX and Linux systems:

```
export MQSSLPROXY="string"
```

The string is either the host name or network address of the HTTP Proxy server which is to be used by GSKit for OCSP checks. This address can be followed by an optional port number, enclosed in parentheses. If you do not specify the port number, the default HTTP port, 80, is used.

For example, on UNIX and Linux systems, you can use the one of the following commands:

- `export MQSSLPROXY="proxy.example.com(80)"`
- `export MQSSLPROXY="127.0.0.1"`

MQSSLRESET

MQSSLRESET represents the number of unencrypted bytes sent and received on an SSL or TLS channel before the secret key is renegotiated.

See Resetting SSL and TLS secret keys for more information about secret key renegotiation.

It can be set to an integer in the range 0 through 999 999 999. The default is 0, which indicates that secret keys are never renegotiated. If you specify an SSL or TLS secret key reset count in the range 1 byte through 32 KB, SSL or TLS channels use a secret key reset count of 32 KB. This secret reset count is to avoid excessive key resets which would occur for small SSL or TLS secret key reset values.

To set this variable use one of these commands:

- On Windows systems:

```
SET MQSSLRESET=integer
```

- On UNIX and Linux systems:

```
export MQSSLRESET=integer
```

Controlling queued publish/subscribe

You can start, stop and display the status of queued publish/subscribe. You can also add and remove streams, and add and delete queue managers from a broker hierarchy.

See the following subtopics for more information on controlling queued publish/subscribe:

Setting queued publish/subscribe message attributes

You control the behavior of some publish/subscribe message attributes using queue manager attributes. The other attributes you control in the *Broker* stanza of the `qm.ini` file.

About this task

You can set the following publish/subscribe attributes: for details see, Queue manager parameters

Table 23. Publish/subscribe configuration parameters

Description	**MQSC parameter name**
Command message retry count	**PSRTYCNT**
Discard undeliverable command input message	**PSNPMSG**
Behavior following undeliverable command response message	**PSNPRES**
Process command messages under syncpoint	**PSSYNCPT**

The Broker stanza is used to manage the following configuration settings:

- `PersistentPublishRetry=yes | force`

  If you specify `Yes`, then if a publication of a persistent message through the queued publish/subscribe interface fails, and no negative reply was requested, the publish operation is retried.

  If you requested a negative response message, the negative response is sent and no further retry occurs.

  If you specify `Force`, then if a publication of a persistent message through the queued publish/subscribe interface fails, the publish operation is retried until the it is successfully processed. No negative response is sent.

- `NonPersistentPublishRetry=yes | force`

  If you specify `Yes`, then if a publication of a non-persistent message through the queued publish/subscribe interface fails, and no negative reply was requested, the publish operation is retried.

  If you requested a negative response message, the negative response is sent and no further retry occurs.

  If you specified `Force`, then if a publication of a non-persistent message through the queued publish/subscribe interface fails, the publish operation is retried until it is successfully processed. No negative response is sent.

  Note: If you want to enable this functionality for non-persistent messages, then as well as setting the `NonPersistentPublishRetry` value you must also ensure that the queue manager attribute **PSSYNCPT** is set to `Yes`.

  Doing this might also have an impact on the performance of processing non-persistent publications as the **MQGET** from the STREAM queue now occurs under syncpoint.

- `PublishBatchSize=`*number*

  The broker normally processes publish messages within syncpoint. It can be inefficient to commit each publication individually, and in some circumstances the broker can process multiple publish messages

in a single unit of work. This parameter specifies the maximum number of publish messages that can be processed in a single unit of work

The default value for PublishBatchSize is 5.

- PublishBatchInterval=*number*

  The broker normally processes publish messages within syncpoint. It can be inefficient to commit each publication individually, and in some circumstances the broker can process multiple publish messages in a single unit of work. This parameter specifies the maximum time (in milliseconds) between the first message in a batch and any subsequent publication included in the same batch.

  A batch interval of 0 indicates that up to PublishBatchSize messages can be processed, provided that the messages are available immediately.

  The default value for PublishBatchInterval is zero.

Procedure

Use WebSphere MQ Explorer, programmable commands, or the **runmqsc** command to alter the queue manager attributes that control the behavior of publish/subscribe.

Example

ALTER QMGR PSNPRES(SAFE)

Starting queued publish/subscribe

Before you begin

Read the description of PSMODE to understand the three modes of publish/subscribe:

- COMPAT
- DISABLED
- ENABLED

Note: If you have migrated from Version 6.0 you must use **strmqbrk** to migrate Version 6.0 publish/subscribe broker state if you are working with an upgraded queue manager. This does not apply to z/OS.

About this task

Set the QMGR PSMODE attribute to start either the queued publish/subscribe interface (also known as the broker), or the publish/subscribe engine (also known as Version 7 publish/subscribe) or both. To start queued publish/subscribe you need to set PSMODE to ENABLED. The default is ENABLED.

Procedure

Use WebSphere MQ Explorer or the **runmqsc** command to enable the queued publish/subscribe interface if the interface is not already enabled.

Example

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

What to do next

WebSphere MQ processes queued publish/subscribe commands and publish/subscribe Message Queue Interface (MQI) calls.

Stopping queued publish/subscribe

Before you begin

Queued publish/subscribe is deprecated.

Read the description of PSMODE to understand the three modes of publish/subscribe:

- COMPAT
- DISABLED
- ENABLED

About this task

Set the QMGR PSMODE attribute to stop either the queued publish/subscribe interface (also known as the broker), or the publish/subscribe engine (also known as Version 7 publish/subscribe) or both. To stop queued publish/subscribe you need to set PSMODE to COMPAT. To stop the publish/subscribe engine entirely, set PSMODE to DISABLED.

Procedure

Use WebSphere MQ Explorer or the **runmqsc** command to disable the queued publish/subscribe interface.

Example

```
ALTER QMGR PSMODE(COMPAT)
```

Adding a stream

You can add streams manually so that they coexist with streams migrated from Version 6.0 queue managers.

Before you begin

Familiarize yourself with the way publish/subscribe streams operate by reading the topic, Streams and topics.

About this task

Use PCF command, **runmqsc**, or IBM WebSphere MQ Explorer to do these steps.

Note: You can perform steps 1 and 2 in any order. Only perform step 3 after steps 1 and 2 have both been completed.

Procedure

1. Define a local queue with the same name as the Version 6.0 stream.
2. Define a local topic with the same name as the Version 6.0 stream.
3. Add the name of the queue to the namelist, SYSTEM.QPUBSUB.QUEUE.NAMELIST
4. Repeat for all queue managers at Version 7.1 or above that are in the publish/subscribe hierarchy.

Adding 'Sport'

In the example of sharing the stream 'Sport', Version 6.0 and Version 7.1 queue managers are working in the same publish/subscribe hierarchy. The Version 6.0 queue managers share a stream called 'Sport'. The example shows how to create a queue and a topic on Version 7.1 queue managers called 'Sport', with a topic string 'Sport' that is shared with the version 6 stream 'Sport'.

A Version 7.1 publish application, publishing to topic `'Sport'`, with topic string `'Soccer/Results'` , creates the resultant topic string `'Sport/Soccer/Results'` . On Version 7.1 queue managers, subscribers to topic `'Sport'`, with topic string `'Soccer/Results'` receive the publication.

On Version 6.0 queue managers, subscribers to stream `'Sport'` , with topic string `'Soccer/Results'` receive the publication.

```
runmqsc QM1
5724-H72 (C) Copyright IBM Corp. 1994, 2025. ALL RIGHTS RESERVED.
Starting MQSC for queue manager QM1.
define qlocal('Sport')
     1 : define qlocal('Sport')
AMQ8006: WebSphere MQ queue created.
define topic('Sport') topicstr('Sport')
     2 : define topic('Sport') topicstr('Sport')
AMQ8690: WebSphere MQ topic created.
alter namelist(SYSTEM.QPUBSUB.QUEUE.NAMELIST) NAMES('Sport', 'SYSTEM.BROKER.DEFAULT.STREAM',
'SYSTEM.BROKER.ADMIN.STREAM')
     3 : alter namelist(SYSTEM.QPUBSUB.QUEUE.NAMELIST) NAMES('Sport',
'SYSTEM.BROKER.DEFAULT.STREAM', 'SYSTEM.BROKER.ADMIN.STREAM')
AMQ8551: WebSphere MQ namelist changed.
```

Note: You need both to provide the existing names in the namelist object, as well as the new names that you are adding, to the **`alter namelist`** command.

What to do next

Information about the stream is passed to other brokers in the hierarchy.

If a broker is Version 6.0, administer it as a Version 6.0 broker. That is, you have a choice of creating the stream queue manually, or letting the broker create the stream queue dynamically when it is needed. The queue is based on the model queue definition, `SYSTEM.BROKER.MODEL.STREAM`.

If a broker is Version 7.1, you must configure each Version 7.1 queue manager in the hierarchy manually.

Deleting a stream

You can delete a stream from an IBM WebSphere MQ Version 7.1, or later, queue manager.

Before you begin

The use of queued publish/subscribe is deprecated in IBM WebSphere MQ Version 7.1.

Before deleting a stream you must ensure that there are no remaining subscriptions to the stream and quiesce all applications that use the stream. If publications continue to flow to a deleted stream, it takes a lot of administrative effort to restore the system to a cleanly working state.

About this task

For instructions on deleting the stream from any Version 6.0 queue managers it is connected to, see Deleting a stream (ps11870_.htm in the v6.0 documentation).

Procedure

1. Find all the connected brokers that host this stream.
2. Cancel all subscriptions to the stream on all the brokers.
3. Remove the queue (with the same name as the stream) from the namelist, `SYSTEM.QPUBSUB.QUEUE.NAMELIST`.
4. Delete or purge all the messages from the queue with the same name as the stream.
5. Delete the queue with the same name as the stream.
6. Delete the associated topic object.

What to do next

1. Repeat steps 3 to 5 on all the other connected Version 7.1, or later, queue managers hosting the stream.
2. Remove the stream from all other connected Version 6.0, or earlier, queue managers.

Adding a subscription point

How to add a subscription point that was not migrated from IBM WebSphere MQ Event Broker or IBM WebSphere MQ Message Broker by **migmbbrk**. Extend an existing queued publish/subscribe application that you have migrated from IBM WebSphere MQ Event Broker or IBM WebSphere MQ Message Broker with a new subscription point.

Before you begin

1. Complete the migration from IBM WebSphere MQ Event Broker and IBM WebSphere MQ Message Broker Version 6.0 to IBM WebSphere MQ Version 7.1.
2. Check that the subscription point is not already defined in SYSTEM.QPUBSUB.SUBPOINT.NAMELIST.
3. Check if there is a topic object or a topic string with the same name as the subscription point.

About this task

Existing IBM WebSphere MQ Event Broker applications use subscription points. New IBM WebSphere MQ Version 7.1 applications do not use subscription points, but they can interoperate with existing applications that do, using the subscription point migration mechanism.

A subscription point might not have been migrated by **migmbbrk**, if the subscription point was not in use at the time of the migration.

You might want to add a subscription point to existing queued publish/subscribe programs migrated from IBM WebSphere MQ Event Broker.

Subscription points do not work with queued publish/subscribe programs that use MQRFH1 headers, which have been migrated from IBM WebSphere MQ Version 6.0, or earlier.

There is no need to add subscription points to use integrated publish/subscribe applications written for IBM WebSphere MQ Version 7.1.

Procedure

1. Add the name of the subscription point to SYSTEM.QPUBSUB.SUBPOINT.NAMELIST.
   - On z/OS, the **NLTYPE** is NONE, the default.
   - Repeat the step on every queue manager that is connected in the same publish/subscribe topology.
2. Add a topic object, preferably giving it the name of the subscription point, with a topic string matching the name of the subscription point.
   - If the subscription point is in a cluster, add the topic object as a cluster topic on the cluster topic host.
   - If a topic object exists with the same topic string as the name of the subscription point, use the existing topic object. You must understand the consequences of the subscription point reusing an existing topic. If the existing topic is part of an existing application, you must resolve the collision between two identically named topics.
   - If a topic object exists with the same name as the subscription point, but a different topic string, create a topic with a different name.
3. Set the **Topic** attribute WILDCARD to the value BLOCK.

   Blocking subscriptions to # or * isolates wildcard subscriptions to subscription points, see Wildcards and subscription points.

4. Set any attributes that you require in the topic object.

Example

The example shows a **runmqsc** command file that adds two subscription points, USD and GBP.

```
DEFINE TOPIC(USD) TOPICSTR(USD)
DEFINE TOPIC(GBP) TOPICSTR(GBP) WILDCARD(BLOCK)
ALTER NL(SYSTEM.QPUBSUB.SUBPOINT.NAMELIST) NAMES(SYSTEM.BROKER.DEFAULT.SUBPOINT, USD, GBP)
```

Note:

1. Include the default subscription point in the list of subscription points added using the **ALTER** command. **ALTER** deletes existing names in the namelist.
2. Define the topics before altering the namelist. The queue manager only checks the namelist when the queue manager starts and when the namelist is altered.

Connect a queue manager to a broker hierarchy

You can connect a local queue manager to a parent queue manager to modify a broker hierarchy.

Before you begin

1. Enable queued publish/subscribe mode. See Starting queued publish/subscribe.
2. This change is propagated to the parent queue manager using an IBM WebSphere MQ connection. There are two ways to establish the connection.
   - Connect the queue managers to an IBM WebSphere MQ cluster, see Adding a queue manager to a cluster
   - Establish a point-to-point channel connection using a transmission queue, or queue manager alias, with the same name as the parent queue manager. For more information about how to establish a point-to-point channel connection, see WebSphere MQ distributed-messaging techniques.

About this task

Use the ALTER QMGR PARENT *(PARENT_NAME)* runmqsc command to connect children to parents.

Distributed publish/subscribe is implemented by using queue manager clusters and clustered topic definitions. For interoperability with IBM WebSphere MQ Version 6.0 and WebSphere Message Broker Version 6.1 and WebSphere Event Broker Version 6.1 and earlier, you can also connect Version 7.1 or later queue managers to a broker hierarchy as long as queued publish/subscribe mode is enabled.

Procedure

ALTER QMGR PARENT(PARENT)

Example

The first example shows how to attach QM2 as a child of QM1, and then querying QM2 for its connection:

```
C:>runmqsc QM2
5724-H72 (C) Copyright IBM Corp. 1994, 2025. ALL RIGHTS RESERVED.
Starting MQSC for queue manager QM2
alter qmgr parent(QM1)
     1 : alter qmgr parent(QM1)
AMQ8005: WebSphere MQ queue manager changed.
display pubsub all
     2 : display pubsub all
AMQ8723: Display pub/sub status details.
   QMNAME(QM2)                             TYPE(LOCAL)
   STATUS(ACTIVE)
AMQ8723: Display pub/sub status details.
   QMNAME(QM1)                             TYPE(PARENT)
   STATUS(ACTIVE)
```

The next example shows the result of querying QM1 for its connections:

```
C:\Documents and Settings\Admin>runmqsc QM1
5724-H72 (C) Copyright IBM Corp. 1994, 2025. ALL RIGHTS RESERVED.
Starting MQSC for queue manager QM1.
display pubsub all
     2 : display pubsub all
AMQ8723: Display pub/sub status details.
   QMNAME(QM1)                             TYPE(LOCAL)
   STATUS(ACTIVE)
AMQ8723: Display pub/sub status details.
   QMNAME(QM2)                             TYPE(CHILD)
   STATUS(ACTIVE)
```

What to do next

You can define topics on one broker or queue manager that are available to publishers and subscribers on the connected queue managers. For more information, see Defining an administrative topic

Related concepts

Streams and topics

Introduction to WebSphere MQ publish/subscribe messaging

Related reference

DISPLAY PUBSUB

Disconnect a queue manager from a broker hierarchy

Disconnect a child queue manager from a parent queue manager in a broker hierarchy.

About this task

Use the **ALTER QMGR** command to disconnect a queue manager from a broker hierarchy. You can disconnect a queue manager in any order at any time.

The corresponding request to update the parent is sent when the connection between the queue managers is running.

Procedure

```
ALTER QMGR PARENT('')
```

Example

```
C:\Documents and Settings\Admin>runmqsc QM2
5724-H72 (C) Copyright IBM Corp. 1994, 2025. ALL RIGHTS RESERVED.
Starting MQSC for queue manager QM2.
    1 : alter qmgr parent('')
AMQ8005: WebSphere MQ queue manager changed.
    2 : display pubsub type(child)
AMQ8147: WebSphere MQ object  not found.
display pubsub type(parent)
    3 : display pubsub type(parent)
AMQ8147: WebSphere MQ object  not found.
```

What to do next

You can delete any streams, queues and manually defined channels that are no longer needed.

Configuring a queue manager cluster

Use the links in this topic to find out how clusters work, how to design a cluster configuration, and to get an example of how to set up a simple cluster.

Before you begin

For an introduction to clustering concepts, see the following topics:

- How clusters work
- “Comparison of clustering and distributed queuing” on page 157
- “Components of a cluster” on page 159

When you are designing your queue manager cluster you have to make some decisions. You must first decide which queue managers in the cluster are to hold the full repositories of cluster information. Any queue manager you create can work in a cluster. You can choose any number of queue managers for this purpose but the ideal number is two. For information about selecting queue managers to hold the full repositories, see “How to choose cluster queue managers to hold full repositories” on page 172.

See the following topics for more information about designing your cluster:

- “Organizing a cluster” on page 174
- “Cluster naming conventions” on page 174
- “Overlapping clusters” on page 175

Example

The smallest possible cluster contains only two queue managers. In this case both queue managers contain full repositories. You need only a few definitions to set up the cluster, and yet there is a high degree of autonomy at each queue manager.

Figure 21 on page 155 shows a cluster called DEMOCLSTR with two queue managers called QM1 and QM2.

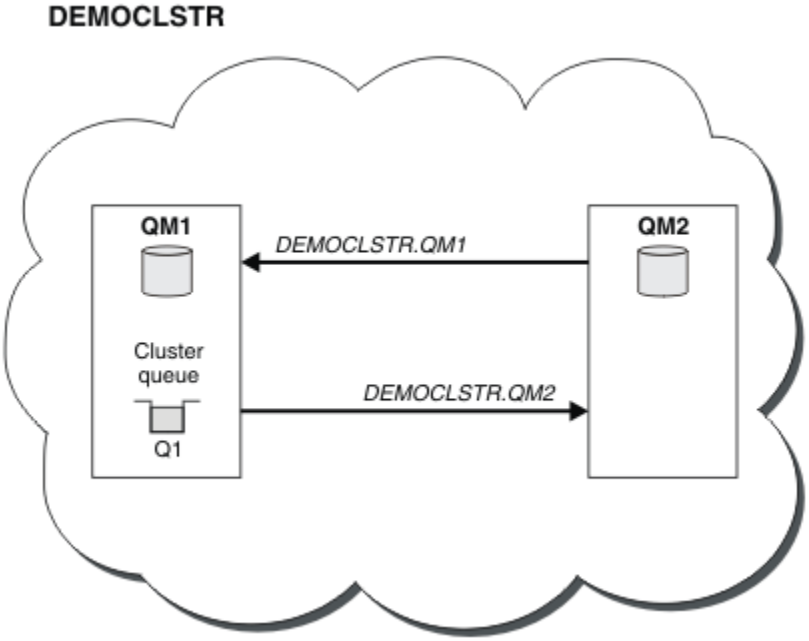


Figure 21. A small cluster of two queue managers

- The queue managers have long names such as LONDON and NEWYORK. The same names are used in the advanced and workload balancing tasks. On IBM WebSphere MQ for z/OS, queue-manager names are limited to four characters.
- The names of the queue managers imply that each queue manager is on a separate machine. You could perform these tasks with all the queue managers on the same machine.
- The tasks use IBM WebSphere MQ script commands as they would be entered by the system administrator using **MQSC** commands. There are other ways of entering commands, including using the easier IBM WebSphere MQ Explorer. The point of using WebSphere MQ script commands is to demonstrate what IBM WebSphere MQ commands are used in the tasks.

For instructions on setting up a similar example cluster, see “Setting up a new cluster” on page 181.

What to do next

See the following topics for more information about configuring and working with clusters:

- "Establishing communication in a cluster" on page 178
- "Managing IBM WebSphere MQ clusters" on page 180
- "Routing messages to and from clusters" on page 243
- "Using clusters for workload management" on page 257

For more information to help you configure your cluster, see "Clustering tips" on page 176.

Related concepts

Clusters

Access control and multiple cluster transmission queues

Choose between three modes of checking when an application puts messages to remote cluster queues. The modes are checking remotely against the cluster queue, checking locally against `SYSTEM.CLUSTER.TRANSMIT.QUEUE`, or checking against local profiles for the cluster queue, or cluster queue manager.

IBM WebSphere MQ gives you the choice of checking locally, or locally and remotely, that a user has permission to put a message to a remote queue. A typical IBM WebSphere MQ application uses local checking only, and relies on the remote queue manager trusting the access checks made on the local queue manager. If remote checking is not used, the message is put to the target queue with the authority of the remote message channel process. To use remote checking you must set the put authority of the receiving channel to context security.

The local checks are made against the queue that the application opens. In distributed queueing, the application usually opens a remote queue definition, and access checks are made against the remote queue definition. If the message is put with a full routing header, the checks are made against the transmission queue. If an application opens a cluster queue that is not on the local queue manager, there is no local object to check. The access control checks are made against the cluster transmission queue, `SYSTEM.CLUSTER.TRANSMIT.QUEUE`. Even with multiple cluster transmission queues, from Version 7.5, local access control checks for remote cluster queues are made against `SYSTEM.CLUSTER.TRANSMIT.QUEUE`.

The choice of local or remote checking is a choice between two extremes. Checking remotely is fine-grained. Every user must have an access control profile on every queue manager in the cluster to put to any cluster queue. Checking locally is coarse-grained. Every user needs only one access control profile for the cluster transmission queue on the queue manager they are connected to. With that profile, they can put a message to any cluster queue on any queue manager in any cluster.

Since Version 7.1, administrators have another way to set up access control for cluster queues. You can create a security profile for a cluster queue on any queue manager in the cluster using the **setmqaut** command. The profile takes affect if you open a remote cluster queue locally, specifying only the queue name. You can also set up a profile for a remote queue manager. If you do so, the queue manager can check the profile of a user that opens a cluster queue by providing a fully qualified name.

The new profiles work only if you change the queue manager stanza, **ClusterQueueAccessControl** to `RQMName`. The default is `Xmitq`. You must create profiles for all the cluster queues existing applications use cluster queues. If you change the stanza to `RQMName` without creating profiles the applications are likely to fail.

Tip: The changes made to cluster queue accessing checking in Version 7.1 do not apply to remote queueing. Access checks are still made against local definitions. The changes mean that you can follow the same approach to configure access checking on cluster queues and cluster topics.

Related concepts

"Clustering: Application isolation using multiple cluster transmission queues" on page 277
You can isolate the message flows between queue managers in a cluster. You can place messages being transported by different cluster-sender channels onto different cluster transmission queues. You can use

the approach in a single cluster or with overlapping clusters. The topic provides examples and some best practices to guide you in choosing an approach to use.

Comparison of clustering and distributed queuing

Compare the components that need to be defined to connect queue managers using distributed queuing and clustering.

If you do not use clusters, your queue managers are independent and communicate using distributed queuing. If one queue manager needs to send messages to another, you must define:

- A transmission queue
- A channel to the remote queue manager

Figure 22 on page 157 shows the components required for distributed queuing.

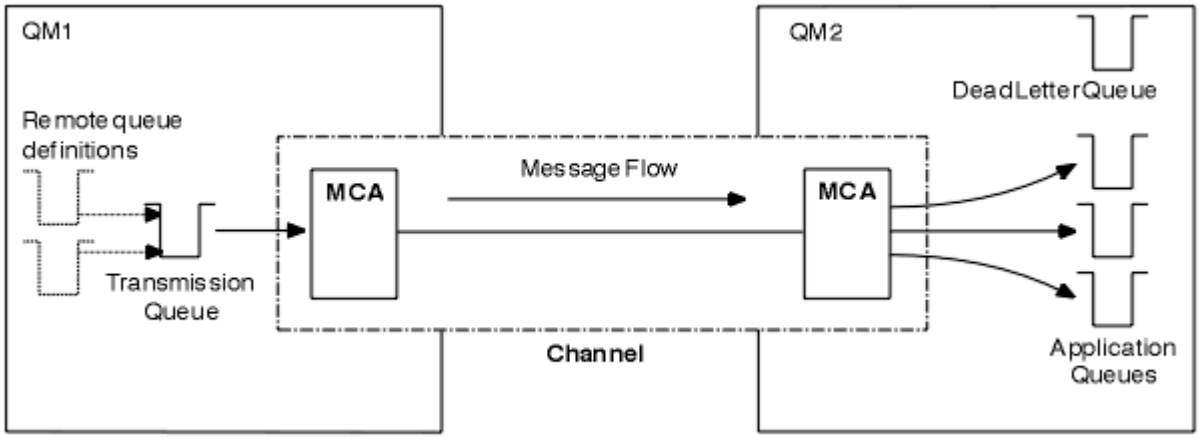


Figure 22. Distributed queuing

If you group queue managers in a cluster, queues on any queue manager are available to any other queue manager in the cluster. Any queue manager can send a message to any other queue manager in the same cluster without explicit definitions. You do not provide channel definitions, remote-queue definitions, or transmission queues for each destination. Every queue manager in a cluster has a single transmission queue from which it can transmit messages to any other queue manager in the cluster. Each queue manager in a cluster needs to define only:

- One cluster-receiver channel on which to receive messages
- One cluster-sender channel with which it introduces itself and learns about the cluster

Definitions to set up a cluster versus distributed queuing

Look at Figure 23 on page 158, which shows four queue managers each with two queues. Consider how many definitions are needed to connect these queue managers using distributed queuing. Compare how many definitions are needed to set up the same network as a cluster.

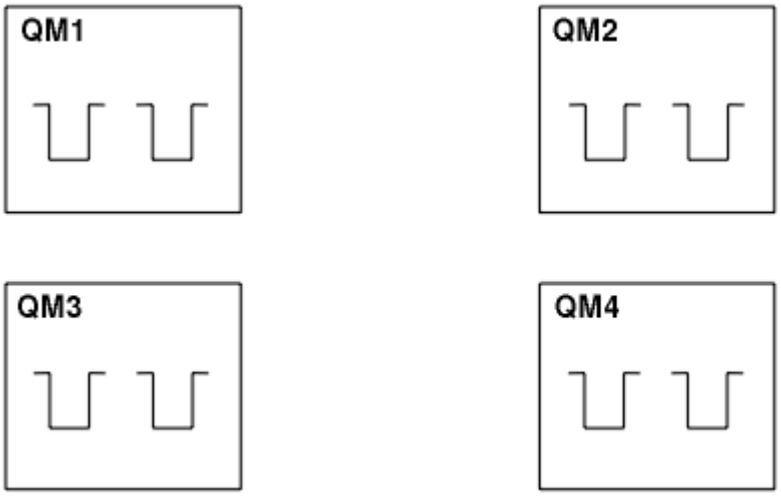


Figure 23. A network of four queue managers

Definitions to set up a network using distributed queuing

To set up the network shown in Figure 22 on page 157 using distributed queuing, you might have the following definitions:

Table 24. Definitions for distributed queuing

Description	Number per queue manager	Total number
A sender-channel definition for a channel on which to send messages to every other queue manager	3	12
A receiver-channel definition for a channel on which to receive messages from every other queue manager	3	12
A transmission-queue definition for a transmission queue to every other queue manager	3	12
A local-queue definition for each local queue	2	8
A remote-queue definition for each remote queue to which this queue manager wants to put messages	6	24

You might reduce this number of definitions by using generic receiver-channel definitions. The maximum number of definitions could be as many as 17 on each queue manager, which is a total of 68 for this network.

Definitions to set up a network using clusters

To set up the network shown in Figure 22 on page 157 using clusters you need the following definitions:

Table 25. Definitions for clustering

Description	Number per queue manager	Total number
A cluster-sender channel definition for a channel on which to send messages to a repository queue manager	1	4

Table 25. Definitions for clustering (continued)		
Description	**Number per queue manager**	**Total number**
A cluster-receiver channel definition for a channel on which to receive messages from other queue managers in the cluster	1	4
A local-queue definition for each local queue	2	8

To set up this cluster of queue managers (with two full repositories), you need four definitions on each queue manager, a total of sixteen definitions altogether. You also need to alter the queue-manager definitions for two of the queue managers, to make them full repository queue managers for the cluster.

Only one CLUSSDR and one CLUSRCVR channel definition is required. When the cluster is defined, you can add or remove queue managers (other than the repository queue managers) without any disruption to the other queue managers.

Using a cluster reduces the number of definitions required to set up a network containing many queue managers.

With fewer definitions to make there is less risk of error:

- Object names always match, for example the channel name in a sender-receiver pair.
- The transmission queue name specified in a channel definition always matches the correct transmission queue definition or the transmission queue name specified in a remote queue definition.
- A QREMOTE definition always points to the correct queue at the remote queue manager.

Once a cluster is set up, you can move cluster queues from one queue manager to another within the cluster without having to do any system management work on any other queue manager. There is no chance of forgetting to delete or modify channel, remote-queue, or transmission-queue definitions. You can add new queue managers to a cluster without any disruption to the existing network.

Components of a cluster

Clusters are composed of queue managers, cluster repositories, cluster channels, and cluster queues.

See the following subtopics for information about each of the cluster components:

Related concepts

Clusters

“Comparison of clustering and distributed queuing” on page 157
Compare the components that need to be defined to connect queue managers using distributed queuing and clustering.

“Managing IBM WebSphere MQ clusters” on page 180
You can create, extend, and maintain IBM WebSphere MQ clusters.

Related tasks

“Configuring a queue manager cluster” on page 154
Use the links in this topic to find out how clusters work, how to design a cluster configuration, and to get an example of how to set up a simple cluster.

“Setting up a new cluster” on page 181

Follow these instructions to set up the example cluster. Separate instructions describe setting up the cluster on TCP/IP, LU 6.2, and with a single transmission queue or multiple transmission queues. Test the cluster works by sending a message from one queue manager to the other.

Cluster repository

A repository is a collection of information about the queue managers that are members of a cluster.

The repository information includes queue manager names, their locations, their channels, which queues they host, and other information. The information is stored in the form of messages on a queue called SYSTEM.CLUSTER.REPOSITORY.QUEUE. The queue is one of the default objects. It is defined when you create a WebSphere MQ queue manager, except on WebSphere MQ for z/OS.

Typically, two queue managers in a cluster hold a full repository. The remaining queue managers all hold a partial repository.

Full repository and partial repository

A queue manager that hosts a complete set of information about every queue manager in the cluster has a full repository. Other queue managers in the cluster have partial repositories containing a subset of the information in the full repositories.

A partial repository contains information about only those queue managers with which the queue manager needs to exchange messages. The queue managers request updates to the information they need, so that if it changes, the full repository queue manager sends them the new information. For much of the time, a partial repository contains all the information a queue manager needs to perform within the cluster. When a queue manager needs some additional information, it makes inquiries of the full repository and updates its partial repository. The queue managers use a queue called SYSTEM.CLUSTER.COMMAND.QUEUE to request and receive updates to the repositories. This queue is one of the default objects.

Cluster queue manager

A cluster queue manager is a queue manager that is a member of a cluster.

A queue manager can be a member of more than one cluster. Each cluster queue manager must have a name that is unique throughout all the clusters of which it is a member.

A cluster queue manager can host queues, which it advertises to the other queue managers in the cluster. A cluster queue manager does not have to host or advertise any queues. It can feed messages into the cluster and receive only responses that are directed explicitly to it, and not to advertised queues.

In WebSphere MQ for z/OS, a cluster queue manager can be a member of a queue-sharing group. In this case, it shares its queue definitions with other queue managers in the same queue-sharing group.

Cluster queue managers are autonomous. They have full control over queues and channels that they define. Their definitions cannot be modified by other queue managers (other than queue managers in the same queue-sharing group). Repository queue managers do not control the definitions in other queue managers in the cluster. They hold a complete set of all the definitions, for use when required. A cluster is a federation of queue managers.

After you create or alter a definition on a cluster queue manager, the information is sent to the full repository queue manager. Other repositories in the cluster are updated later.

Full Repository queue manager

A full repository queue manager is a cluster queue manager that holds a full representation of the cluster's resources. To ensure availability, set up two or more full repository queue managers in each cluster. Full repository queue managers receive information sent by the other queue managers in the cluster and update their repositories. They send messages to each other to be sure that they are both kept up to date with new information about the cluster.

Queue managers and repositories

Every cluster has at least one (preferably two) queue managers holding full repositories of information about the queue managers, queues, and channels in a cluster. These repositories also contain requests from the other queue managers in the cluster for updates to the information.

The other queue managers each hold a partial repository, containing information about the subset of queues and queue managers with which they need to communicate. The queue managers build up their partial repositories by making inquiries when they first need to access another queue or queue manager. They request that they are notified of any new information concerning that queue or queue manager.

Each queue manager stores its repository information in messages on a queue called SYSTEM.CLUSTER.REPOSITORY.QUEUE. The queue managers exchange repository information in messages on a queue called SYSTEM.CLUSTER.COMMAND.QUEUE.

Each queue manager that joins a cluster defines a cluster-sender, CLUSSDR, channel to one of the repositories. It immediately learns which other queue managers in the cluster hold full repositories. From then on, the queue manager can request information from any of the repositories. When the queue manager sends information to the chosen repository, it also sends information to one other repository (if there is one).

A full repository is updated when the queue manager hosting it receives new information from one of the queue managers that is linked to it. The new information is also sent to another repository, to reduce the risk of it being delayed if a repository queue manager is out of service. Because all the information is sent twice, the repositories have to discard duplicates. Each item of information carries a sequence number, which the repositories use to identify duplicates. All repositories are kept in step with each other by exchanging messages.

Cluster queues

A cluster queue is a queue that is hosted by a cluster queue manager and made available to other queue managers in the cluster.

Define a cluster queue as a local queue on the cluster queue manager where the queue is hosted. Specify the name of the cluster the queue belongs to. The following example shows a **runmqsc** command to define a cluster queue with the CLUSTER option:

```
DEFINE QLOCAL(Q1) CLUSTER(SALES)
```

A cluster queue definition is advertised to other queue managers in the cluster. The other queue managers in the cluster can put messages to a cluster queue without needing a corresponding remote-queue definition. A cluster queue can be advertised in more than one cluster by using a cluster namelist.

When a queue is advertised, any queue manager in the cluster can put messages to it. To put a message, the queue manager must find out, from the full repositories, where the queue is hosted. Then it adds some routing information to the message and puts the message on a cluster transmission queue.

A cluster queue can be a queue that is shared by members of a queue-sharing group in IBM WebSphere MQ for z/OS.

Binding

You can create a cluster in which more than one queue manager hosts an instance of the same cluster queue. Make sure that all the messages in a sequence are sent to the same instance of the queue. You can bind a series of messages to a particular queue by using the MQOO_BIND_ON_OPEN option on the MQOPEN call.

Cluster transmission queues

Except on z/OS, a queue manager can store messages for other queue managers in a cluster on multiple transmission queues. You can configure a queue manager to store messages on multiple cluster transmission queues in two different ways. If you set the queue manager

attribute DEFCLXQ to CHANNEL, a different cluster transmission queue is created automatically from SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE for each cluster-sender channel. If you set the CLCHNAME transmission queue option to match one or more cluster-senders channel, the queue manager can store messages for the matching channels on that transmission queue.

Attention: If you are using dedicated SYSTEM.CLUSTER.TRANSMIT.QUEUES with a queue manager that was upgraded from an earlier version of the product, ensure that the SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE has the SHARE/NOSHARE option set to **SHARE**.

A message for a cluster queue on a different queue manager is placed upon a cluster transmission queue before being sent. A cluster-sender channel transfers the messages from a cluster transmission queue to cluster-receiver channels on other queue managers. By default, one system defined cluster transmission queue holds all the messages that are to be transferred to other cluster queue managers. The queue is called SYSTEM.CLUSTER.TRANSMIT.QUEUE. A queue manager that is part of a cluster can send messages on this cluster transmission queue to any other queue manager in the same cluster.

A definition for the single SYSTEM.CLUSTER.TRANSMIT.QUEUE queue is created by default on every queue manager except on z/OS.

On platforms other than z/OS, you can configure a queue manager to transfer messages to other clustered queue managers using multiple transmission queues. You can define additional cluster transmission queues manually, or have the queue manager create the queues automatically.

To have the queues created automatically by the queue manager, change the queue manager attribute DEFCLXQ from SCTQ to CHANNEL. The result is the queue manager creates an individual cluster transmission queue for each cluster-sender channel that is created. The transmission queues are created as permanent dynamic queues from the model queue, SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE. The name of each permanent dynamic queue is SYSTEM.CLUSTER.TRANSMIT.*ChannelName*. The name of the cluster-sender channel that each permanent dynamic cluster transmit queue is associated with is set in the local transmission queue attribute CLCHNAME. Messages for remote clustered queue managers are placed on the permanent dynamic cluster transmission queue for the associated cluster-sender channel, rather than on SYSTEM.CLUSTER.TRANSMIT.QUEUE.

To create the cluster transmission queues manually, create a local queue with the USAGE attribute set to XMITQ, and the CLCHNAME attribute set to a generic channel name that resolves to one or more cluster-sender channels; see ClusterChannelName. If you create cluster transmission queues manually, you have the choice of associating the transmission queue with a single cluster-sender channel, or with multiple cluster-sender channels. The CLCHNAME attribute is a generic-name, which means you can place multiple wildcard characters, "*", in the name.

Except for the initial cluster-sender channels that you create manually to connect a queue manager to a full repository, cluster-sender channels are created automatically. They are created automatically when there is a message to transfer to a cluster queue manager. They are created with the same name as the name of the cluster-receiver channel that receives cluster messages for that particular cluster on the destination queue manager.

If you follow a naming convention for cluster-receiver channels, it is possible to define a generic value for CLCHNAME that filters different kinds of cluster messages to different transmission queues. For example, if you follow the naming convention for cluster-receiver channels of *ClusterName*.*QmgrName*, then the generic name *ClusterName*.* filters messages for different clusters onto different transmission queues. You must define the transmission queues manually, and set CLCHNAME in each transmission queue to *ClusterName*.*.

Changes to the association of cluster transmission queues to cluster-sender channels do not take immediate effect. The currently associated transmission queue that a cluster-sender channel is servicing might contain messages that are in the process of being transferred by the cluster-sender channel. Only when no messages on the currently associated transmission queue are being processed by a cluster-sender channel, can the queue manager change the association of the cluster-sender channel a different transmission queue. This can occur either when no messages remain on the transmission queue to be processed by the cluster-sender channel, or when processing of messages is suspended and the cluster-sender channel has no "in-flight" messages. When this happens, any unprocessed messages

for the cluster-sender channel are transferred to the newly associated transmission queue, and the association of the cluster-sender channel changes.

You can create a remote queue definition that resolves to a cluster transmission queue. In the definition, queue manager QMX is in the same cluster as the local queue manager, and there is no transmission queue, QMX.

```
DEFINE QREMOTE(A) RNAME(B) RQMNAME(QMX)
```

During queue name resolution, the cluster transmission queue takes precedence over the default transmission queue. A message put to A is stored on the cluster transmission queue and then sent to the remote queue B on QMX.

Queue managers can also communicate with other queue managers that are not part of a cluster. You must define channels and a transmission queue to the other queue manager, in the same way as in a distributed-queuing environment.

Note: Applications must write to queues that resolve to the cluster transmission queue, and must not write directly to the cluster transmission queue.

Auto definition of remote queues

A queue manager in a cluster does not need a remote-queue definition for remote queues in the cluster. The cluster queue manager finds the location of a remote queue from the full repository. It adds routing information to the message and puts it on the cluster transmission queue. WebSphere MQ automatically creates a definition equivalent to a remote-queue definition so that the message can be sent.

You cannot alter or delete an automatically created remote-queue definition. However, by using the DISPLAY QUEUE **runmqsc** command with the CLUSINFO attribute, you can view all of the local queues on a queue manager as well as all of the cluster queues, including cluster queues on remote queue managers. For example:

```
DISPLAY QUEUE(*) CLUSINFO
```

Related reference

ClusterChannelName (MQCHAR20)

Cluster channels

You must define cluster-receiver and cluster-sender channels for queue managers in your cluster. Special considerations apply to full repositories.

Within clusters, messages are distributed between cluster queue managers on a special type of channel for which you need cluster-receiver channel definitions and cluster-sender channel definitions.

Cluster-sender channel: CLUSSDR

Manually define a cluster-sender channel to a full repository at each queue manager in the cluster. The cluster-sender definition enables the queue manager to make its initial contact with the cluster. It names the full repository queue manager to which the queue manager preferentially chooses to send cluster information. The cluster-sender channel is used to notify the repository of any changes to the status of the queue manager. For example, if a queue is added or removed. It is also used to transmit messages.

The full repository queue managers themselves have cluster-sender channels that point to each other. They use them to communicate cluster status changes to each other.

It is of little importance which full repository a CLUSSDR channel definition points to. Once the initial contact has been made, further cluster queue manager objects are defined automatically as necessary. The queue manager can send cluster information to every full repository, and messages to every queue manager.

The CLUSSDR definitions made on the full repository queue managers are special. All the updates exchanged by the full repositories flow exclusively on these channels. The administrator controls the

network of full repositories explicitly. The administrator must define a CLUSSDR channel from every full repository queue manager to every other full repository queue manager in the cluster. The administrator must make the CLUSSDR definitions on full repository queue managers manually and not leave them to be auto-defined.

Cluster sender channels must only be defined to connect a partial repository to a full repository, or to connect two full repositories together. Manually configuring a CLUSSDR channel that addresses a partial repository, or a queue manager not in the cluster, leads to error messages, such as AMQ9427 and AMQ9428 being issued.

Although this might sometimes be unavoidable as a temporary situation (for example when modifying the location of a full repository) the incorrect definitions should be deleted as soon as possible to stop these errors being issued.

Cluster-receiver channel: CLUSRCVR

A cluster-receiver channel definition defines the end of a channel on which a cluster queue manager can receive messages from other queue managers in the cluster.

A cluster-receiver channel can also carry information about the cluster-information destined for the local repository. By defining the cluster-receiver channel, the queue manager shows the other cluster queue managers that it is available to receive messages. You need at least one cluster-receiver channel for each cluster queue manager.

A CLUSRCVR definition enables other queue managers to auto-define corresponding CLUSSDR channel definitions.

Related concepts

“Auto-definition of cluster channels” on page 164

A queue manager must have a definition for a cluster-sender channel before it can send a message to a remote destination. After you introduce a queue manager to a cluster by making its initial CLUSSDR and CLUSRCVR definitions, WebSphere MQ automatically makes cluster-sender channel definitions when they are needed. You cannot modify auto-defined cluster-sender channels. You can modify their behavior using a channel auto-definition exit.

Auto-definition of cluster channels

A queue manager must have a definition for a cluster-sender channel before it can send a message to a remote destination. After you introduce a queue manager to a cluster by making its initial CLUSSDR and CLUSRCVR definitions, WebSphere MQ automatically makes cluster-sender channel definitions when they are needed. You cannot modify auto-defined cluster-sender channels. You can modify their behavior using a channel auto-definition exit.

When both the cluster-sender end and the cluster-receiver end of a channel are defined, the channel is started. An auto-defined channel remains active until it is no longer needed and is shut down using the normal disconnect-interval rules.

Auto-defined cluster-sender channels take their attributes from the corresponding cluster-receiver channel definition on the receiving queue manager. Even if there is a manually defined cluster-sender channel, its attributes are automatically modified to ensure that they match the corresponding cluster-receiver definition. Suppose, for example that you define a CLUSRCVR without specifying a port number in the CONNAME parameter, and manually define a CLUSSDR that does specify a port number. When the auto-defined CLUSSDR replaces the manually defined one, the port number (taken from the CLUSRCVR) becomes blank. The default port number is used and the channel fails.

You cannot modify an auto-defined cluster-sender definition.

You cannot see automatically defined channels using the DISPLAY CHANNEL **runmqsc** command. To see the auto-defined channels use the command:

```
DISPLAY CLUSQMGR(qMgrName)
```

To display the status of the auto-defined CLUSSDR channel corresponding to the CLUSRCVR channel definition you created, use the command:

```
DISPLAY CHSTATUS(channelname)
```

You can use the WebSphere MQ channel auto-definition exit if you want to write a user exit program to customize a cluster-sender channel or cluster-receiver channel. You might use the channel auto-definition exit in a cluster environment to:

- Tailor communications definitions, that is, SNA LU6.2 names
- Add or remove other exits, for example, security exits
- Change the names of channel exits. You must change the name of a CLUSSDR channel exit because the CLUSSDR channel exit name is auto-generated from the CLUSRCVR channel definition. The auto-generated name might be wrong, and almost certainly is wrong if the two ends of the channel are on different platforms. The format of exit names is different on different platforms. For example, on Windows it is, SCYEXIT('*drive*:*path**library*(secexit)').

  Exit names on platforms other than z/OS are of the general form *path*/*library*(*function*). If *function* is present, up to eight chars of that are used. Otherwise, *library*, truncated to eight chars is used. For example,

  - /var/mqm/exits/myExit.so(MsgExit) converts to MSGEXIT
  - /var/mqm/exits/myExit converts to MYEXIT
  - /var/mqm/exits/myExit.so(ExitLongName) converts to EXITLONG

To enable an outbound (TCP) channel to use a particular IP address, port or port range, use the channel attribute LOCLADDR. LOCLADDR is useful if you have more than one network card and you want a channel to use a specific one for outbound communications.

To specify a virtual IP address on CLUSSDR channels, use the IP address from the LOCLADDR on a manually defined CLUSSDR. To specify the port range, use the port range from the CLUSRCVR.

If a cluster needs to use LOCLADDR to get the outbound communication channels to bind to a specific IP address, you must write a Channel Auto-Definition Exit in order to force the LOCLADDR value into any of their automatically defined CLUSSDR channels, and you must specify it in the manually defined CLUSSDR channel.

Do not put an IP address in the LOCLADDR field of a CLUSRCVR channel, unless all queue managers are on the same server. The LOCLADDR IP address is propagated to the auto-defined CLUSSDR channels of all queue managers that connect using the CLUSRCVR channel.

Put a port number or port range in the LOCLADDR of a CLUSRCVR channel, if you want all the queue managers in a cluster to use a specific port or range of ports, for all their outbound communications

distributed On distributed platforms, it is possible to set a default local address value that will be used for all sender channels that do not have a local address defined. The default value is defined by setting the MQ_LCLADDR environment variable prior to starting the queue manager. The format of the value matches that of MQSC attribute LOCLADDR.

Auto-defined cluster-sender channel definitions are not real channel objects. On platforms other than z/OS, the OAM (object authority manager) is not aware of their existence. If you try to issue start, stop, ping, reset, or resolve commands on auto-defined cluster-sender channels, the OAM checks to see whether you are authorized to perform the same action on the cluster-receiver channel for the cluster.

If the cluster needs to use PROPCTL to remove application headers such as RFH2 from messages going from a WebSphere MQ Version 7 queue manager to a queue manager on an earlier level of WebSphere MQ, you must write a Channel Auto-Definition Exit that forces PROPCTL to a value of NONE. The exit is necessary because cluster-sender channels have their definition based on the corresponding cluster-receiver channels. As the earlier level cluster-receiver channel does not have a PROPCTL attribute, the attribute is set to COMPAT by the auto cluster-sender channel. The attribute is set to COMPAT regardless of what is set on the manual cluster-sender channel.

Related reference
Local Address (LOCLADDR)

Default cluster objects

Create the default cluster objects when using WebSphere MQ clusters. They are included in the set of default objects automatically created when you define a queue manager.

You can alter the default channel definitions in the same way as any other channel definition, by running MQSC or PCF commands.

Do not alter the default queue definitions, except for SYSTEM.CLUSTER.HISTORY.QUEUE.

SYSTEM.CLUSTER.COMMAND.QUEUE
Each queue manager in a cluster has a local queue called SYSTEM.CLUSTER.COMMAND.QUEUE which is used to transfer messages to the full repository. The message contains any new or changed information about the queue manager, or any requests for information about other queue managers. SYSTEM.CLUSTER.COMMAND.QUEUE is normally empty.

SYSTEM.CLUSTER.HISTORY.QUEUE
Each queue manager in a cluster has a local queue called SYSTEM.CLUSTER.HISTORY.QUEUE. SYSTEM.CLUSTER.HISTORY.QUEUE is used to store the history of cluster state information for service purposes.

In the default object settings, SYSTEM.CLUSTER.HISTORY.QUEUE is set to PUT(ENABLED). To suppress history collection change the setting to PUT(DISABLED).

SYSTEM.CLUSTER.REPOSITORY.QUEUE
Each queue manager in a cluster has a local queue called SYSTEM.CLUSTER.REPOSITORY.QUEUE. This queue is used to store all the full repository information. This queue is not normally empty.

SYSTEM.CLUSTER.TRANSMIT.QUEUE
Each queue manager has a definition for a local queue called SYSTEM.CLUSTER.TRANSMIT.QUEUE. SYSTEM.CLUSTER.TRANSMIT.QUEUE is the default transmission queue for all messages to all queues and queue managers that are within clusters. You can change the default transmission queue for each cluster-sender channel to SYSTEM.CLUSTER.TRANSMIT.*ChannelName*, by changing the queue manager attribute DEFXMITQ. You cannot delete SYSTEM.CLUSTER.TRANSMIT.QUEUE. It is also used to define authorization checks whether the default transmission queue that is used is SYSTEM.CLUSTER.TRANSMIT.QUEUE or SYSTEM.CLUSTER.TRANSMIT.*ChannelName*.

SYSTEM.DEF.CLUSRCVR
Each cluster has a default CLUSRCVR channel definition called SYSTEM.DEF.CLUSRCVR. SYSTEM.DEF.CLUSRCVR is used to supply default values for any attributes that you do not specify when you create a cluster-receiver channel on a queue manager in the cluster.

SYSTEM.DEF.CLUSSDR
Each cluster has a default CLUSSDR channel definition called SYSTEM.DEF.CLUSSDR. SYSTEM.DEF.CLUSSDR is used to supply default values for any attributes that you do not specify when you create a cluster-sender channel on a queue manager in the cluster.

Cluster transmission queues and cluster-sender channels

Messages between clustered queue managers are stored on cluster transmission queues and forwarded by cluster-sender channels.

When you display cluster-sender channel you see that it is associated with a transmission queue. At any point in time, a cluster-sender channel is associated with one transmission queue. If you change the configuration of the channel, it might switch to a different transmission queue next time it starts. Run the following MQSC command to display the transmission queues that cluster-sender channels are associated with:

```
DISPLAY CHSTATUS(*) WHERE(CHLTYPE EQ CLUSSDR)
```

```
AMQ8417: Display Channel Status details.
   CHANNEL(TO.QM2)                     CHLTYPE(CLUSSDR)
```

```
    CONNAME(9.146.163.190(1416))        CURRENT
    RQMNAME(QM2)                        STATUS(STOPPED)
    SUBSTATE( )                         XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)
```

The transmission queue shown in the saved channel status of a stopped cluster-sender channel might change when the channel starts again. “Selection of default transmission queues by cluster-sender channels” on page 167 describes the process of selecting a default transmission queue; “Selection of manually defined transmission queues by cluster-sender channels” on page 168 describes the process of selecting a manually defined transmission queue.

When any cluster-sender channel starts it rechecks its association with transmission queues. If the configuration of transmission queues, or the queue manager defaults, changes, it might reassociate the channel with a different transmission queue. If the channel restarts with a different transmission queue as a result of a configuration change, a process of transferring messages to the newly associated transmission queue takes place. “How the process to switch cluster-sender channel to a different transmission queue works” on page 169 describes the process of transferring a cluster-sender channel from one transmission queue to another.

The behavior of cluster-sender channels is different to sender and server channels. They remain associated with the same transmission queue until the channel attribute **XMITQ** is altered. If you alter the transmission queue attribute on a sender or server channel, and restart it, messages are not transferred from the old transmission queue to the new one.

Another difference between cluster-sender channels, and sender or server channels, is that multiple cluster-sender channels can open a cluster transmission queue, but only one sender or server channel can open a normal transmission queue. Until Version 7.5 cluster connections shared the single cluster transmission queue, SYSTEM.CLUSTER.TRANSMIT.QUEUE. From Version 7.5 onwards you have the option of cluster-sender channels not sharing transmission queues. Exclusivity is not enforced; it is an outcome of the configuration. You can configure the path a message takes in a cluster so that it does not share any transmission queues or channels with messages that flow between other applications. See “Clustering: Planning how to configure cluster transmission queues” on page 281 and “Adding a cluster and a cluster transmit queue to isolate cluster message traffic sent from a gateway queue manager” on page 201.

Selection of default transmission queues by cluster-sender channels

A cluster transmission queue is either a system default queue, with a name that starts with SYSTEM.CLUSTER.TRANSMIT, or a manually defined queue. A cluster-sender channel is associated with a cluster transmission queue in one of two ways: by the default cluster transmission queue mechanism, or by manual configuration.

The default cluster transmission queue is set as a queue manager attribute, **DEFCLXQ**. Its value is either SCTQ or CHANNEL. New and migrated queue managers are set to SCTQ. You can alter the value to CHANNEL.

If SCTQ is set, the default cluster transmission queue is SYSTEM.CLUSTER.TRANSMIT.QUEUE. Every cluster-sender channel can open this queue. The cluster-sender channels that do open the queue are the ones that are not associated with manually defined cluster transmission queues.

If CHANNEL is set, then the queue manager can create a separate permanent dynamic transmission queue for every cluster-sender channel. Each queue is named SYSTEM.CLUSTER.TRANSMIT.*ChannelName* and is created from the model queue, SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE. Each cluster-sender channel that is not associated with a manually defined cluster transmission queue is associated with a permanent-dynamic cluster transmission queue. The queue is created by the queue manager when it requires a separate cluster transmission queue for the cluster destination served by this cluster-sender channel, and no queue exists.

Some cluster destinations can be served by cluster-sender channels associated with manually defined transmission queues, and others by the default queue or queues. In the association of cluster-sender channels with transmission queues, the manually defined transmission queues always take precedence over the default transmission queues.

The precedence of cluster transmission queues is illustrated in Figure 24 on page 168. The only cluster-sender channel not associated with a manually defined cluster transmission queue is CS.QM1. It is not associated with a manually defined transmission queue, because none of the channel names in the **CLCHNAME** attribute of the transmission queues match CS.QM1.

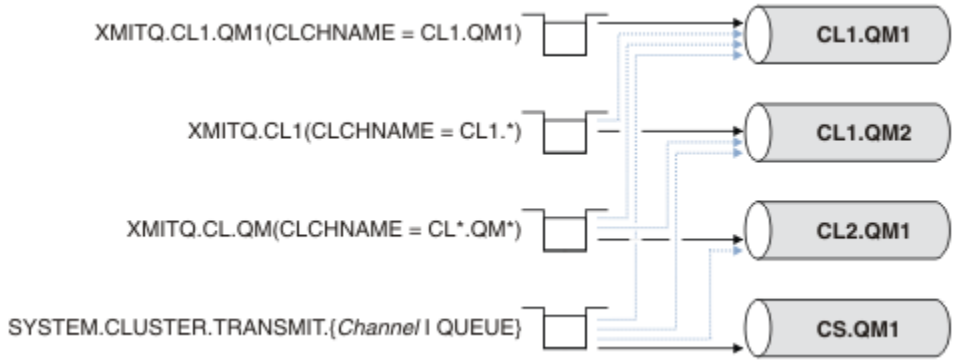


Figure 24. Transmission queue / cluster-sender channel precedence

Selection of manually defined transmission queues by cluster-sender channels

A manually defined queue has the transmission queue attribute **USAGE** attribute set to XMITQ, and the cluster channel name attribute **CLCHNAME** set to a specific or generic channel name.

If the name in the **CLCHNAME** queue attribute matches a cluster-sender channel name, the channel is associated with the queue. The name is either an exact match, if the name contains no wildcards, or it the best match, if the name contains wildcards.

If **CLCHNAME** definitions on multiple transmission queues match the same cluster-sender channel, the definitions are said to overlap. To resolve the ambiguity there is an order of precedence between matches. Exact matches always take precedence. Figure 24 on page 168 shows associations between transmission queues and cluster-sender channels. The black arrows show actual associations, and the gray arrows, potential associations. The precedence order of transmission queues in Figure 24 on page 168 is,

XMITQ.CL1.QM1
: The transmission queue XMITQ.CL1.QM1 has its **CLCHNAME** attribute set to CL1.QM1. The definition of the **CLCHNAME** attribute, CL1.QM1, has no wildcards, and takes precedence over any other CLCHNAME attributes, defined on other transmission queues, that match with wildcards. The queue manager stores any cluster message that is to be transferred by the CL1.QM1 cluster-sender channel on the XMITQ.CL1.QM1 transmission queue. The only exception is if multiple transmission queues have their **CLCHNAME** attribute set to CL1.QM1. In that case, the queue manager stores messages for the CL1.QM1 cluster-sender channel on any of those queues. It selects a queue arbitrarily when the channel starts. It might select a different queue when the channel starts again.

XMITQ.CL1
: The transmission queue XMITQ.CL1 has its **CLCHNAME** attribute set to CL1.*. The definition of the **CLCHNAME** attribute, CL1.*, has one trailing wildcard, which matches the name of any cluster-sender channel that starts with CL1.. The queue manager stores any cluster message that is to be transferred by any cluster-sender channel whose name begins with CL1. on the transmission queue XMITQ.CL1, unless there is a transmission queue with a more specific match, such as the queue XMITQ.CL1.QM1. One trailing wildcard makes the definition less specific than a definition with no wildcards, and more specific than a definition with multiple wildcards, or wildcards that are followed by more trailing characters.

XMITQ.CL.QM
: XMITQ.CL.QM is the name of the transmission queue with its **CLCHNAME** attribute set to CL*.QM*. The definition of CL*.QM* has two wildcards, which match the name of any cluster-sender channel that starts with CL., and either includes or ends with QM. The match is less specific than a match with one wildcard.

SYSTEM.CLUSTER.TRANSMIT.*channelName*|QUEUE
If no transmission queue has a **CLCHNAME** attribute that matches the name of the cluster-sender channel that the queue manager is to use, then the queue manager uses the default cluster transmission queue. The default cluster transmission queue is either the single system cluster transmission queue, SYSTEM.CLUSTER.TRANSMIT.QUEUE, or a system cluster transmission queue that the queue manager created for a specific cluster-sender channel, SYSTEM.CLUSTER.TRANSMIT.*channelName*. Which queue is the default depends on the setting of the queue manager **DEFXMITQ** attribute.

Tip: Unless you have a clear need for overlapping definitions, avoid them as they can lead to complicated configurations that are hard to understand.

How the process to switch cluster-sender channel to a different transmission queue works

To change the association of cluster-sender channels with cluster transmission queues, change the **CLCHNAME** parameter of any transmission queue or the queue manager parameter **DEFCLXQ** at any time. Nothing happens immediately. Changes occur only when a channel starts. When it starts, it checks whether to continue forwarding messages from the same transmission queue. Three kinds of change alter the association of a cluster-sender channel with a transmission queue.

1. Redefining the **CLCHNAME** parameter of the transmission queue the cluster-sender channel is currently associated with to be less specific or blank, or deleting the cluster transmission queue when the channel is stopped.

   Some other cluster transmission queue might now be a better match for the channel name. Or, if no other transmission queues match the name of the cluster-sender channel, the association must revert to the default transmission queue.
2. Redefining the **CLCHNAME** parameter of any other cluster transmission queue, or adding a cluster transmission queue.

   The **CLCHNAME** parameter of another transmission queue might now be a better match for the cluster-sender channel than the transmission queue the cluster-sender channel is currently associated with. If the cluster-sender channel is currently associated with a default cluster transmission queue, it might become associated with a manually defined cluster transmission queue.
3. If the cluster-sender channel is currently associated with a default cluster transmission queue, changing the **DEFCLXQ** queue manager parameter.

If the association of a cluster-sender channel changes, when the channel starts it switches its association to the new transmission queue. During the switch, it ensures that no messages are lost. Messages are transferred to the new transmission queue in the order in which the channel would transfer the messages to the remote queue manager.

Remember: In common with any forwarding of messages in a cluster, you must put messages into groups to ensure that messages that must be delivered in order are delivered in order. On rare occasions, messages can get out of order in a cluster.

The switch process goes through the following transactional steps. If the switch process is interrupted, the current transactional step is resumed when the channel restarts again.

Step 1 - Process messages from the original transmission queue
The cluster-sender channel is associated with the new transmission queue, which it might share with other cluster-sender channels. Messages for the cluster-sender channel continue to be placed on the original transmission queue. A transitional switch process transfers messages from the original transmission queue onto the new transmission queue. The cluster-sender channel forwards the messages from the new transmission queue to the cluster-receiver channel. The channel status shows the cluster-sender channel still associated with the old transmission queue.

The switch process continues to transfer newly arrived messages as well. This step continues until the number of remaining messages to be forwarded by the switch process reaches zero. When the number of messages reaches zero, the procedure moves onto step 2.

During step 1, disk activity for the channel increases. Persistent messages are committed off the first transmission queue and onto the second transmission queue. This disk activity is in addition to messages being committed when they are placed on and removed from the transmission queue as part of transferring the messages normally. Ideally, no messages arrive during the switching process, so the transition can take place as quickly as possible. If messages do arrive, they are processed by the switch process.

Step 2 - Process messages from the new transmission queue

As soon as no messages remain on the original transmission queue for the cluster-sender channel, new messages are placed directly on the new transmission queue. The channel status shows the cluster-sender channel is associated with the new transmission queue. The following message is written to the queue manager error log: "AMQ7341 The `transmission queue for channel` *`ChannelName`* `is` *`QueueName`*."

Multiple cluster transmission queues and cluster transmission queue attributes

You have a choice of forwarding cluster messages to different queue managers storing the messages on a single cluster transmission queue, or multiple queues. With one queue, you have one set of cluster transmission queue attributes to set and query, with multiple queues, you have multiple sets. For some attributes, having multiple sets is an advantage: for example querying queue depth tells you how many messages are waiting to be forwarded by one or a set of channels, rather than by all channels. For other attributes, having multiple sets is a disadvantage: for example, you probably do not want to configure the same access permissions for every cluster transmission queue. For this reason, access permissions are always checked against the profile for `SYSTEM.CLUSTER.TRANSMIT.QUEUE`, and not against profiles for a particular cluster transmission queue. If you want to apply more granular security checks, see “Access control and multiple cluster transmission queues” on page 156.

Multiple cluster-sender channels and multiple transmission queues

A queue manager stores a message on a cluster transmission queue before forwarding it on a cluster-sender channel. It selects a cluster-sender channel that is connected to the destination for the message. It might have a choice of cluster-sender channels that all connect to the same destination. The destination might be the same physical queue, connected by multiple cluster-sender channels to a single queue manager. The destination might also be many physical queues with the same queue name, hosted on different queue managers in the same cluster. Where is a choice of cluster-sender channels connected to a destination, the workload balancing algorithm chooses one. The choice depends on a number of factors; see The cluster workload management algorithm.

In Figure 25 on page 171, `CL1.QM1`, `CL1.QM2` and `CS.QM1` are all channels that might lead to the same destination. For example, if you define `Q1` in `CL1` on `QM1` and `QM2` then `CL1.QM1` and `CL1.QM2` both provide routes to the same destination, `Q1`, on two different queue managers. If the channel `CS.QM1` is also in `CL1`, it too is a channel that a message for `Q1` can take. The cluster membership of `CS.QM1` might be defined by a cluster namelist, which is why the channel name does not include a cluster name in its construction. Depending on the workload balancing parameters, and the sending application, some messages for `Q1` might be placed on each of the transmission queues, `XMITQ.CL1.QM1`, `XMITQ.CL1` and `SYSTEM.CLUSTER.TRANSMIT.CS.QM1`.

If you intend to separate message traffic, so that messages for the same destination do not share queues or channels with messages for different destinations, you must consider how to divide traffic onto different cluster-sender channels first, and then how to separate messages for a particular channel onto a different transmission queue. Cluster queues on the same cluster, on the same queue manager, normally share the same cluster channels. Defining multiple cluster transmission queues alone is not sufficient to separate cluster message traffic onto different queues. Unless you separate messages for different destination queues onto different channels, the messages share the same cluster transmission queue.

A straightforward way to separate the channels that messages take, is to create multiple clusters. On any queue manager in each cluster, define only one cluster queue. Then, if you define a different cluster-receiver channel for each cluster/queue manager combination, the messages for each cluster queue do not share a cluster channel with messages for other cluster queues. If you define separate transmission queues for the cluster channels, the sending queue manager stores messages for only one cluster queue on each transmission queue. For example, if you want two cluster queues not to share resources, you can either place them in different clusters on the same queue manager, or on different queue managers in the same cluster.

The choice of cluster transmission queue does not affect the workload balancing algorithm. The workload balancing algorithm chooses which cluster-sender channel to forward a message. It places the message on the transmission queue that is serviced by that channel. If the workload balancing algorithm is called on to choose again, for instance if the channel stops, it might be able to select a different channel to forward the message. If it does choose a different channel, and the new channel forwards messages from a different cluster transmission queue, the workload balancing algorithm transfers the message to the other transmission queue.

In Figure 25 on page 171, two cluster-sender channels, `CS.QM1` and `CS.QM2`, are associated with the default system transmission queue. When the workload balancing algorithm stores a message on `SYSTEM.CLUSTER.TRANSMIT.QUEUE`, or any other cluster transmission queue, the name of the cluster-sender channel that is to forward the message is stored in the correlation ID of the message. Each channel forwards just those messages that match the correlation ID with the channel name.

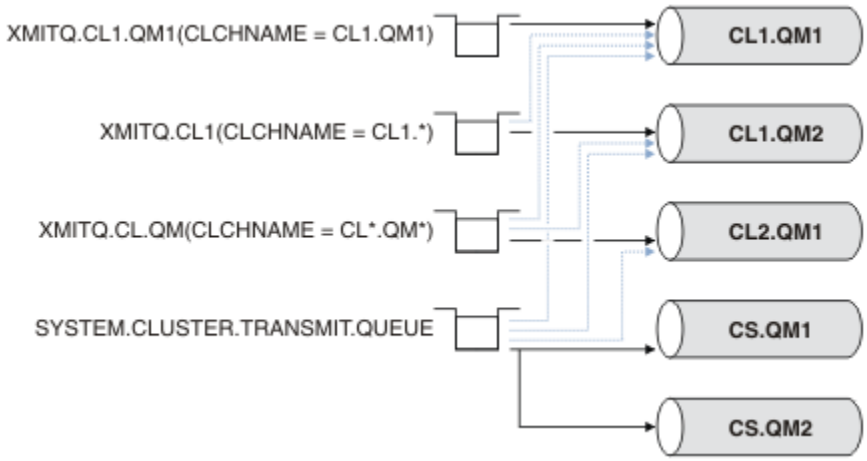


Figure 25. Multiple cluster sender channels

If `CS.QM1` stops, the messages on the transmission queue for that cluster-sender channel are examined. Those messages that can be forwarded by another channel are reprocessed by the workload balancing algorithm. Their correlation ID is reset to an alternative cluster-sender channel name. If the alternative cluster-sender channel is `CS.QM2`, the message remains on `SYSTEM.CLUSTER.TRANSMIT.QUEUE`. If the alternative channel is `CL1.QM1`, the workload balancing algorithm transfers the message to `XMITQ.CL1.QM1`. When the cluster-sender channel restarts, new messages, and messages that were not flagged for a different cluster-sender channel, are transferred by the channel again.

You might change the association between transmission queues and cluster-sender channels on a running system. You might change a `CLCHNAME` parameter on a transmission queue, or, change the **DEFCLXQ** queue manager parameter. When a channel that is affected by the change restarts, it starts the transmission queue switching process; see "How the process to switch cluster-sender channel to a different transmission queue works" on page 169.

The process to switch the transmission queue starts when the channel is restarted. The workload rebalancing process starts when the channel is stopped. The two process can run in parallel.

The simple case is when stopping a cluster-sender channel does not cause the rebalancing process to change the cluster-sender channel that is to forward any messages on the queue. This case is when no other cluster-sender channel can forward the messages to the correct destination. With no alternative

cluster-sender channel to forward the messages to their destination, the messages remain flagged for the same cluster-sender channel after the cluster-sender channel stops. When the channel starts, if a switch is pending, the switching processes moves the messages to a different transmission queue where they are processed by the same cluster-sender channel.

The more complex case is where more than one cluster-sender channel can process some messages to the same destination. You stop and restart the cluster-sender channel to trigger the transmission queue switch. In many cases, by the time you restart the channel, the workload balancing algorithm has already moved messages from the original transmission queue to different transmission queues served by different cluster-sender channels. Only those messages that cannot be forwarded by a different cluster-sender channel remain to be transferred to the new transmission queue. In some cases, if the channel is restarted quickly, some messages that could be transferred by the workload balancing algorithm remain. In which case some remaining messages are switched by the workload balancing process, and some by the process of switching the transmission queue.

Related concepts

"Clustering: Application isolation using multiple cluster transmission queues" on page 277
You can isolate the message flows between queue managers in a cluster. You can place messages being transported by different cluster-sender channels onto different cluster transmission queues. You can use the approach in a single cluster or with overlapping clusters. The topic provides examples and some best practices to guide you in choosing an approach to use.

"Calculating the size of the log" on page 392
Estimating the size of log a queue manager needs.

Related tasks

"Clustering: Planning how to configure cluster transmission queues" on page 281
You are guided through the choices of cluster transmission queues. You can configure one common default queue, separate default queues, or manually defined queues. Configuring multiple cluster transmission queues applies to platforms other than z/OS.

"Creating two-overlapping clusters with a gateway queue manager" on page 212
Follow the instructions in the task to construct overlapping clusters with a gateway queue manager. Use the clusters as a starting point for the following examples of isolating messages to one application from messages to other applications in a cluster.

"Adding a queue manager to a cluster: separate transmission queues" on page 193
Follow these instructions to add a queue manager to the cluster you created. Messages to cluster queues and topics are transferred using multiple cluster transmission queues.

"Adding a cluster transmit queue to isolate cluster message traffic sent from a gateway queue manager" on page 198
Modify the configuration of overlapping clusters that use a gateway queue manager. After the modification messages are transferred to an application from the gateway queue manager without using the same transmission queue or channels as other cluster messages. The solution uses an additional cluster transmission queue to separate message traffic to a single queue manager in a cluster.

"Adding a cluster and a cluster transmit queue to isolate cluster message traffic sent from a gateway queue manager" on page 201
Modify the configuration of overlapping clusters that use a gateway queue manager. After the modification messages are transferred to an application from the gateway queue manager without using the same transmission queue or channels as other cluster messages. The solution uses an additional cluster to isolate the messages to a particular cluster queue.

How to choose cluster queue managers to hold full repositories

In each cluster you must choose at least one, and preferably two queue managers to hold full repositories. Two full repositories are sufficient for all but the most exceptional circumstances. If possible, choose queue managers that are hosted on robust and permanently-connected platforms, that do not have coinciding outages, and that are in a central position geographically. Also consider dedicating systems as full repository hosts, and not using these systems for any other tasks.

Full repositories are queue managers that maintain a complete picture of the state of the cluster. To share this information, each full repository is connected by CLUSSDR channels (and their corresponding CLUSRCVR definitions) to every other full repository in the cluster. You must manually define these channels.

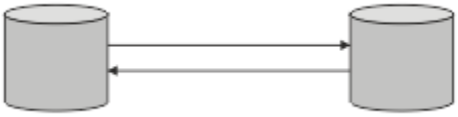


Figure 26. Two connected full repositories.

Every other queue manager in the cluster maintains a picture of what it currently knows about the state of the cluster in a *partial repository*. These queue managers publish information about themselves, and request information about other queue managers, using any two available full repositories. If a chosen full repository is not available, another is used. When the chosen full repository becomes available again, it collects the latest new and changed information from the others so that they keep in step. If all the full repositories go out of service, the other queue managers use the information they have in their partial repositories. However, they are limited to using the information that they have; new information and requests for updates cannot be processed. When the full repositories reconnect to the network, messages are exchanged to bring all repositories (both full and partial) up to date.

When planning the allocation of full repositories, include the following considerations:

- The queue managers chosen to hold full repositories need to be reliable and managed. Choose queue managers that are hosted on a robust and permanently-connected platform.
- Consider the planned outages for the systems hosting your full repositories, and ensure that they do not have coinciding outages.
- Consider network performance: Choose queue managers that are in a central position geographically, or that share the same system as other queue managers in the cluster.
- Consider whether a queue manager is a member of more than one cluster. It can be administratively convenient to use the same queue manager to host the full repositories for several clusters, provided this benefit is balanced against how busy you expect the queue manager to be.
- Consider dedicating some systems to contain only full repositories, and not using these systems for any other tasks. This ensures that these systems only require maintenance for queue manager configuration, and are not removed from service for the maintenance of other business applications. It also ensures that the task of maintaining the repository does not compete with applications for system resources. This can be particularly beneficial in large clusters (say, clusters of more than a thousand queue managers), where full repositories have a much higher workload in maintaining the cluster state.

Having more than two full repositories is possible, but rarely recommended. Although object definitions (that is, queues, topics and channels) flow to all available full repositories, requests only flow from a partial repository to a maximum of two full repositories. This means that, when more than two full repositories are defined, and any two full repositories become unavailable, some partial repositories might not receive updates they would expect. See MQ Clusters: Why only two Full Repositories?

One situation in which you might find it useful to define more than two full repositories is when migrating existing full repositories to new hardware or new queue managers. In this case, you should introduce the replacement full repositories, and confirm that they have become fully populated, before you remove the previous full repositories. Whenever you add a full repository, remember that you must directly connect it to every other full repository with CLUSSDR channels.

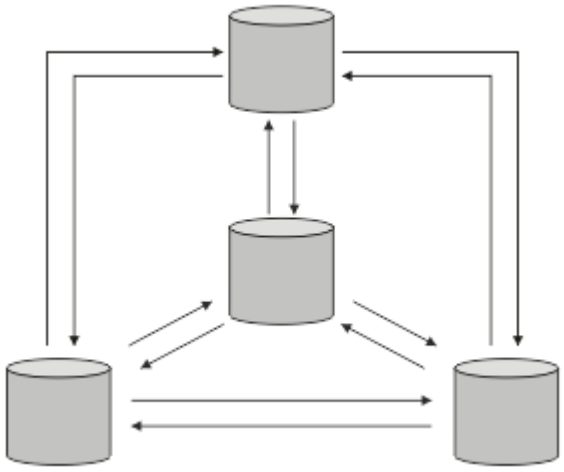


Figure 27. More than two connected full repositories

Related information

MQ Clusters: Why only two Full Repositories?

How big can an MQ Cluster be?

Organizing a cluster

Select which queue managers to link to which full repository. Consider the performance effect, the queue manager version, and whether multiple CLUSSDR channels are desirable.

Having selected the queue managers to hold full repositories, you need to decide which queue managers to link to which full repository. The CLUSSDR channel definition links a queue manager to a full repository from which it finds out about the other full repositories in the cluster. From then on, the queue manager sends messages to any two full repositories. It always tries to use the one to which it has a CLUSSDR channel definition first. You can choose to link a queue manager to either full repository. In choosing, consider the topology of your configuration, and the physical or geographical location of the queue managers.

Because all cluster information is sent to two full repositories, there might be situations in which you want to make a second CLUSSDR channel definition. You might define a second CLUSSDR channel in a cluster that has many full repositories spread over a wide area. You can then control which two full repositories your information is sent to.

Cluster naming conventions

Consider naming queue managers in the same cluster using a naming convention that identifies the cluster to which the queue manager belongs. Use a similar naming convention for channel names and extend it to describe the channel characteristics.

Best practices when naming MQ Clusters

Although cluster names can be up to 48 characters, relatively short cluster names are helpful when applying naming conventions to other objects. See “Best practices when choosing cluster channel names” on page 175.

When choosing a cluster name, it is usually helpful to represent the 'purpose' of the cluster (which is likely to be long-lived) rather than the 'content'. For example 'B2BPROD' or 'ACTTEST' rather than 'QM1_QM2_QM3_CLUS'.

Best practices when choosing cluster Queue Manager names

If you are creating a new cluster and its members from scratch, consider a naming convention for the queue managers that reflects their cluster usage. Every queue manager must have a different name. However, you can give queue managers in a cluster a set of similar names, to help in identifying and remembering logical groupings (for example 'ACTTQM1, ACTTQM2).

Relatively short queue manager names (for example less than 8 characters) help if you choose to use the convention described in the next section, or something similar, for channel names.

Best practices when choosing cluster channel names

Because queue managers and clusters can have names of up to 48 characters, and a channel name is limited to 20 characters, take care when first naming objects to avoid having to change the naming convention midway through a project (see previous section).

When defining channels, remember that automatically-created cluster-sender channels on any queue manager in the cluster take their name from the corresponding cluster-receiver channel configured on the receiving queue manager in the cluster, and these must therefore be unique and make sense *on remote queue managers in the cluster.*

One common approach is to use the queue manager name preceded by the cluster name. For example, if the cluster name is CLUSTER1 and the queue managers are QM1, QM2, then cluster-receiver channels are CLUSTER1.QM1, CLUSTER1.QM2.

You might extend this convention if channels have different priorities or use different protocols. For example:

- CLUSTER1.QM1.S1
- CLUSTER1.QM1.N3
- CLUSTER1.QM1.T4

In this example, S1 might be the first SNA channel, N3 might be the NetBIOS channel with a network priority of three, and T4 might be TCP IP using an IPV4 network.

Naming shared channel definitions

A single channel definition can be shared across multiple clusters, in which case the naming conventions suggested here would need modification. However, as described in Managing channel definitions it is usually preferable to define discrete channels for each cluster in any case.

Older channel naming conventions

Outside of cluster environments it has historically been common to use a 'FROMQM.TO.TARGETQM' naming convention, so you might find existing clusters have used something similar (such as CLUSTER.TO.TARGET). This is not recommended as part of a new cluster naming scheme because it further reduces the available characters to convey 'useful' information within your channel name.

Overlapping clusters

Overlapping clusters provide additional administrative capabilities. Use namelists to reduce the number of commands needed to administer overlapping clusters.

You can create clusters that overlap. There are a number of reasons you might define overlapping clusters; for example:

- To allow different organizations to have their own administration.
- To allow independent applications to be administered separately.
- To create classes of service.

In Figure 28 on page 176, the queue manager STF2 is a member of both the clusters. When a queue manager is a member of more than one cluster, you can take advantage of namelists to reduce the number of definitions you need. Namelists contain a list of names, for example, cluster names. You can

create a namelist naming the clusters. Specify the namelist on the ALTER QMGR command for STF2 to make it a full repository queue manager for both clusters.

If you have more than one cluster in your network, you must give them different names. If two clusters with the same name are ever merged, it is not possible to separate them again. It is also a good idea to give the clusters and channels different names. They are more easily distinguished when you look at the output from the DISPLAY commands. Queue manager names must be unique within a cluster for it to work correctly.

Defining classes of service

Imagine a university that has a queue manager for each member of staff and each student. Messages between members of staff are to travel on channels with a high priority and high bandwidth. Messages between students are to travel on cheaper, slower channels. You can set up this network using traditional distributed queuing techniques. WebSphere MQ selects which channels to use by looking at the destination queue name and queue manager name.

To clearly differentiate between the staff and students, you could group their queue managers into two clusters as shown in Figure 28 on page 176. WebSphere MQ moves messages to the meetings queue in the staff cluster only over channels that are defined in that cluster. Messages for the gossip queue in the students cluster go over channels defined in that cluster and receive the appropriate class of service.

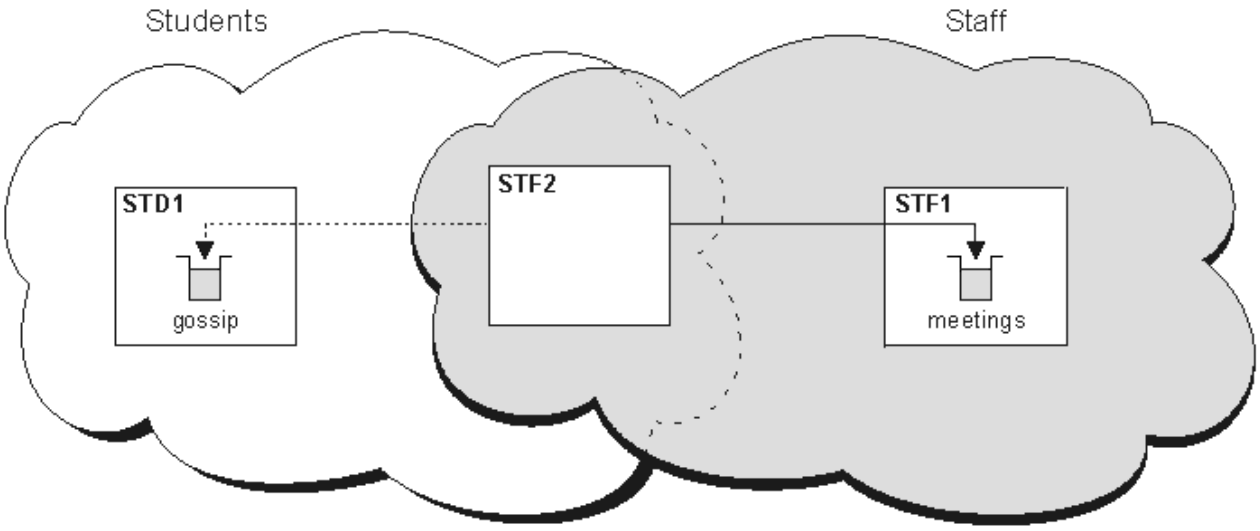


Figure 28. Classes of service

Clustering tips

You might need to make some changes to your systems or applications before using clustering. There are both similarities and differences from the behavior of distributed queuing.

- The WebSphere MQ Explorer cannot directly administer WebSphere MQ for z/OS queue managers at versions earlier than Version 6.0.
- You must add manual configuration definitions to queue managers outside a cluster for them to access cluster queues.
- If you merge two clusters with the same name, you cannot separate them again. Therefore it is advisable to give all clusters a unique name.
- If a message arrives at a queue manager but there is no queue there to receive it, the message is put on the dead-letter queue. If there is no dead-letter queue, the channel fails and tries again. The use of the dead-letter queue is the same as with distributed queuing.
- The integrity of persistent messages is maintained. Messages are not duplicated or lost as a result of using clusters.

- Using clusters reduces system administration. Clusters make it easy to connect larger networks with many more queue managers than you would be able to contemplate using distributed queuing. There is a risk that you might consume excessive network resources if you attempt to enable communication between every queue manager in a cluster.
- If you use the WebSphere MQ Explorer, which presents the queue managers in a tree structure, the view for large clusters might be cumbersome.
- WebSphere MQ Explorer can administer a cluster with repository queue managers on WebSphere MQ for z/OS Version 6 or later. You need not nominate an additional repository on a separate system. For earlier versions of WebSphere MQ on z/OS, the WebSphere MQ Explorer cannot administer a cluster with repository queue managers. You must nominate an additional repository on a system that the WebSphere MQ Explorer can administer.
- The purpose of distribution lists is to use a single MQPUT command to send the same message to multiple destinations. Distribution lists are supported on WebSphere MQ for AIX, IBM i, HP-UX, Solaris, Linux, and Windows. You can use distribution lists with queue manager clusters. In a cluster, all the messages are expanded at MQPUT time. The advantage, in terms of network traffic, is not so great as in a non-clustering environment. The advantage of distribution lists is that the numerous channels and transmission queues do not need to be defined manually.
- If you are going to use clusters to balance your workload examine your applications. See whether they require messages to be processed by a particular queue manager or in a particular sequence. Such applications are said to have message affinities. You might need to modify your applications before you can use them in complex clusters.
- You might choose to use the MQOO_BIND_ON_OPEN option on an MQOPEN to force messages to be sent to a specific destination. If the destination queue manager is not available the messages are not delivered until the queue manager becomes available again. Messages are not routed to another queue manager because of the risk of duplication.
- If a queue manager is to host a cluster repository, you need to know its host name or IP address. You have to specify this information in the CONNAME parameter when you make the CLUSSDR definition on other queue managers joining the cluster. If you use DHCP, the IP address is subject to change because DHCP can allocate a new IP address each time you restart a system. Therefore, you must not specify the IP address in the CLUSSDR definitions. Even if all the CLUSSDR definitions specify the host name rather than the IP address, the definitions would still not be reliable. DHCP does not necessarily update the DNS directory entry for the host with the new address. If you must nominate queue managers as full repositories on systems that use DHCP, install software that guarantees to keep your DNS directory up to date.
- Do not use generic names, for example VTAM generic resources or Dynamic Domain Name Server (DDNS) generic names as the connection names for your channels. If you do, your channels might connect to a different queue manager than expected.
- You can only get a message from a local cluster queue, but you can put a message to any queue in a cluster. If you open a queue to use the MQGET command, the queue manager opens the local queue.
- You do not need to alter any of your applications if you set up a simple WebSphere MQ cluster. The application can name the target queue on the MQOPEN call and does not need to know about the location of the queue manager. If you set up a cluster for workload management you must review your applications and modify them as necessary.
- You can view current monitoring and status data for a channel or queue using the DISPLAY CHSTATUS and the DISPLAY QSTATUS **runmqsc** commands. The monitoring information can be used to help gauge the performance and health of the system. Monitoring is controlled by queue manager, queue, and channel attributes. Monitoring of auto-defined cluster-sender channels is possible with the MONACLS queue manager attribute.

Related concepts

Clusters

How clusters work

“Comparison of clustering and distributed queuing” on page 157

Compare the components that need to be defined to connect queue managers using distributed queuing and clustering.

“Components of a cluster” on page 159
Clusters are composed of queue managers, cluster repositories, cluster channels, and cluster queues.

“Managing IBM WebSphere MQ clusters” on page 180
You can create, extend, and maintain IBM WebSphere MQ clusters.

Related tasks

“Configuring a queue manager cluster” on page 154
Use the links in this topic to find out how clusters work, how to design a cluster configuration, and to get an example of how to set up a simple cluster.

“Setting up a new cluster” on page 181
Follow these instructions to set up the example cluster. Separate instructions describe setting up the cluster on TCP/IP, LU 6.2, and with a single transmission queue or multiple transmission queues. Test the cluster works by sending a message from one queue manager to the other.

Establishing communication in a cluster

A channel initiator is needed to start a communication channel when there is a message to deliver. A channel listener waits to start the other end of a channel to receive the message.

Before you begin

To establish communication between queue managers in a cluster, configure a link using one of the supported communication protocols. The supported protocols are TCP or LU 6.2 on any platform, and NetBIOS or SPX on Windows systems. As part of this configuration, you also need channel initiators and channel listeners just as you do with distributed queuing.

About this task

All cluster queue managers need a channel initiator to monitor the system-defined initiation queue `SYSTEM.CHANNEL.INITQ`. `SYSTEM.CHANNEL.INITQ` is the initiation queue for all transmission queues including the cluster transmission queue.

Each queue manager must have a channel listener. A channel listener program waits for incoming network requests and starts the appropriate receiver-channel when it is needed. The implementation of channel listeners is platform-specific, however there are some common features. On all WebSphere MQ platforms, the listener can be started using the `START LISTENER` command. On WebSphere MQ for IBM i, Windows, UNIX and Linux systems, you can start the listener automatically at the same time as the queue manager. To start the listener automatically, set the `CONTROL` attribute of the `LISTENER` object to `QMGR` or `STARTONLY`.

Procedure

1. Start the channel initiator.
   - Windows UNIX Linux

     IBM WebSphere MQ for Windows, UNIX and Linux systems
     When you start a queue manager, if the queue manager attribute `SCHINIT` is set to `QMGR`, a channel initiator is automatically started. Otherwise it can be started using the **runmqsc** `START CHINIT` command or the **runmqchi** control command.
2. Start the channel listener.
   - Windows

     IBM WebSphere MQ for Windows
     Use either the channel listener program provided by WebSphere MQ, or the facilities provided by the operating system.

To start the WebSphere MQ channel listener use the RUNMQLSR command. For example:

```
RUNMQLSR -t tcp -p 1414 -m QM1
```

- 

IBM WebSphere MQ on UNIX and Linux systems

Use either the channel listener program provided by WebSphere MQ, or the facilities provided by the operating system; for example, **inetd** for TCP communications.

To start the WebSphere MQ channel listener use the **runmqlsr** command. For example:

```
runmqlsr -t tcp -p 1414 -m QM1
```

To use **inetd** to start channels, configure two files:

a. Edit the file `/etc/services`. You must be logged in as a superuser or root. If the following line is not in the file, add it as shown:

```
MQSeries       1414/tcp      # Websphere MQ channel listener
```

where 1414 is the port number required by IBM WebSphere MQ. You can change the port number, but it must match the port number specified at the sending end.

b. Edit the file `/etc/inetd.conf`. If you do not have the following line in that file, add it as shown:

```
MQSeries stream tcp nowait mqm MQ_INSTALLATION_PATH/bin/amqcrsta amqcrsta
-m queue.manager.name
```

where *MQ_INSTALLATION_PATH* is replaced by the high-level directory in which WebSphere MQ is installed.

The updates become active after **inetd** has reread the configuration files. Issue the following commands from the root user ID:

On AIX:

```
refresh -s inetd
```

On HP-UX:

```
inetd -c
```

On Solaris or Linux:

a. Find the process ID of the **inetd** with the command:

```
ps -ef | grep inetd
```

b. Run the appropriate command, as follows:

– For Solaris 9 and Linux:

```
kill -1 inetd processid
```

– For Solaris 10, or later versions:

```
inetconv
```

How long do the queue manager repositories retain information?

Queue manager repositories retain information for 30 days. An automatic process efficiently refreshes information that is being used.

When a queue manager sends out some information about itself, the full and partial repository queue managers store the information for 30 days. Information is sent out, for example, when a queue manager advertises the creation of a new queue. To prevent this information from expiring, queue managers automatically resend all information about themselves after 27 days. If a partial repository sends a new request for information part way through the 30 day lifetime, the expiry time remains the original 30 days.

When information expires, it is not immediately removed from the repository. Instead it is held for a grace period of 60 days. If no update is received within the grace period, the information is removed. The grace period allows for the fact that a queue manager might have been temporarily out of service at the expiry date. If a queue manager becomes disconnected from a cluster for more than 90 days, it stops being part of the cluster. However, if it reconnects to the network it becomes part of the cluster again. Full repositories do not use information that has expired to satisfy new requests from other queue managers.

Similarly, when a queue manager sends a request for up-to-date information from a full repository, the request lasts for 30 days. After 27 days IBM WebSphere MQ checks the request. If it has been referenced during the 27 days, it is refreshed automatically. If not, it is left to expire and is refreshed by the queue manager if it is needed again. Expiring requests prevents a buildup of requests for information from dormant queue managers.

Note: For large clusters, it can be disruptive if many queue managers automatically resend all information about themselves at the same time. See “Refreshing in a large cluster can affect performance and availability of the cluster” on page 300.

Related concepts

“Clustering: Using REFRESH CLUSTER best practices” on page 300
You use the **REFRESH CLUSTER** command to discard all locally held information about a cluster and rebuild that information from the full repositories in the cluster. You should not need to use this command, except in exceptional circumstances. If you do need to use it, there are special considerations about how you use it. This information is a guide based on testing and feedback from customers.

Managing IBM WebSphere MQ clusters

You can create, extend, and maintain IBM WebSphere MQ clusters.

For details on how to manage your IBM WebSphere MQ clusters, see the following subtopics:

Related concepts

Clusters

How clusters work

“Comparison of clustering and distributed queuing” on page 157
Compare the components that need to be defined to connect queue managers using distributed queuing and clustering.

“Components of a cluster” on page 159
Clusters are composed of queue managers, cluster repositories, cluster channels, and cluster queues.

Related tasks

“Configuring a queue manager cluster” on page 154
Use the links in this topic to find out how clusters work, how to design a cluster configuration, and to get an example of how to set up a simple cluster.

“Setting up a new cluster” on page 181

Follow these instructions to set up the example cluster. Separate instructions describe setting up the cluster on TCP/IP, LU 6.2, and with a single transmission queue or multiple transmission queues. Test the cluster works by sending a message from one queue manager to the other.

Setting up a new cluster

Follow these instructions to set up the example cluster. Separate instructions describe setting up the cluster on TCP/IP, LU 6.2, and with a single transmission queue or multiple transmission queues. Test the cluster works by sending a message from one queue manager to the other.

Before you begin

- Instead of following these instructions, you can use one of the wizards supplied with IBM WebSphere MQ Explorer to create a cluster like the one created by this task. Right-click the Queue Manager Clusters folder, then click **New** > **Queue Manager Cluster**, and follow the instructions given in the wizard.
- For background information which might aid your understanding about the steps taken to set up a cluster, see "Cluster queues" on page 161, Channels and Listeners.

About this task

You are setting up a new IBM WebSphere MQ network for a chain store. The store has two branches, one in London and one in New York. The data and applications for each store are hosted by systems running separate queue managers. The two queue managers are called LONDON and NEWYORK. The inventory application runs on the system in New York, connected to queue manager NEWYORK. The application is driven by the arrival of messages on the INVENTQ queue, hosted by NEWYORK. The two queue managers, LONDON and NEWYORK, are to be linked in a cluster called INVENTORY so that they can both put messages to the INVENTQ.

Figure 29 on page 181 shows what this cluster looks like.

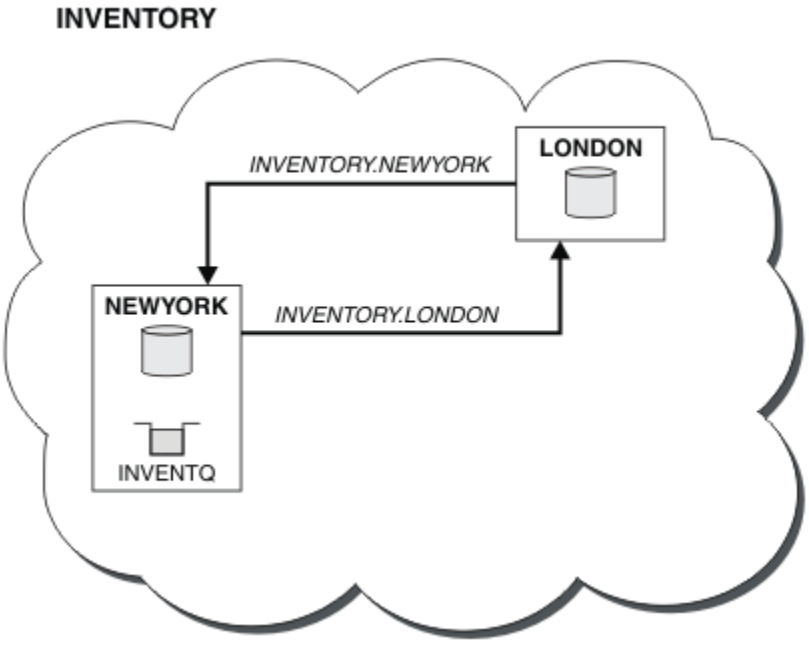


Figure 29. The INVENTORY cluster with two queue managers

You can configure each queue manager in the cluster that is not on z/OS to send messages to other queue managers in the cluster using different cluster transmission queues.

The instructions to set up the cluster vary a little by transport protocol , number of transmission queues, or platform. You have a choice of three combinations. The verification procedure remains the same for all combinations.

Procedure

- "Setting up a cluster using TCP/IP with a single transmission queue per queue manager" on page 182
- "Setting up a cluster on TCP/IP using multiple transmission queues per queue manager" on page 185
- "Setting up a cluster using LU 6.2 on z/OS" on page 187
- "Verifying the cluster" on page 189

Results

Figure 29 on page 181 shows the INVENTORY cluster setup by this task.

Clearly, INVENTORY is a small cluster. However, it is useful as a proof of concept. The important thing to understand about this cluster is the scope it offers for future enhancement.

Related concepts

Clusters

How clusters work

"Comparison of clustering and distributed queuing" on page 157
Compare the components that need to be defined to connect queue managers using distributed queuing and clustering.

"Components of a cluster" on page 159
Clusters are composed of queue managers, cluster repositories, cluster channels, and cluster queues.

"Managing IBM WebSphere MQ clusters" on page 180
You can create, extend, and maintain IBM WebSphere MQ clusters.

Related tasks

"Configuring a queue manager cluster" on page 154
Use the links in this topic to find out how clusters work, how to design a cluster configuration, and to get an example of how to set up a simple cluster.

Setting up a cluster using TCP/IP with a single transmission queue per queue manager

Before you begin

- On AIX, HP-UX, IBM i, Linux, Solaris, and Windows, the queue manager attribute, **DEFCLXQ**, must be left as its default value, SCTQ.

About this task

Follow these steps to set up a cluster on AIX, HP-UX, IBM i, Linux, Solaris, and Windows using the transport protocol TCP/IP.

Procedure

1. Decide on the organization of the cluster and its name.

   You decided to link the two queue managers, LONDON and NEWYORK, into a cluster. A cluster with only two queue managers offers only marginal benefit over a network that is to use distributed queuing. It is a good way to start and it provides scope for future expansion. When you open new branches of your store, you are able to add the new queue managers to the cluster easily. Adding new queue managers does not disrupt the existing network; see "Adding a queue manager to a cluster" on page 191.

   For the time being, the only application you are running is the inventory application. The cluster name is INVENTORY.

2. Decide which queue managers are to hold full repositories.

   In any cluster you must nominate at least one queue manager, or preferably two, to hold full repositories. In this example, there are only two queue managers, LONDON and NEWYORK, both of which hold full repositories.

   a. You can perform the remaining steps in any order.

   b. As you proceed through the steps, warning messages might be written to the queue-manager log. The messages are a result of missing definitions that you have yet to add.

   ```
   Examples of the responses to the commands are shown in a box
   like this after each step in this task.
   These examples show the responses returned by WebSphere MQ for AIX.
   The responses vary on other platforms.
   ```

   c. Before proceeding with these steps, make sure that the queue managers are started.

3. Alter the queue-manager definitions to add repository definitions.

   On each queue manager that is to hold a full repository, alter the local queue-manager definition, using the ALTER QMGR command and specifying the REPOS attribute:

   ```
   ALTER QMGR REPOS(INVENTORY)
   ```

   ```
   1 : ALTER QMGR REPOS(INVENTORY)
   AMQ8005: Websphere MQ queue manager changed.
   ```

   For example, if you enter:

   a. `runmqsc LONDON`

   b. `ALTER QMGR REPOS(INVENTORY)`

   LONDON is changed to a full repository.

4. Define the listeners.

   Define a listener that accepts network requests from other queue managers for every queue manager in the cluster. On the LONDON queue managers, issue the following command:

   ```
   DEFINE LISTENER(LONDON_LS) TRPTYPE(TCP) CONTROL(QMGR)
   ```

   Note: When you define a listener, a port number should be defined if you are using IP addresses in the CONNAME field and the port number is not the default port (1414). For example:

   ```
   DEFINE LISTENER(LONDON_LS) TRPTYPE(TCP) CONTROL(QMGR) PORT(1415)
   ```

   The CONTROL attribute ensures that the listener starts and stops when the queue manager does.

   The listener is not started when it is defined, so it must be manually started the first time with the following MQSC command:

   ```
   START LISTENER(LONDON_LS)
   ```

   Issue similar commands for all the other queue managers in the cluster, changing the listener name for each one.

   There are several ways to define these listeners, as shown in Listeners.

5. Define the CLUSRCVR channel for the LONDON queue manager.

   On every queue manager in a cluster, define a cluster-receiver channel on which the queue manager can receive messages. CLUSRCVR defines the connection name of the queue manager. The connection name is stored in the repositories, where other queue managers can refer to it. The CLUSTER keyword shows the availability of the queue manager to receive messages from other queue managers in the cluster.

In this example the channel name is INVENTORY.LONDON, and the connection name (CONNAME) is the network address of the machine the queue manager resides on, which is LONDON.CHSTORE.COM. The network address can be entered as an alphanumeric DNS host name, or an IP address in either in IPv4 dotted decimal form. For example, 192.0.2.0, or IPv6 hexadecimal form; for example 2001:DB8:0204:acff:fe97:2c34:fde0:3485. The port number is not specified, so the default port (1414) is used.

```
DEFINE CHANNEL(INVENTORY.LONDON) CHLTYPE(CLUSRCVR) TRPTYPE(TCP)
CONNAME(LONDON.CHSTORE.COM) CLUSTER(INVENTORY)
DESCR('TCP Cluster-receiver channel for queue manager LONDON')
```

```
1 : DEFINE CHANNEL(INVENTORY.LONDON) CHLTYPE(CLUSRCVR) TRPTYPE(TCP)
CONNAME(LONDON.CHSTORE.COM) CLUSTER(INVENTORY)
DESCR('TCP Cluster-receiver channel for queue manager LONDON')
AMQ8014: Websphere MQ channel created.
07/09/98  12:56:35  No repositories for cluster 'INVENTORY'
```

6. Define the CLUSRCVR channel for the NEWYORK queue manager.

   If the channel listener is using the default port, typically 1414, and the cluster does not include a queue manager on z/OS, you can omit the CONNAME

   ```
   DEFINE CHANNEL(INVENTORY.NEWYORK) CHLTYPE(CLUSRCVR) TRPTYPE(TCP) CLUSTER(INVENTORY)
   DESCR('TCP Cluster-receiver channel for queue manager NEWYORK')
   ```

7. Define the CLUSSDR channel on the LONDON queue manager.

   On every queue manager in a cluster, define one cluster-sender channel. The queue manager sends messages to one of the full repository queue managers on the cluster-sender channel. In this case, there are only two queue managers, both of which hold full repositories. They must each have a CLUSSDR definition that points to the CLUSRCVR channel defined at the other queue manager. The channel names given on the CLUSSDR definitions must match the channel names on the corresponding CLUSRCVR definitions. Once a queue manager has definitions for both a cluster-receiver channel and a cluster-sender channel in the same cluster, the cluster-sender channel is started.

   ```
   DEFINE CHANNEL(INVENTORY.NEWYORK) CHLTYPE(CLUSSDR) TRPTYPE(TCP)
   CONNAME(NEWYORK.CHSTORE.COM) CLUSTER(INVENTORY)
   DESCR('TCP Cluster-sender channel from LONDON to repository at NEWYORK')
   ```

   ```
   1 : DEFINE CHANNEL(INVENTORY.NEWYORK) CHLTYPE(CLUSSDR) TRPTYPE(TCP)
   CONNAME(NEWYORK.CHSTORE.COM) CLUSTER(INVENTORY)
   DESCR('TCP Cluster-sender channel from LONDON to repository at NEWYORK')
   AMQ8014: Websphere MQ channel created.
   07/09/98  13:00:18    Channel program started.
   ```

8. Define the CLUSSDR channel on the NEWYORK queue manager.

   ```
   DEFINE CHANNEL(INVENTORY.LONDON) CHLTYPE(CLUSSDR) TRPTYPE(TCP)
   CONNAME(LONDON.CHSTORE.COM) CLUSTER(INVENTORY)
   DESCR('TCP Cluster-sender channel from NEWYORK to repository at LONDON')
   ```

9. Define the cluster queue INVENTQ

   Define the INVENTQ queue on the NEWYORK queue manager, specifying the CLUSTER keyword.

   ```
   DEFINE QLOCAL(INVENTQ) CLUSTER(INVENTORY)
   ```

   ```
   1 : DEFINE QLOCAL(INVENTQ) CLUSTER(INVENTORY)
   AMQ8006: Websphere MQ queue created.
   ```

   The CLUSTER keyword causes the queue to be advertised to the cluster. As soon as the queue is defined it becomes available to the other queue managers in the cluster. They can send messages to it without having to make a remote-queue definition for it.

All the definitions are complete. On all platforms, start a listener program on each queue manager. The listener program waits for incoming network requests and starts the cluster-receiver channel when it is needed.

Setting up a cluster on TCP/IP using multiple transmission queues per queue manager

About this task

Follow these steps to set up a cluster on AIX, HP-UX, IBM i, Linux, Solaris, and Windows using the transport protocol TCP/IP. The repository queue managers are configured to use a different cluster transmission queue to send messages to each other, and to other queue managers in the cluster. If you add queue managers to the cluster that are also to use different transmission queues, follow the task, "Adding a queue manager to a cluster: separate transmission queues" on page 193. You cannot set up a queue manager on z/OS to use separate cluster transmission queues.

Procedure

1. Decide on the organization of the cluster and its name.

   You decided to link the two queue managers, LONDON and NEWYORK, into a cluster. A cluster with only two queue managers offers only marginal benefit over a network that is to use distributed queuing. It is a good way to start and it provides scope for future expansion. When you open new branches of your store, you are able to add the new queue managers to the cluster easily. Adding new queue managers does not disrupt the existing network; see "Adding a queue manager to a cluster" on page 191.

   For the time being, the only application you are running is the inventory application. The cluster name is INVENTORY.

2. Decide which queue managers are to hold full repositories.

   In any cluster you must nominate at least one queue manager, or preferably two, to hold full repositories. In this example, there are only two queue managers, LONDON and NEWYORK, both of which hold full repositories.

   a. You can perform the remaining steps in any order.

   b. As you proceed through the steps, warning messages might be written to the queue-manager log. The messages are a result of missing definitions that you have yet to add.

   ```
   Examples of the responses to the commands are shown in a box
   like this after each step in this task.
   These examples show the responses returned by WebSphere MQ for AIX.
   The responses vary on other platforms.
   ```

   c. Before proceeding with these steps, make sure that the queue managers are started.

3. Alter the queue-manager definitions to add repository definitions.

   On each queue manager that is to hold a full repository, alter the local queue-manager definition, using the ALTER QMGR command and specifying the REPOS attribute:

   ```
   ALTER QMGR REPOS(INVENTORY)
   ```

   ```
   1 : ALTER QMGR REPOS(INVENTORY)
   AMQ8005: Websphere MQ queue manager changed.
   ```

   For example, if you enter:

   a. `runmqsc LONDON`

   b. `ALTER QMGR REPOS(INVENTORY)`

   LONDON is changed to a full repository.

4. Alter the queue-manager definitions to create separate cluster transmission queues for each destination.

```
ALTER QMGR DEFCLXQ(CHANNEL)
```

   On each queue manager that you add to the cluster decide whether to use separate transmission queues or not. See the topics, “Adding a queue manager to a cluster” on page 191 and “Adding a queue manager to a cluster: separate transmission queues” on page 193.

5. Define the listeners.

   Define a listener that accepts network requests from other queue managers for every queue manager in the cluster. On the LONDON queue managers, issue the following command:

```
DEFINE LISTENER(LONDON_LS) TRPTYPE(TCP) CONTROL(QMGR)
```

   Note: When you define a listener, a port number should be defined if you are using IP addresses in the CONNAME field and the port number is not the default port (1414). For example:

```
DEFINE LISTENER(LONDON_LS) TRPTYPE(TCP) CONTROL(QMGR) PORT(1415)
```

   The CONTROL attribute ensures that the listener starts and stops when the queue manager does.

   The listener is not started when it is defined, so it must be manually started the first time with the following MQSC command:

```
START LISTENER(LONDON_LS)
```

   Issue similar commands for all the other queue managers in the cluster, changing the listener name for each one.

   There are several ways to define these listeners, as shown in Listeners.

6. Define the CLUSRCVR channel for the LONDON queue manager.

   On every queue manager in a cluster, define a cluster-receiver channel on which the queue manager can receive messages. CLUSRCVR defines the connection name of the queue manager. The connection name is stored in the repositories, where other queue managers can refer to it. The CLUSTER keyword shows the availability of the queue manager to receive messages from other queue managers in the cluster.

   In this example the channel name is INVENTORY.LONDON, and the connection name (CONNAME) is the network address of the machine the queue manager resides on, which is LONDON.CHSTORE.COM. The network address can be entered as an alphanumeric DNS host name, or an IP address in either in IPv4 dotted decimal form. For example, 192.0.2.0, or IPv6 hexadecimal form; for example 2001:DB8:0204:acff:fe97:2c34:fde0:3485. The port number is not specified, so the default port (1414) is used.

```
DEFINE CHANNEL(INVENTORY.LONDON) CHLTYPE(CLUSRCVR) TRPTYPE(TCP)
CONNAME(LONDON.CHSTORE.COM) CLUSTER(INVENTORY)
DESCR('TCP Cluster-receiver channel for queue manager LONDON')
```

```
1 : DEFINE CHANNEL(INVENTORY.LONDON) CHLTYPE(CLUSRCVR) TRPTYPE(TCP)
CONNAME(LONDON.CHSTORE.COM) CLUSTER(INVENTORY)
DESCR('TCP Cluster-receiver channel for queue manager LONDON')
AMQ8014: Websphere MQ channel created.
07/09/98  12:56:35  No repositories for cluster 'INVENTORY'
```

7. Define the CLUSRCVR channel for the NEWYORK queue manager.

   If the channel listener is using the default port, typically 1414, and the cluster does not include a queue manager on z/OS, you can omit the CONNAME

```
DEFINE CHANNEL(INVENTORY.NEWYORK) CHLTYPE(CLUSRCVR) TRPTYPE(TCP) CLUSTER(INVENTORY)
DESCR('TCP Cluster-receiver channel for queue manager NEWYORK')
```

8. Define the CLUSSDR channel on the LONDON queue manager.

   On every queue manager in a cluster, define one cluster-sender channel. The queue manager sends messages to one of the full repository queue managers on the cluster-sender channel. In this case, there are only two queue managers, both of which hold full repositories. They must each have a CLUSSDR definition that points to the CLUSRCVR channel defined at the other queue manager. The channel names given on the CLUSSDR definitions must match the channel names on the corresponding CLUSRCVR definitions. Once a queue manager has definitions for both a cluster-receiver channel and a cluster-sender channel in the same cluster, the cluster-sender channel is started.

```
DEFINE CHANNEL(INVENTORY.NEWYORK) CHLTYPE(CLUSSDR) TRPTYPE(TCP)
CONNAME(NEWYORK.CHSTORE.COM) CLUSTER(INVENTORY)
DESCR('TCP Cluster-sender channel from LONDON to repository at NEWYORK')
```

```
1 : DEFINE CHANNEL(INVENTORY.NEWYORK) CHLTYPE(CLUSSDR) TRPTYPE(TCP)
CONNAME(NEWYORK.CHSTORE.COM) CLUSTER(INVENTORY)
DESCR('TCP Cluster-sender channel from LONDON to repository at NEWYORK')
AMQ8014: Websphere MQ channel created.
07/09/98  13:00:18    Channel program started.
```

9. Define the CLUSSDR channel on the NEWYORK queue manager.

```
DEFINE CHANNEL(INVENTORY.LONDON) CHLTYPE(CLUSSDR) TRPTYPE(TCP)
CONNAME(LONDON.CHSTORE.COM) CLUSTER(INVENTORY)
DESCR('TCP Cluster-sender channel from NEWYORK to repository at LONDON')
```

10. Define the cluster queue INVENTQ

    Define the INVENTQ queue on the NEWYORK queue manager, specifying the CLUSTER keyword.

```
DEFINE QLOCAL(INVENTQ) CLUSTER(INVENTORY)
```

```
1 : DEFINE QLOCAL(INVENTQ) CLUSTER(INVENTORY)
AMQ8006: Websphere MQ queue created.
```

    The CLUSTER keyword causes the queue to be advertised to the cluster. As soon as the queue is defined it becomes available to the other queue managers in the cluster. They can send messages to it without having to make a remote-queue definition for it.

    All the definitions are complete. On all platforms, start a listener program on each queue manager. The listener program waits for incoming network requests and starts the cluster-receiver channel when it is needed.

Setting up a cluster using LU 6.2 on z/OS

Procedure

1. Decide on the organization of the cluster and its name.

   You decided to link the two queue managers, LONDON and NEWYORK, into a cluster. A cluster with only two queue managers offers only marginal benefit over a network that is to use distributed queuing. It is a good way to start and it provides scope for future expansion. When you open new branches of your store, you are able to add the new queue managers to the cluster easily. Adding new queue managers does not disrupt the existing network; see “Adding a queue manager to a cluster” on page 191.

   For the time being, the only application you are running is the inventory application. The cluster name is INVENTORY.
2. Decide which queue managers are to hold full repositories.

In any cluster you must nominate at least one queue manager, or preferably two, to hold full repositories. In this example, there are only two queue managers, LONDON and NEWYORK, both of which hold full repositories.

a. You can perform the remaining steps in any order.

b. As you proceed through the steps, warning messages might be written the z/OS system console. The messages are a result of missing definitions that you have yet to add.

c. Before proceeding with these steps, make sure that the queue managers are started.

3. Alter the queue-manager definitions to add repository definitions.

   On each queue manager that is to hold a full repository, alter the local queue-manager definition, using the ALTER QMGR command and specifying the REPOS attribute:

```
ALTER QMGR REPOS(INVENTORY)
```

```
1 : ALTER QMGR REPOS(INVENTORY)
AMQ8005: Websphere MQ queue manager changed.
```

   For example, if you enter:

   a. runmqsc LONDON

   b. ALTER QMGR REPOS(INVENTORY)

   LONDON is changed to a full repository.

4. Define the listeners.

   The listener is not started when it is defined, so it must be manually started the first time with the following MQSC command:

```
START LISTENER(LONDON_LS)
```

   Issue similar commands for all the other queue managers in the cluster, changing the listener name for each one.

5. Define the CLUSRCVR channel for the LONDON queue manager.

   On every queue manager in a cluster, define a cluster-receiver channel on which the queue manager can receive messages. CLUSRCVR defines the connection name of the queue manager. The connection name is stored in the repositories, where other queue managers can refer to it. The CLUSTER keyword shows the availability of the queue manager to receive messages from other queue managers in the cluster.

```
DEFINE CHANNEL(INVENTORY.LONDON) CHLTYPE(CLUSRCVR) TRPTYPE(LU62)
CONNAME(LONDON.LUNAME) CLUSTER(INVENTORY)
MODENAME('#INTER') TPNAME('MQSERIES')
DESCR('LU62 Cluster-receiver channel for queue manager LONDON')
```

```
1 : DEFINE CHANNEL(INVENTORY.LONDON) CHLTYPE(CLUSRCVR) TRPTYPE(LU62)
CONNAME(LONDON.LUNAME) CLUSTER(INVENTORY)
MODENAME('#INTER') TPNAME('MQSERIES')
DESCR('LU62 Cluster-receiver channel for queue manager LONDON')
AMQ8014: Websphere MQ channel created.
07/09/98  12:56:35  No repositories for cluster 'INVENTORY'
```

6. Define the CLUSRCVR channel for the NEWYORK queue manager.

```
DEFINE CHANNEL(INVENTORY.NEWYORK) CHLTYPE(CLUSRCVR) TRPTYPE(LU62)
CONNAME(NEWYORK.LUNAME) CLUSTER(INVENTORY)
MODENAME('#INTER') TPNAME('MQSERIES')
DESCR('LU62 Cluster-receiver channel for queue manager NEWYORK')
```

7. Define the CLUSSDR channel on the LONDON queue manager.

On every queue manager in a cluster, define one cluster-sender channel. The queue manager sends messages to one of the full repository queue managers on the cluster-sender channel. In this case, there are only two queue managers, both of which hold full repositories. They must each have a CLUSSDR definition that points to the CLUSRCVR channel defined at the other queue manager. The channel names given on the CLUSSDR definitions must match the channel names on the corresponding CLUSRCVR definitions. Once a queue manager has definitions for both a cluster-receiver channel and a cluster-sender channel in the same cluster, the cluster-sender channel is started.

```
DEFINE CHANNEL(INVENTORY.NEWYORK) CHLTYPE(CLUSSDR) TRPTYPE(LU62)
CONNAME(CPIC) CLUSTER(INVENTORY)
DESCR('LU62 Cluster-sender channel from LONDON to repository at NEWYORK')
```

```
1 : DEFINE CHANNEL(INVENTORY.NEWYORK) CHLTYPE(CLUSSDR) TRPTYPE(LU62)
CONNAME(NEWYORK.LUNAME) CLUSTER(INVENTORY)
MODENAME('#INTER') TPNAME('MQSERIES')
DESCR('LU62 Cluster-sender channel from LONDON to repository at NEWYORK')
AMQ8014: Websphere MQ channel created.
07/09/98  13:00:18    Channel program started.
```

8. Define the CLUSSDR channel on the NEWYORK queue manager.

```
DEFINE CHANNEL(INVENTORY.LONDON) CHLTYPE(CLUSSDR) TRPTYPE(LU62)
CONNAME(LONDON.LUNAME) CLUSTER(INVENTORY)
DESCR('LU62 Cluster-sender channel from NEWYORK to repository at LONDON')
```

9. Define the cluster queue INVENTQ

   Define the INVENTQ queue on the NEWYORK queue manager, specifying the CLUSTER keyword.

```
DEFINE QLOCAL(INVENTQ) CLUSTER(INVENTORY)
```

```
1 : DEFINE QLOCAL(INVENTQ) CLUSTER(INVENTORY)
AMQ8006: Websphere MQ queue created.
```

   The CLUSTER keyword causes the queue to be advertised to the cluster. As soon as the queue is defined it becomes available to the other queue managers in the cluster. They can send messages to it without having to make a remote-queue definition for it.

   All the definitions are complete. On all platforms, start a listener program on each queue manager. The listener program waits for incoming network requests and starts the cluster-receiver channel when it is needed.

Verifying the cluster

About this task

You can verify the cluster in one or more of these ways:

1. Running administrative commands to display cluster and channel attributes.
2. Run the sample programs to send and receive messages on a cluster queue.
3. Write your own programs to send a request message to a cluster queue and reply with a response messages to an non-clustered reply queue.

Procedure

Issue DISPLAY **runmqsc** commands to verify the cluster.

The responses you see ought to be like the responses in the steps that follow.

1. From the NEWYORK queue manager, run the **DISPLAY CLUSQMGR** command:

```
dis clusqmgr(*)
```

```
1 : dis clusqmgr(*)
AMQ8441: Display Cluster Queue Manager details.
CLUSQMGR(NEWYORK)              CLUSTER(INVENTORY)
CHANNEL(INVENTORY.NEWYORK)
AMQ8441: Display Cluster Queue Manager details.
CLUSQMGR(LONDON)               CLUSTER(INVENTORY)
CHANNEL(INVENTORY.LONDON)
```

2. From the NEWYORK queue manager, run the **DISPLAY CHANNEL STATUS** command:

```
dis chstatus(*)
```

```
1 : dis chstatus(*)
AMQ8417: Display Channel Status details.
CHANNEL(INVENTORY.NEWYORK)     XMITQ( )
CONNAME(192.0.2.0)             CURRENT
CHLTYPE(CLUSRCVR)              STATUS(RUNNING)
RQMNAME(LONDON)
AMQ8417: Display Channel Status details.
CHANNEL(INVENTORY.LONDON)      XMITQ(SYSTEM.CLUSTER.TRANSMIT.INVENTORY.LONDON)
CONNAME(192.0.2.1)             CURRENT
CHLTYPE(CLUSSDR)               STATUS(RUNNING)
RQMNAME(LONDON)
```

Send messages between the two queue managers, using **amqsput**.

3. On LONDON run the command **amqsput INVENTQ LONDON**.

   Type some messages, followed by a blank line.

4. On NEWYORK run the command **amqsget INVENTQ NEWYORK**.

   You now see the messages you entered on LONDON. After 15 seconds the program ends.

Send messages between the two queue managers using your own programs.

In the following steps, LONDON puts a message to the INVENTQ at NEWYORK and receives a reply on its queue LONDON_reply.

5. On LONDON put a messages to the cluster queue.
   a) Define a local queue called LONDON_reply.
   b) Set the MQOPEN options to MQOO_OUTPUT.
   c) Issue the MQOPEN call to open the queue INVENTQ.
   d) Set the *ReplyToQ* name in the message descriptor to LONDON_reply.
   e) Issue the MQPUT call to put the message.
   f) Commit the message.
6. On NEWYORK receive the message on the cluster queue and put a reply to the reply queue.
   a) Set the MQOPEN options to MQOO_BROWSE.
   b) Issue the MQOPEN call to open the queue INVENTQ.
   c) Issue the MQGET call to get the message from INVENTQ.
   d) Retrieve the *ReplyToQ* name from the message descriptor.
   e) Put the *ReplyToQ* name in the ObjectName field of the object descriptor.
   f) Set the MQOPEN options to MQOO_OUTPUT.
   g) Issue the MQOPEN call to open LONDON_reply at queue manager LONDON.
   h) Issue the MQPUT call to put the message to LONDON_reply.
7. On LONDON receive the reply.
   a) Set the MQOPEN options to MQOO_BROWSE.
   b) Issue the MQOPEN call to open the queue LONDON_reply.
   c) Issue the MQGET call to get the message from LONDON_reply.

Adding a queue manager to a cluster

Follow these instructions to add a queue manager to the cluster you created. Messages to cluster queues and topics are transferred using the single cluster transmission queue SYSTEM.CLUSTER.TRANSMIT.QUEUE.

Before you begin

Note: For changes to a cluster to be propagated throughout the cluster, at least one full repository must always be available. Ensure that your repositories are available before starting this task.

Scenario:

- The INVENTORY cluster is set up as described in "Setting up a new cluster" on page 181. It contains two queue managers, LONDON and NEWYORK, which both hold full repositories.
- The queue manager PARIS is owned by the primary installation. If it is not, you must run the **setmqenv** command to set up the command environment for the installation that PARIS belongs to.
- TCP connectivity exists between all three systems, and the queue manager is configured with a TCP listener that starts under the control of the queue manager.

About this task

1. A new branch of the chain store is being set up in Paris and you want to add a queue manager called PARIS to the cluster.
2. Queue manager PARIS sends inventory updates to the application running on the system in New York by putting messages on the INVENTQ queue.

Follow these steps to add a queue manager to a cluster.

Procedure

1. Decide which full repository PARIS refers to first.

   Every queue manager in a cluster must refer to one or other of the full repositories. It gathers information about the cluster from a full repository and so builds up its own partial repository. Choose either of the repositories as the full repository. As soon as a new queue manager is added to the cluster it immediately learns about the other repository as well. Information about changes to a queue manager is sent directly to two repositories. In this example, you link PARIS to the queue manager LONDON, purely for geographical reasons.

   Note: Perform the remaining steps in any order, after queue manager PARIS is started.
2. Define a CLUSRCVR channel on queue manager PARIS.

   Every queue manager in a cluster must define a cluster-receiver channel on which it can receive messages. On PARIS, define:

   ```
   DEFINE CHANNEL(INVENTORY.PARIS) CHLTYPE(CLUSRCVR) TRPTYPE(TCP)
   CONNAME(PARIS.CHSTORE.COM) CLUSTER(INVENTORY)
   DESCR('Cluster-receiver channel for queue manager PARIS')
   ```

   The cluster-receiver channel advertises the availability of the queue manager to receive messages from other queue managers in the cluster INVENTORY. There is no need to make definitions on other queue managers for a sending end to the cluster-receiver channel INVENTORY.PARIS. Other definitions are made automatically when needed.
3. Define a CLUSSDR channel on queue manager PARIS.

   Every queue manager in a cluster must define one cluster-sender channel on which it can send messages to its initial full repository.

On PARIS, make the following definition for a channel called INVENTORY.LONDON to the queue manager with the network address LONDON.CHSTORE.COM.

```
DEFINE CHANNEL(INVENTORY.LONDON) CHLTYPE(CLUSSDR) TRPTYPE(TCP)
CONNAME(LONDON.CHSTORE.COM) CLUSTER(INVENTORY)
DESCR('Cluster-sender channel from PARIS to repository at LONDON')
```

4. Optional: If this queue manager is rejoining a cluster, complete some extra steps.
   a) If you are adding a queue manager to a cluster that has previously been removed from the same cluster, check that it is now showing as a cluster member. If not, complete the following extra steps:
      i) Issue the **REFRESH CLUSTER** command on the queue manager you are adding. This step stops the cluster channels, and gives your local cluster cache a fresh set of sequence numbers that are assured to be up-to-date within the rest of the cluster.

      ```
      REFRESH CLUSTER(INVENTORY) REPOS(YES)
      ```

      Note: For large clusters, use of the **REFRESH CLUSTER** command can be disruptive to the cluster while it is in progress, and again at 27 day intervals thereafter when the cluster objects automatically send status updates to all interested queue managers. See Refreshing in a large cluster can affect performance and availability of the cluster.
      ii) Restart the CLUSSDR channel (for example, using the START CHANNEL command).
      iii) Restart the CLUSRCVR channel.
   b) If the cluster is a publish/subscribe cluster, and the rejoining queue manager has subscriptions, issue the following command to ensure the proxy subscriptions are correctly synchronized across the cluster:

      ```
      REFRESH QMGR TYPE(PROXYSUB)
      ```

Results

The following figure shows the cluster set up by this task.

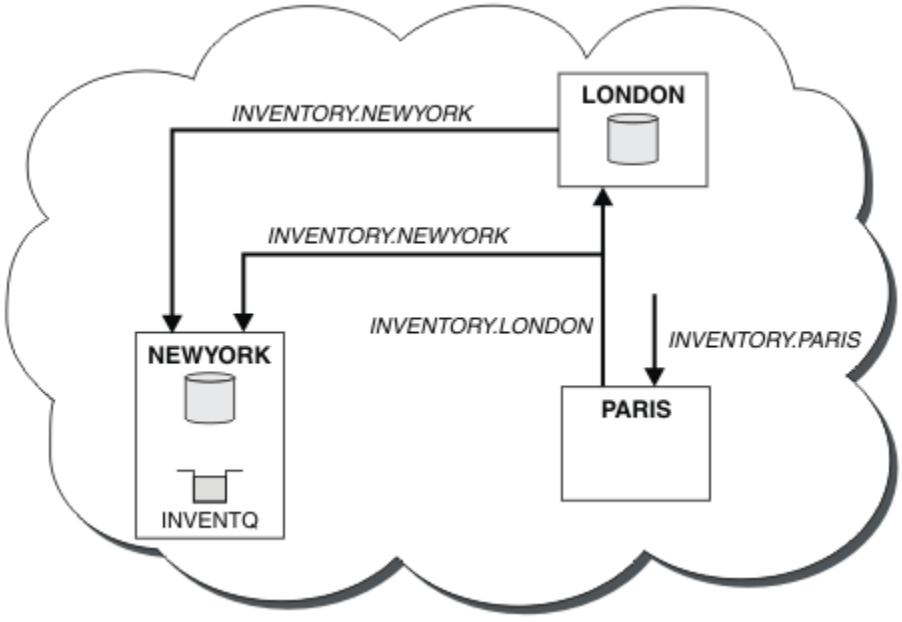


Figure 30. The INVENTORY cluster with three queue managers

By making only two definitions, a CLUSRCVR definition and a CLUSSDR definition, we added the queue manager PARIS to the cluster.

Now the PARIS queue manager learns, from the full repository at LONDON, that the INVENTQ queue is hosted by queue manager NEWYORK. When an application hosted by the system in Paris tries to put messages to the INVENTQ, PARIS automatically defines a cluster-sender channel to connect to the cluster-receiver channel INVENTORY.NEWYORK. The application can receive responses when its queue-manager name is specified as the target queue manager and a reply-to queue is provided.

Adding a queue manager to a cluster: separate transmission queues

Follow these instructions to add a queue manager to the cluster you created. Messages to cluster queues and topics are transferred using multiple cluster transmission queues.

Before you begin

- The queue manager is defined on a platform other than z/OS.
- The queue manager is not a member of any clusters.
- The cluster exists; there is a full repository to which this queue manager can connect directly and the repository is available. For the steps to create the cluster, see "Setting up a new cluster" on page 181.

About this task

This task is an alternative to "Adding a queue manager to a cluster" on page 191, in which you add a queue manager to a cluster that places cluster messages on a single transmission queue.

In this task, you add a queue manager to a cluster that automatically creates separate cluster transmission queues for each cluster-sender channel.

To keep the number of definitions of queues small, the default is to use a single transmission queue. Using separate transmission queues is advantageous if you want to monitor traffic destined to different queue managers and different clusters. You might also want to separate traffic to different destinations to achieve isolation or performance goals.

Procedure

1. Alter the default cluster channel transmission queue type.

   Alter the queue manager PARIS:

   ```
   ALTER QMGR DEFCLXQ(CHANNEL)
   ```

   Every time the queue manager creates a cluster-sender channel to send a message to a queue manager, it creates a cluster transmission queue. The transmission queue is used only by this cluster-sender channel. The transmission queue is permanent-dynamic. It is created from the model queue, SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE, with the name SYSTEM.CLUSTER.TRANSMIT.*ChannelName*.

   Attention: If you are using dedicated SYSTEM.CLUSTER.TRANSMIT.QUEUES with a queue manager that was upgraded from an earlier version of the product, ensure that the SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE has the SHARE/NOSHARE option set to **SHARE**.
2. Decide which full repository PARIS refers to first.

   Every queue manager in a cluster must refer to one or other of the full repositories. It gathers information about the cluster from a full repository and so builds up its own partial repository. Choose either of the repositories as the full repository. As soon as a new queue manager is added to the cluster it immediately learns about the other repository as well. Information about changes to a queue manager is sent directly to two repositories. In this example, you link PARIS to the queue manager LONDON, purely for geographical reasons.

   Note: Perform the remaining steps in any order, after queue manager PARIS is started.
3. Define a CLUSRCVR channel on queue manager PARIS.

Every queue manager in a cluster must define a cluster-receiver channel on which it can receive messages. On PARIS, define:

```
DEFINE CHANNEL(INVENTORY.PARIS) CHLTYPE(CLUSRCVR) TRPTYPE(TCP)
CONNAME(PARIS.CHSTORE.COM) CLUSTER(INVENTORY)
DESCR('Cluster-receiver channel for queue manager PARIS')
```

The cluster-receiver channel advertises the availability of the queue manager to receive messages from other queue managers in the cluster INVENTORY. There is no need to make definitions on other queue managers for a sending end to the cluster-receiver channel INVENTORY.PARIS. Other definitions are made automatically when needed.

4. Define a CLUSSDR channel on queue manager PARIS.

   Every queue manager in a cluster must define one cluster-sender channel on which it can send messages to its initial full repository.

   On PARIS, make the following definition for a channel called INVENTORY.LONDON to the queue manager with the network address LONDON.CHSTORE.COM.

```
DEFINE CHANNEL(INVENTORY.LONDON) CHLTYPE(CLUSSDR) TRPTYPE(TCP)
CONNAME(LONDON.CHSTORE.COM) CLUSTER(INVENTORY)
DESCR('Cluster-sender channel from PARIS to repository at LONDON')
```

   The queue manager automatically creates the permanent dynamic cluster transmission queue SYSTEM.CLUSTER.TRANSMIT.INVENTORY.LONDON from the model queue SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE. It sets the CLCHNAME attribute of the transmission queue to INVENTORY.LONDON.

Results

The following figure shows the cluster set up by this task.

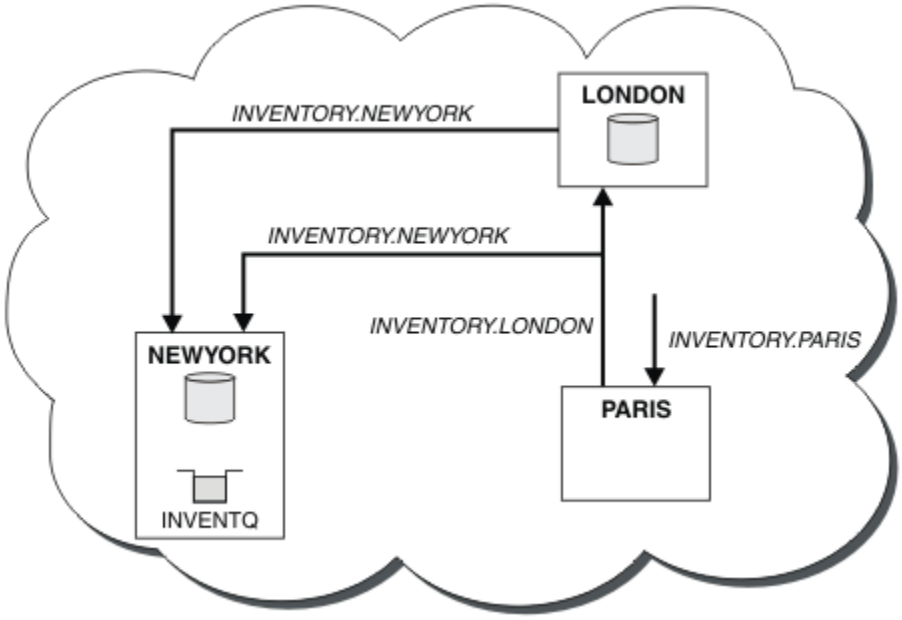


Figure 31. The INVENTORY cluster with three queue managers

By making only two definitions, a CLUSRCVR definition and a CLUSSDR definition, we added the queue manager PARIS to the cluster.

Now the PARIS queue manager learns, from the full repository at LONDON, that the INVENTQ queue is hosted by queue manager NEWYORK. When an application hosted by the system in Paris tries to put messages to the INVENTQ, PARIS automatically defines a cluster-sender channel to connect to the

cluster-receiver channel INVENTORY.NEWYORK. The application can receive responses when its queue-manager name is specified as the target queue manager and a reply-to queue is provided.

Adding a remote queue definition to isolate messages sent from a gateway queue manager

Modify the configuration of overlapping clusters that use a gateway queue manager. After the modification messages are transferred to an application from the gateway queue manager without using the same transmission queue or channels as other cluster messages. The solution uses a clustered queue remote definition, and a separate sender channel and transmission queue.

Before you begin

Construct the overlapping clusters shown in Figure 37 on page 213 in "Creating two-overlapping clusters with a gateway queue manager" on page 212 by following the steps in that task.

About this task

The solution uses distributed queueing to separate the messages for the `Server App` application from other message traffic on the gateway queue manager. You must define a clustered remote queue definition on QM1 to divert the messages to a different transmission queue, and a different channel. The remote queue definition must include a reference to the specific transmission queue that stores messages only for Q1 on QM3. In Figure 32 on page 196, the cluster queue alias Q1A is supplemented by a remote queue definition Q1R, and a transmission queue and sender-channel added.

In this solution, any reply messages are returned using the common SYSTEM.CLUSTER.TRANSMIT.QUEUE.

The advantage of this solution is that it is easy to separate traffic for multiple destination queues on the same queue manager, in the same cluster. The disadvantage of the solution is that you cannot use cluster workload balancing between multiple copies of Q1 on different queue managers. To overcome this disadvantage, see "Adding a cluster transmit queue to isolate cluster message traffic sent from a gateway queue manager" on page 198. You also have to manage the switch from one transmission queue to the other.

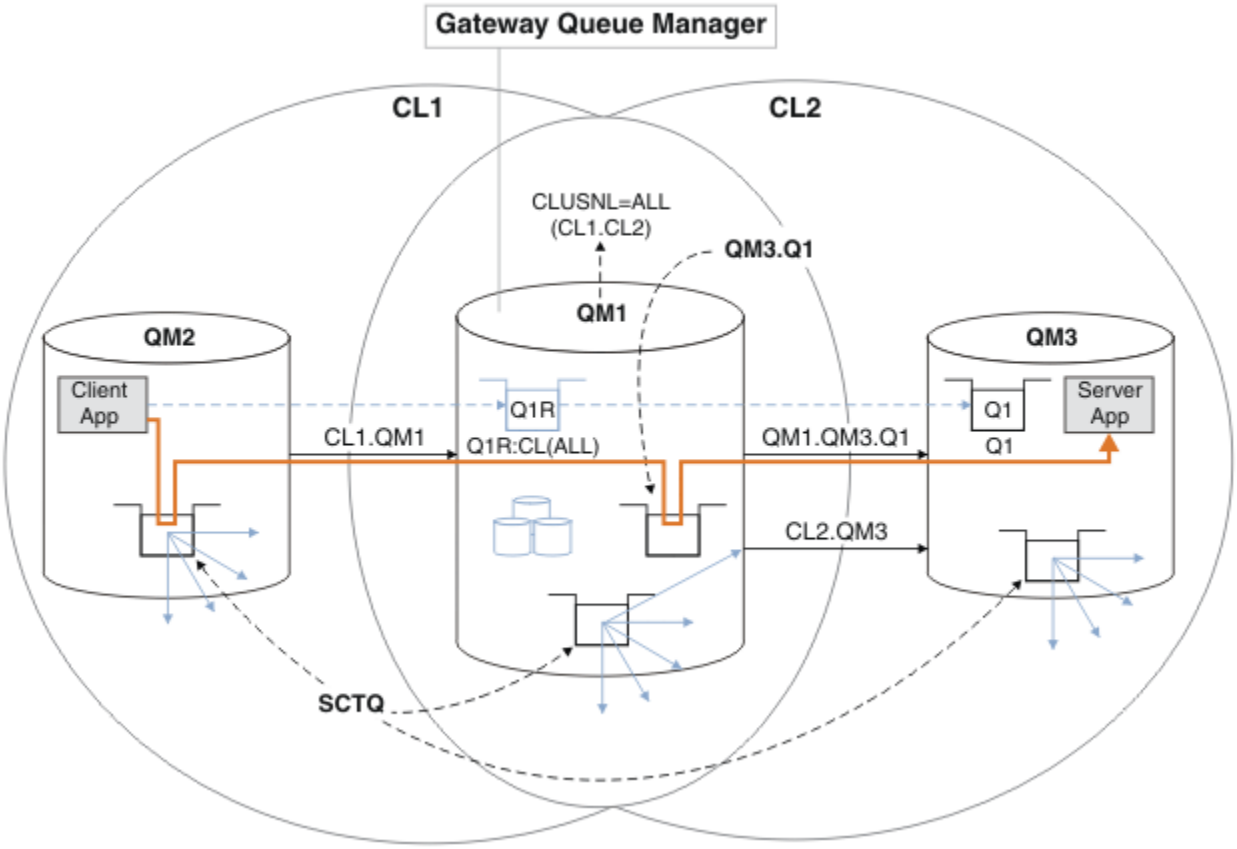


Figure 32. Client-server application deployed to hub and spoke cluster architecture using remote queue definitions

Procedure

1. Create a channel to separate the message traffic for Q1 from the gateway queue manager
   a) Create a sender channel on the gateway queue manager, QM1, to the target queue manager, QM3.

   ```
   DEFINE CHANNEL(QM1.QM3.Q1) CHLTYPE(SDR) CONNAME(QM3HostName(1413)) XMITQ(QM3.Q1) REPLACE
   ```

   b) Create a receiver channel on the target queue manager, QM3.

   ```
   DEFINE CHANNEL(QM1.QM3.Q1) CHLTYPE(RCVR) REPLACE
   ```

2. Create a transmission queue on the gateway queue manager for message traffic to Q1

   ```
   DEFINE QLOCAL(QM3.Q1) USAGE(XMITQ) REPLACE
   START CHANNEL(QM1.QM3.Q1)
   ```

   Starting the channel that is associated with the transmission queue, associates the transmission queue with the channel. The channel starts automatically, once the transmission queue has been associated with the channel.
3. Supplement the clustered queue alias definition for Q1 on the gateway queue manager with a clustered remote queue definition.

   ```
   DEFINE QREMOTE CLUSNL(ALL) RNAME(Q1) RQMNAME(QM3) XMITQ(QM3.Q1) REPLACE
   ```

What to do next

Test the configuration by sending a message to Q1 on QM3 from QM2 using the clustered queue remote definition Q1R on the gateway queue manager QM1.

1. Run the sample program **amqsput** on QM2 to put a message.

```
C:\IBM\MQ>amqsput Q1R QM2
Sample AMQSPUT0 start
target queue is Q1R
Sample request message from QM2 to Q1 using Q1R

Sample AMQSPUT0 end
```

2. Run the sample program **amqsget** to get the message from Q1 on QM3

```
C:\IBM\MQ>amqsget Q1 QM3
Sample AMQSGET0 start
message <Sample request message from QM2 to Q1 using Q1R>
no more messages
Sample AMQSGET0 end
```

Related concepts

"Access control and multiple cluster transmission queues" on page 156
Choose between three modes of checking when an application puts messages to remote cluster queues. The modes are checking remotely against the cluster queue, checking locally against SYSTEM.CLUSTER.TRANSMIT.QUEUE, or checking against local profiles for the cluster queue, or cluster queue manager.

"Clustering: Application isolation using multiple cluster transmission queues" on page 277
You can isolate the message flows between queue managers in a cluster. You can place messages being transported by different cluster-sender channels onto different cluster transmission queues. You can use the approach in a single cluster or with overlapping clusters. The topic provides examples and some best practices to guide you in choosing an approach to use.

Related tasks

"Adding a queue manager to a cluster: separate transmission queues" on page 193
Follow these instructions to add a queue manager to the cluster you created. Messages to cluster queues and topics are transferred using multiple cluster transmission queues.

"Creating two-overlapping clusters with a gateway queue manager" on page 212
Follow the instructions in the task to construct overlapping clusters with a gateway queue manager. Use the clusters as a starting point for the following examples of isolating messages to one application from messages to other applications in a cluster.

"Adding a remote queue definition to isolate messages sent from a gateway queue manager" on page 195
Modify the configuration of overlapping clusters that use a gateway queue manager. After the modification messages are transferred to an application from the gateway queue manager without using the same transmission queue or channels as other cluster messages. The solution uses a clustered queue remote definition, and a separate sender channel and transmission queue.

"Changing the default to separate cluster transmission queues to isolate message traffic" on page 218
You can change the default way a queue manager stores messages for a clustered queue or topic on a transmission queue. Changing the default provides you with a way to isolate cluster messages on a gateway queue manager.

"Adding a cluster transmit queue to isolate cluster message traffic sent from a gateway queue manager" on page 198
Modify the configuration of overlapping clusters that use a gateway queue manager. After the modification messages are transferred to an application from the gateway queue manager without using the same transmission queue or channels as other cluster messages. The solution uses an additional cluster transmission queue to separate message traffic to a single queue manager in a cluster.

"Adding a cluster and a cluster transmit queue to isolate cluster message traffic sent from a gateway queue manager" on page 201
Modify the configuration of overlapping clusters that use a gateway queue manager. After the modification messages are transferred to an application from the gateway queue manager without using the same

transmission queue or channels as other cluster messages. The solution uses an additional cluster to isolate the messages to a particular cluster queue.

"Clustering: Planning how to configure cluster transmission queues" on page 281
You are guided through the choices of cluster transmission queues. You can configure one common default queue, separate default queues, or manually defined queues. Configuring multiple cluster transmission queues applies to platforms other than z/OS.

Adding a cluster transmit queue to isolate cluster message traffic sent from a gateway queue manager

Modify the configuration of overlapping clusters that use a gateway queue manager. After the modification messages are transferred to an application from the gateway queue manager without using the same transmission queue or channels as other cluster messages. The solution uses an additional cluster transmission queue to separate message traffic to a single queue manager in a cluster.

Before you begin

1. The gateway queue manager must be on Version 7.5, or later, and on a platform other than z/OS.
2. Construct the overlapping clusters shown in Figure 37 on page 213 in "Creating two-overlapping clusters with a gateway queue manager" on page 212 by following the steps in that task.

About this task

On the gateway queue manager, `QM1`, add a transmission queue and set its queue attribute `CLCHNAME`. Set `CLCHNAME` to the name of the cluster-receiver channel on `QM3`; see Figure 33 on page 199.

This solution has a number of advantages over the solution described in "Adding a remote queue definition to isolate messages sent from a gateway queue manager" on page 195:

- It requires fewer additional definitions.
- It supports workload balancing between multiple copies of the target queue, `Q1`, on different queue managers in the same cluster, `CL2`.
- The gateway queue manager switches automatically to the new configuration when the channel restarts without loosing any messages.
- The gateway queue manager continues to forward messages in the same order as it received them. It does so, even if the switch takes place with messages for the queue `Q1` at `QM3` still on `SYSTEM.CLUSTER.TRANSMIT.QUEUE`.

The configuration to isolate cluster message traffic in Figure 33 on page 199 does not result in as great an isolation of traffic as the configuration using remote queues in "Adding a remote queue definition to isolate messages sent from a gateway queue manager" on page 195. If the queue manager `QM3` in `CL2` is hosting a number of different cluster queues and server applications, all those queues share the cluster channel, `CL2.QM3`, connecting `QM1` to `QM3`. The additional flows are illustrated in Figure 33 on page 199 by the gray arrow representing potential cluster message traffic from the `SYSTEM.CLUSTER.TRANSMIT.QUEUE` to the cluster-sender channel `CL2.QM3`.

The remedy is to restrict the queue manager to hosting one cluster queue in a particular cluster. If the queue manager is already hosting a number of cluster queues, then to meet this restriction, you must either create another queue manager, or create another cluster; see "Adding a cluster and a cluster transmit queue to isolate cluster message traffic sent from a gateway queue manager" on page 201.

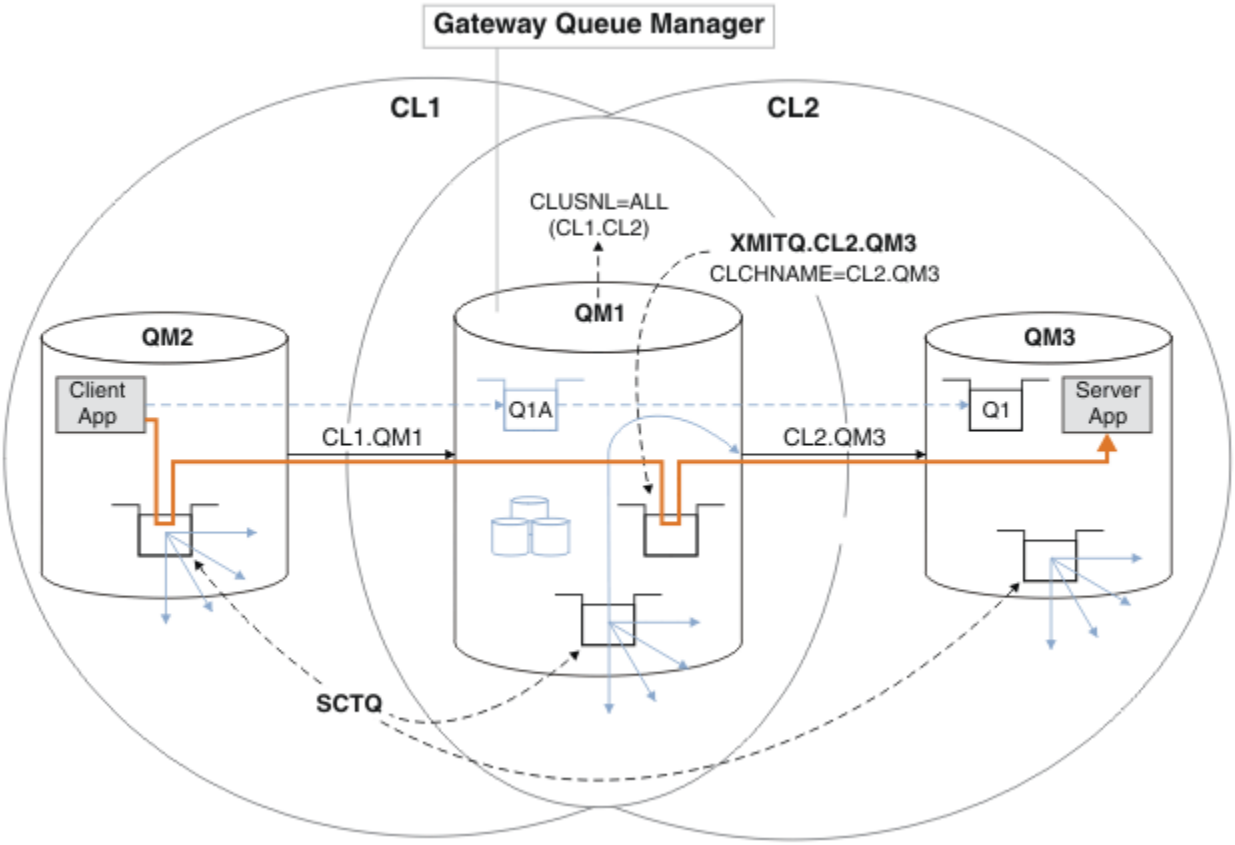


Figure 33. Client-server application deployed to hub and spoke architecture using an additional cluster transmission queue.

Procedure

1. Create an additional cluster transmission queue for the cluster-sender channel `CL2.QM3` on the gateway queue manager, `QM1`.

```
*... on QM1
DEFINE QLOCAL(XMITQ.CL2.QM3) USAGE(XMITQ) CLCHNAME(CL2.QM3)
```

2. Switch to using the transmission queue, `XMITQ.CL2.QM3`.
   a) Stop the cluster-sender channel `CL2.QM3`.

```
*... On QM1
STOP CHANNEL(CL2.QM3)
```

   The response is that the command is accepted:

```
AMQ8019: Stop WebSphere MQ channel accepted.
```

   b) Check that the channel `CL2.QM3` is stopped

   If the channel does not stop, you can run the **STOP CHANNEL** command again with the FORCE option. An example of setting the FORCE option would be if the channel does not stop, and you cannot restart the other queue manager to synchronize the channel.

```
*... On QM1
start
```

   The response is a summary of the channel status

```
AMQ8417: Display Channel Status details.
   CHANNEL(CL2.QM3)              CHLTYPE(CLUSSDR)
   CONNAME(127.0.0.1(1413))      CURRENT
   RQMNAME(QM3)                  STATUS(STOPPED)
   SUBSTATE(MQGET)               XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)
```

c) Start the channel, CL2.QM3.

```
*... On QM1
START CHANNEL(CL2.QM3)
```

The response is that the command is accepted:

```
AMQ8018: Start WebSphere MQ channel accepted.
```

d) Check the channel started.

```
*... On QM1
DISPLAY CHSTATUS(CL2.QM3)
```

The response is a summary of the channel status:

```
AMQ8417: Display Channel Status details.
   CHANNEL(CL2.QM3)                  CHLTYPE(CLUSSDR)
   CONNAME(127.0.0.1(1413))          CURRENT
   RQMNAME(QM3)                      STATUS(RUNNING)
   SUBSTATE(MQGET)                   XMITQ(XMITQ.CL2.QM3)
```

e) Check the transmission queue was switched.

Monitor the gateway queue manager error log for the message "AMQ7341 The transmission queue for channel CL2.QM3 is XMITQ.CL2.QM3".

What to do next

Test the separate transmission queue by sending a message from QM2 to Q1 on QM3 using the queue alias definition Q1A

1. Run the sample program **amqsput** on QM2 to put a message.

```
C:\IBM\MQ>amqsput Q1A QM2
Sample AMQSPUT0 start
target queue is Q1A
Sample request message from QM2 to Q1 using Q1A

Sample AMQSPUT0 end
```

2. Run the sample program **amqsget** to get the message from Q1 on QM3

```
C:\IBM\MQ>amqsget Q1 QM3
Sample AMQSGET0 start
message <Sample request message from QM2 to Q1 using Q1A>
no more messages
Sample AMQSGET0 end
```

Related concepts

“Access control and multiple cluster transmission queues” on page 156
Choose between three modes of checking when an application puts messages to remote cluster queues. The modes are checking remotely against the cluster queue, checking locally against SYSTEM.CLUSTER.TRANSMIT.QUEUE, or checking against local profiles for the cluster queue, or cluster queue manager.

“Clustering: Application isolation using multiple cluster transmission queues” on page 277
You can isolate the message flows between queue managers in a cluster. You can place messages being transported by different cluster-sender channels onto different cluster transmission queues. You can use

the approach in a single cluster or with overlapping clusters. The topic provides examples and some best practices to guide you in choosing an approach to use.

"Cluster transmission queues and cluster-sender channels" on page 166
Messages between clustered queue managers are stored on cluster transmission queues and forwarded by cluster-sender channels.

Related tasks

"Adding a queue manager to a cluster: separate transmission queues" on page 193
Follow these instructions to add a queue manager to the cluster you created. Messages to cluster queues and topics are transferred using multiple cluster transmission queues.

"Creating two-overlapping clusters with a gateway queue manager" on page 212
Follow the instructions in the task to construct overlapping clusters with a gateway queue manager. Use the clusters as a starting point for the following examples of isolating messages to one application from messages to other applications in a cluster.

"Adding a remote queue definition to isolate messages sent from a gateway queue manager" on page 195
Modify the configuration of overlapping clusters that use a gateway queue manager. After the modification messages are transferred to an application from the gateway queue manager without using the same transmission queue or channels as other cluster messages. The solution uses a clustered queue remote definition, and a separate sender channel and transmission queue.

"Changing the default to separate cluster transmission queues to isolate message traffic" on page 218
You can change the default way a queue manager stores messages for a clustered queue or topic on a transmission queue. Changing the default provides you with a way to isolate cluster messages on a gateway queue manager.

"Adding a cluster transmit queue to isolate cluster message traffic sent from a gateway queue manager" on page 198
Modify the configuration of overlapping clusters that use a gateway queue manager. After the modification messages are transferred to an application from the gateway queue manager without using the same transmission queue or channels as other cluster messages. The solution uses an additional cluster transmission queue to separate message traffic to a single queue manager in a cluster.

"Adding a cluster and a cluster transmit queue to isolate cluster message traffic sent from a gateway queue manager" on page 201
Modify the configuration of overlapping clusters that use a gateway queue manager. After the modification messages are transferred to an application from the gateway queue manager without using the same transmission queue or channels as other cluster messages. The solution uses an additional cluster to isolate the messages to a particular cluster queue.

"Clustering: Planning how to configure cluster transmission queues" on page 281
You are guided through the choices of cluster transmission queues. You can configure one common default queue, separate default queues, or manually defined queues. Configuring multiple cluster transmission queues applies to platforms other than z/OS.

Adding a cluster and a cluster transmit queue to isolate cluster message traffic sent from a gateway queue manager

Modify the configuration of overlapping clusters that use a gateway queue manager. After the modification messages are transferred to an application from the gateway queue manager without using the same transmission queue or channels as other cluster messages. The solution uses an additional cluster to isolate the messages to a particular cluster queue.

Before you begin

The steps in the task are written to modify the configuration illustrated in Figure 33 on page 199.

1. The gateway queue manager must be on Version 7.5, or later, and on a platform other than z/OS.
2. Construct the overlapping clusters shown in Figure 37 on page 213 in "Creating two-overlapping clusters with a gateway queue manager" on page 212 by following the steps in that task.

3. Do the steps in Figure 33 on page 199in "Adding a cluster transmit queue to isolate cluster message traffic sent from a gateway queue manager" on page 198 to create the solution without the additional cluster. Use this as a base for the steps in this task.

About this task

The solution to isolating message traffic to a single application in "Adding a cluster transmit queue to isolate cluster message traffic sent from a gateway queue manager" on page 198 works if the target cluster queue is the only cluster queue on a queue manager. If it is not, you have two choices. Either move the queue to a different queue manager, or create a cluster that isolates the queue from other cluster queues on the queue manager.

This task takes you through the steps to add a cluster to isolate the target queue. The cluster is added just for that purpose. In practice, approach the task of isolating certain applications systematically when you are in the process of designing clusters and cluster naming schemes. Adding a cluster each time a queue requires isolation might end up with many clusters to manage. In this task, you change the configuration in "Adding a cluster transmit queue to isolate cluster message traffic sent from a gateway queue manager" on page 198 by adding a cluster `CL3` to isolate `Q1` on `QM3`. Applications continue to run throughout the change.

The new and changed definitions are highlighted in Figure 34 on page 203. The summary of the changes is as follows: Create a cluster, which means you must also create a new full cluster repository. In the example, `QM3` is made one of the full repositories for `CL3`. Create cluster-sender and cluster-receiver channels for `QM1`to add the gateway queue manager to the new cluster. Change the definition of `Q1` to switch it to `CL3`. Modify the cluster namelist on the gateway queue manager, and add a cluster transmission queue to use the new cluster channel. Finally, switch the queue alias `Q1A` to the new cluster namelist.

IBM WebSphere MQ cannot transfer messages from the transmission queue `XMITQ.CL2.QM3` that you added in "Adding a cluster transmit queue to isolate cluster message traffic sent from a gateway queue manager" on page 198 to the new transmission queue `XMITQ.CL3.QM3`, automatically. It can transfer messages automatically only if both transmission queues are served by the same cluster-sender channel. Instead, the task describes one way to perform the switch manually, which might be appropriate to you. When the transfer is completed, you have the option of reverting to using the default cluster transmission queue for other `CL2` cluster queues on `QM3`. Or you can continue to use `XMITQ.CL2.QM3`. If you decide to revert to a default cluster transmission queue, the gateway queue manager manages the switch for you automatically.

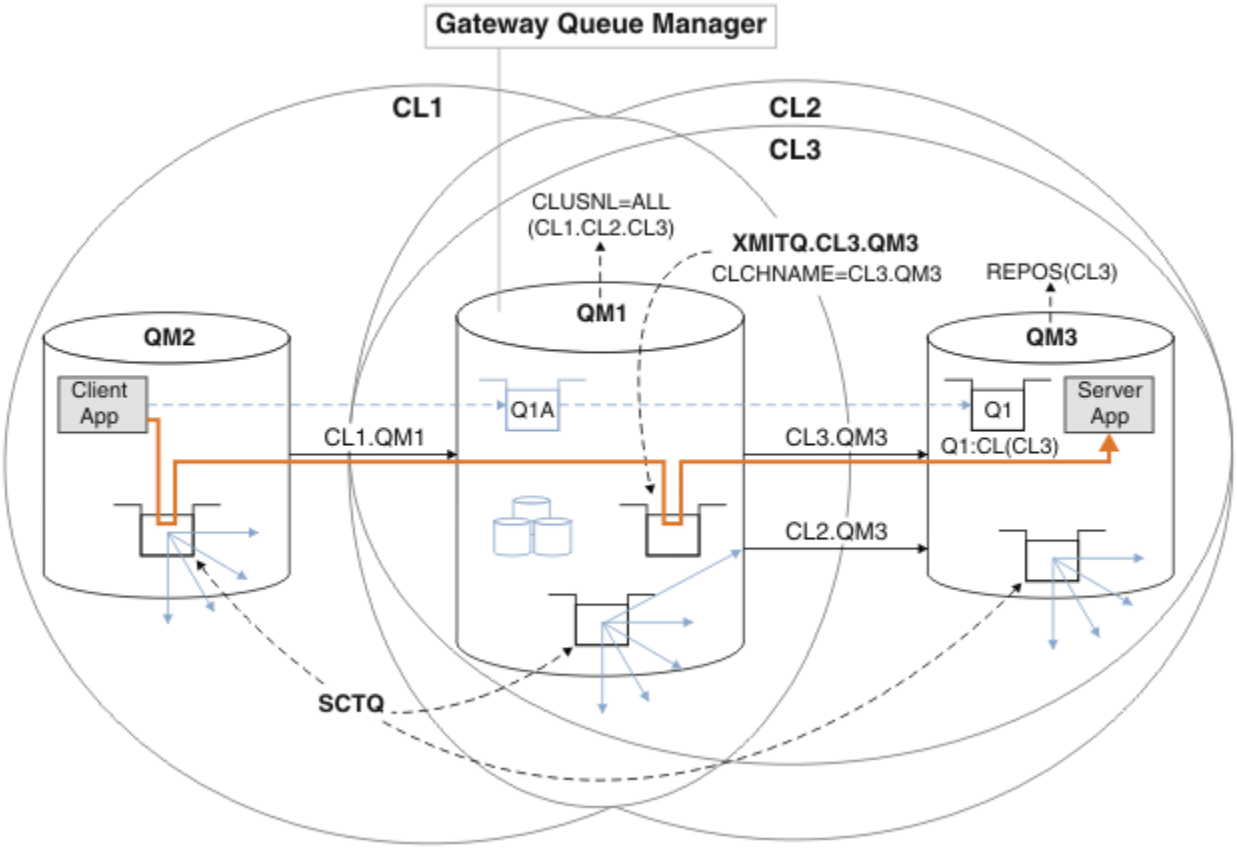


Figure 34. Using an additional cluster to separate message traffic in the gateway queue manager that goes to one of a number of cluster queues on the same queue manager

Procedure

1. Alter the queue managers QM3 and QM5 to make them repositories for both CL2 and CL3.

   To make a queue manager a member of multiple clusters, it must use a cluster name list to identify the clusters it is a member of.

   ```
   *... On QM3 and QM5
   DEFINE NAMELIST(CL23) NAMES(CL2, CL3) REPLACE
   ALTER QMGR REPOS(' ') REPOSNL(CL23)
   ```

2. Define the channels between the queue managers QM3 and QM5 for CL3.

   ```
   *... On QM3
   DEFINE CHANNEL(CL3.QM5) CHLTYPE(CLUSSDR)  CONNAME('localhost(1415)') CLUSTER(CL3) REPLACE
   DEFINE CHANNEL(CL3.QM3) CHLTYPE(CLUSRCVR) CONNAME('localhost(1413)') CLUSTER(CL3) REPLACE

   *... On QM5
   DEFINE CHANNEL(CL3.QM3) CHLTYPE(CLUSSDR)  CONNAME('localhost(1413)') CLUSTER(CL3) REPLACE
   DEFINE CHANNEL(CL3.QM5) CHLTYPE(CLUSRCVR) CONNAME('localhost(1415)') CLUSTER(CL3) REPLACE
   ```

3. Add the gateway queue manager to CL3.

   Add the gateway queue manager by adding QM1 to CL3 as a partial repository. Create a partial repository by adding cluster-sender and cluster-receiver channels to QM1.

   Also, add CL3 to the name list of all clusters connected to the gateway queue manager.

   ```
   *... On QM1
   DEFINE CHANNEL(CL3.QM3) CHLTYPE(CLUSSDR)  CONNAME('localhost(1413)') CLUSTER(CL3) REPLACE
   DEFINE CHANNEL(CL3.QM1) CHLTYPE(CLUSRCVR) CONNAME('localhost(1411)') CLUSTER(CL3) REPLACE
   ALTER NAMELIST(ALL) NAMES(CL1, CL2, CL3)
   ```

4. Add a cluster transmission queue to the gateway queue manager, QM1, for messages going to CL3 on QM3.

   Initially, stop the cluster-sender channel transferring messages from the transmission queue until you are ready to switch transmission queues.

   ```
   *... On QM1
   DEFINE QLOCAL(XMITQ.CL3.QM3) USAGE(XMITQ) CLCHNAME(CL3.QM3) GET(DISABLED) REPLACE
   ```

5. Drain messages from the existing cluster transmission queue XMITQ.CL2.QM3.

   This subprocedure is intended to preserve the order of messages in Q1 to match the order they arrived at the gateway queue manager. With clusters, message ordering is not fully guaranteed, but is likely. If guaranteed message ordering is required, applications must define the order of messages; see The order in which messages are retrieved from a queue.

   a) Change the target queue Q1 on QM3 from CL2 to CL3.

      ```
      *... On QM3
      ALTER QLOCAL(Q1) CLUSTER(CL3)
      ```

   b) Monitor XMITQ.CL3.QM3 until messages start to be delivered to it.

      Messages start to be delivered to XMITQ.CL3.QM3 when the switch of Q1 to CL3 is propagated to the gateway queue manager.

      ```
      *... On QM1
      DISPLAY QUEUE(XMITQ.CL3.QM3) CURDEPTH
      ```

   c) Monitor XMITQ.CL2.QM3 until it has no messages waiting to be delivered to Q1 on QM3.

      Note: XMITQ.CL2.QM3 might be storing messages for other queues on QM3 that are members of CL2, in which case the depth might not go to zero.

      ```
      *... On QM1
      DISPLAY QUEUE(XMITQ.CL2.QM3) CURDEPTH
      ```

   d) Enable get from the new cluster transmission queue, XMITQ.CL3.QM3

      ```
      *... On QM1
      ALTER QLOCAL(XMITQ.CL3.QM3) GET(ENABLED)
      ```

6. Remove the old cluster transmission queue, XMITQ.CL2.QM3, if it is no longer required.

   Messages for cluster queues in CL2 on QM3 revert to using the default cluster transmission queue on the gateway queue manager, QM1. The default cluster transmission queue is either SYSTEM.CLUSTER.TRANSMIT.QUEUE, or SYSTEM.CLUSTER.TRANSMIT.CL2.QM3. Which one depends on whether the value of the queue manager attribute **DEFCLXQ** on QM1 is SCTQ or CHANNEL. The queue manager transfers messages from XMITQ.CL2.QM3 automatically when the cluster-sender channel CL2.QM3 next starts.

   a) Change the transmission queue, XMITQ.CL2.QM3, from being a cluster transmission queue to being a normal transmission queue.

      This breaks the association of the transmission queue with any cluster-sender channels. In response, IBM WebSphere MQ automatically transfers messages from XMITQ.CL2.QM3 to the default cluster transmission queue when the cluster-sender channel is next started. Until then, messages for CL2 on QM3 continue to be placed on XMITQ.CL2.QM3.

      ```
      *... On QM1
      ALTER QLOCAL(XMITQ.CL2.QM3) CLCHNAME(' ')
      ```

   b) Stop the cluster-sender channel CL2.QM3.

Stopping and restarting the cluster-sender channel initiates the transfer of messages from XMITQ.CL2.QM3 to the default cluster transmission queue. Typically you would stop and start the channel manually to start the transfer. The transfer starts automatically if the channel restarts after shutting down on the expiry of its disconnect interval.

```
*... On QM1
STOP CHANNEL(CL2.QM3)
```

The response is that the command is accepted:

```
AMQ8019: Stop WebSphere MQ channel accepted.
```

c) Check that the channel CL2.QM3 is stopped

If the channel does not stop, you can run the **STOP CHANNEL** command again with the FORCE option. An example of setting the FORCE option would be if the channel does not stop, and you cannot restart the other queue manager to synchronize the channel.

```
*... On QM1
DISPLAY CHSTATUS(CL2.QM3)
```

The response is a summary of the channel status

```
AMQ8417: Display Channel Status details.
   CHANNEL(CL2.QM3)                        CHLTYPE(CLUSSDR)
   CONNAME(127.0.0.1(1413))                CURRENT
   RQMNAME(QM3)                            STATUS(STOPPED)
   SUBSTATE(MQGET)                         XMITQ(XMITQ.CL2.QM3)
```

d) Start the channel, CL2.QM3.

```
*... On QM1
START CHANNEL(CL2.QM3)
```

The response is that the command is accepted:

```
AMQ8018: Start WebSphere MQ channel accepted.
```

e) Check the channel started.

```
*... On QM1
DISPLAY CHSTATUS(CL2.QM3)
```

The response is a summary of the channel status:

```
AMQ8417: Display Channel Status details.
   CHANNEL(CL2.QM3)           CHLTYPE(CLUSSDR)
   CONNAME(127.0.0.1(1413))   CURRENT
   RQMNAME(QM3)               STATUS(RUNNING)
   SUBSTATE(MQGET)            XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE|CL2.QM3)
```

f) Monitor the gateway queue manager error log for the message "AMQ7341 The transmission queue for channel CL2.QM3 is SYSTEM.CLUSTER.TRANSMIT.*QUEUE*|*CL2.QM3*".

g) Delete the cluster transmission queue, XMITQ.CL2.QM3.

```
*... On QM1
DELETE QLOCAL(XMITQ.CL2.QM3)
```

What to do next

Test the separately clustered queue by sending a message from QM2 to Q1 on QM3 using the queue alias definition Q1A

1. Run the sample program **amqsput** on QM2 to put a message.

```
C:\IBM\MQ>amqsput Q1A QM2
Sample AMQSPUT0 start
target queue is Q1A
Sample request message from QM2 to Q1 using Q1A

Sample AMQSPUT0 end
```

2. Run the sample program **amqsget** to get the message from Q1 on QM3

```
C:\IBM\MQ>amqsget Q1 QM3
Sample AMQSGET0 start
message <Sample request message from QM2 to Q1 using Q1A>
no more messages
Sample AMQSGET0 end
```

Related concepts

“Access control and multiple cluster transmission queues” on page 156
Choose between three modes of checking when an application puts messages to remote cluster queues. The modes are checking remotely against the cluster queue, checking locally against SYSTEM.CLUSTER.TRANSMIT.QUEUE, or checking against local profiles for the cluster queue, or cluster queue manager.

“Clustering: Application isolation using multiple cluster transmission queues” on page 277
You can isolate the message flows between queue managers in a cluster. You can place messages being transported by different cluster-sender channels onto different cluster transmission queues. You can use the approach in a single cluster or with overlapping clusters. The topic provides examples and some best practices to guide you in choosing an approach to use.

“Cluster transmission queues and cluster-sender channels” on page 166
Messages between clustered queue managers are stored on cluster transmission queues and forwarded by cluster-sender channels.

Related tasks

“Adding a queue manager to a cluster: separate transmission queues” on page 193
Follow these instructions to add a queue manager to the cluster you created. Messages to cluster queues and topics are transferred using multiple cluster transmission queues.

“Creating two-overlapping clusters with a gateway queue manager” on page 212
Follow the instructions in the task to construct overlapping clusters with a gateway queue manager. Use the clusters as a starting point for the following examples of isolating messages to one application from messages to other applications in a cluster.

“Adding a remote queue definition to isolate messages sent from a gateway queue manager” on page 195
Modify the configuration of overlapping clusters that use a gateway queue manager. After the modification messages are transferred to an application from the gateway queue manager without using the same transmission queue or channels as other cluster messages. The solution uses a clustered queue remote definition, and a separate sender channel and transmission queue.

“Changing the default to separate cluster transmission queues to isolate message traffic” on page 218
You can change the default way a queue manager stores messages for a clustered queue or topic on a transmission queue. Changing the default provides you with a way to isolate cluster messages on a gateway queue manager.

“Adding a cluster transmit queue to isolate cluster message traffic sent from a gateway queue manager” on page 198
Modify the configuration of overlapping clusters that use a gateway queue manager. After the modification messages are transferred to an application from the gateway queue manager without using the same transmission queue or channels as other cluster messages. The solution uses an additional cluster transmission queue to separate message traffic to a single queue manager in a cluster.

“Adding a cluster and a cluster transmit queue to isolate cluster message traffic sent from a gateway queue manager” on page 201

Modify the configuration of overlapping clusters that use a gateway queue manager. After the modification messages are transferred to an application from the gateway queue manager without using the same transmission queue or channels as other cluster messages. The solution uses an additional cluster to isolate the messages to a particular cluster queue.

"Clustering: Planning how to configure cluster transmission queues" on page 281
You are guided through the choices of cluster transmission queues. You can configure one common default queue, separate default queues, or manually defined queues. Configuring multiple cluster transmission queues applies to platforms other than z/OS.

Adding a queue manager to a cluster by using DHCP

Add a queue manager to a cluster, using DHCP. The task demonstrates omitting CONNAME value on a CLUSRCVR definition.

Before you begin

Note: For changes to a cluster to be propagated throughout the cluster, at least one full repository must always be available. Ensure that your repositories are available before starting this task.

The task demonstrates two special features:

- The ability to omit the CONNAME value on a CLUSRCVR definition.
- The ability to use +QMNAME+ on a CLUSSDR definition.

Neither feature is provided on z/OS.

Scenario:

- The INVENTORY cluster has been set up as described in "Setting up a new cluster" on page 181. It contains two queue managers, LONDON and NEWYORK, which both hold full repositories.
- A new branch of the chain store is being set up in Paris and you want to add a queue manager called PARIS to the cluster.
- Queue manager PARIS sends inventory updates to the application running on the system in New York by putting messages on the INVENTQ queue.
- Network connectivity exists between all three systems.
- The network protocol is TCP.
- The PARIS queue manager system uses DHCP, which means that the IP addresses might change on system restart.
- The channels between the PARIS and LONDON systems are named according to a defined naming convention. The convention uses the queue manager name of the full repository queue manager on LONDON.
- Administrators of the PARIS queue manager have no information about the name of the queue manager on the LONDON repository. The name of the queue manager on the LONDON repository is subject to change.

About this task

Follow these steps to add a queue manager to a cluster by using DHCP.

Procedure

1. Decide which full repository PARIS refers to first.

   Every queue manager in a cluster must refer to one or other of the full repositories. It gathers information about the cluster from a full repository and so builds up its own partial repository. Choose either of the repositories as the full repository. As soon as a new queue manager is added to the cluster it immediately learns about the other repository as well. Information about changes to a queue

manager is sent directly to two repositories. In this example we choose to link PARIS to the queue manager LONDON, purely for geographical reasons.

Note: Perform the remaining steps in any order, after queue manager PARIS is started.

2. Define a CLUSRCVR channel on queue manager PARIS.

   Every queue manager in a cluster needs to define a cluster-receiver channel on which it can receive messages. On PARIS, define:

   ```
   DEFINE CHANNEL(INVENTORY.PARIS) CHLTYPE(CLUSRCVR)
   TRPTYPE(TCP) CLUSTER(INVENTORY)
   DESCR('Cluster-receiver channel for queue manager PARIS')
   ```

   The cluster-receiver channel advertises the availability of the queue manager to receive messages from other queue managers in the cluster INVENTORY. You do not need to specify the CONNAME on the cluster-receiver channel. You can request IBM WebSphere MQ to find out the connection name from the system, either by omitting CONNAME, or by specifying CONNAME(' '). IBM WebSphere MQ generates the CONNAME value using the current IP address of the system; see CONNAME. There is no need to make definitions on other queue managers for a sending end to the cluster-receiver channel INVENTORY.PARIS. Other definitions are made automatically when needed.

3. Define a CLUSSDR channel on queue manager PARIS.

   Every queue manager in a cluster needs to define one cluster-sender channel on which it can send messages to its initial full repository. On PARIS, make the following definition for a channel called INVENTORY.+QMNAME+ to the queue manager with the network address LONDON.CHSTORE.COM.

   ```
   DEFINE CHANNEL(INVENTORY.+QMNAME+) CHLTYPE(CLUSSDR) TRPTYPE(TCP)
   CONNAME(LONDON.CHSTORE.COM) CLUSTER(INVENTORY)
   DESCR('Cluster-sender channel from PARIS to repository at LONDON')
   ```

4. Optional: If this queue manager is rejoining a cluster, complete some extra steps.

   a) If you are adding a queue manager to a cluster that has previously been removed from the same cluster, check that it is now showing as a cluster member. If not, complete the following extra steps:

      i) Issue the **REFRESH CLUSTER** command on the queue manager you are adding. This step stops the cluster channels, and gives your local cluster cache a fresh set of sequence numbers that are assured to be up-to-date within the rest of the cluster.

      ```
      REFRESH CLUSTER(INVENTORY) REPOS(YES)
      ```

      Note: For large clusters, use of the **REFRESH CLUSTER** command can be disruptive to the cluster while it is in progress, and again at 27 day intervals thereafter when the cluster objects automatically send status updates to all interested queue managers. See Refreshing in a large cluster can affect performance and availability of the cluster.

      ii) Restart the CLUSSDR channel (for example, using the START CHANNEL command).

      iii) Restart the CLUSRCVR channel.

   b) If the cluster is a publish/subscribe cluster, and the rejoining queue manager has subscriptions, issue the following command to ensure the proxy subscriptions are correctly synchronized across the cluster:

   ```
   REFRESH QMGR TYPE(PROXYSUB)
   ```

Results

The cluster set up by this task is the same as for “Adding a queue manager to a cluster” on page 191:

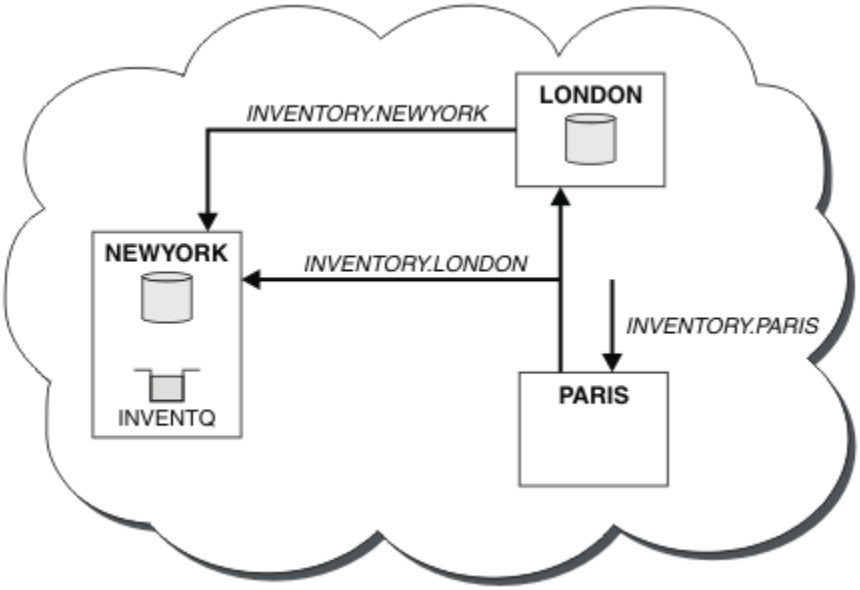


Figure 35. The INVENTORY cluster with three queue managers

By making only two definitions, a CLUSRCVR definition and a CLUSSDR definition, we have added the queue manager PARIS to the cluster.

On the PARIS queue manager, the CLUSSDR containing the string +QMNAME+ starts. On the LONDON system IBM WebSphere MQ resolves the +QMNAME+ to the queue manager name (LONDON). IBM WebSphere MQ then matches the definition for a channel called INVENTORY.LONDON to the corresponding CLUSRCVR definition.

WebSphere MQ sends back the resolved channel name to the PARIS queue manager. At PARIS, the CLUSSDR channel definition for the channel called INVENTORY.+QMNAME+ is replaced by an internally generated CLUSSDR definition for INVENTORY.LONDON. This definition contains the resolved channel name, but otherwise is the same as the +QMNAME+ definition that you made. The cluster repositories are also brought up to date with the channel definition with the newly resolved channel name.

Note:

1. The channel created with the +QMNAME+ name becomes inactive immediately. It is never used to transmit data.
2. Channel exits might see the channel name change between one invocation and the next.

Now the PARIS queue manager learns, from the repository at LONDON, that the INVENTQ queue is hosted by queue manager NEWYORK. When an application hosted by the system in Paris tries to put messages to the INVENTQ, PARIS automatically defines a cluster-sender channel to connect to the cluster-receiver channel INVENTORY.NEWYORK. The application can receive responses when its queue-manager name is specified as the target queue manager and a reply-to queue is provided.

Related reference

DEFINE CHANNEL

Adding a queue manager that hosts a queue

Add another queue manager to the cluster, to host another INVENTQ queue. Requests are sent alternately to the queues on each queue manager. No changes need to be made to the existing INVENTQ host.

Before you begin

Note: For changes to a cluster to be propagated throughout the cluster, at least one full repository must always be available. Ensure that your repositories are available before starting this task.

Scenario:

- The INVENTORY cluster has been set up as described in “Adding a queue manager to a cluster” on page 191. It contains three queue managers; LONDON and NEWYORK both hold full repositories, PARIS holds a partial repository. The inventory application runs on the system in New York, connected to the NEWYORK queue manager. The application is driven by the arrival of messages on the INVENTQ queue.
- A new store is being set up in Toronto. To provide additional capacity you want to run the inventory application on the system in Toronto as well as New York.
- Network connectivity exists between all four systems.
- The network protocol is TCP.

Note: The queue manager TORONTO contains only a partial repository. If you want to add a full-repository queue manager to a cluster, refer to “Moving a full repository to another queue manager” on page 225.

About this task

Follow these steps to add a queue manager that hosts a queue.

Procedure

1. Decide which full repository TORONTO refers to first.

   Every queue manager in a cluster must refer to one or other of the full repositories. It gathers information about the cluster from a full repository and so builds up its own partial repository. It is of no particular significance which repository you choose. In this example, we choose NEWYORK. Once the new queue manager has joined the cluster it communicates with both of the repositories.
2. Define the CLUSRCVR channel.

   Every queue manager in a cluster needs to define a cluster-receiver channel on which it can receive messages. On TORONTO, define a CLUSRCVR channel:

   ```
   DEFINE CHANNEL(INVENTORY.TORONTO) CHLTYPE(CLUSRCVR) TRPTYPE(TCP)
   CONNAME(TORONTO.CHSTORE.COM) CLUSTER(INVENTORY)
   DESCR('Cluster-receiver channel for TORONTO')
   ```

   The TORONTO queue manager advertises its availability to receive messages from other queue managers in the INVENTORY cluster using its cluster-receiver channel.
3. Define a CLUSSDR channel on queue manager TORONTO.

   Every queue manager in a cluster needs to define one cluster-sender channel on which it can send messages to its first full repository. In this case choose NEWYORK. TORONTO needs the following definition:

   ```
   DEFINE CHANNEL(INVENTORY.NEWYORK) CHLTYPE(CLUSSDR) TRPTYPE(TCP)
   CONNAME(NEWYORK.CHSTORE.COM) CLUSTER(INVENTORY)
   DESCR('Cluster-sender channel from TORONTO to repository at NEWYORK')
   ```
4. Optional: If this queue manager is rejoining a cluster, complete some extra steps.
   a) If you are adding a queue manager to a cluster that has previously been removed from the same cluster, check that it is now showing as a cluster member. If not, complete the following extra steps:
      i) Issue the **REFRESH CLUSTER** command on the queue manager you are adding. This step stops the cluster channels, and gives your local cluster cache a fresh set of sequence numbers that are assured to be up-to-date within the rest of the cluster.

      ```
      REFRESH CLUSTER(INVENTORY) REPOS(YES)
      ```

      Note: For large clusters, use of the **REFRESH CLUSTER** command can be disruptive to the cluster while it is in progress, and again at 27 day intervals thereafter when the cluster objects automatically send status updates to all interested queue managers. See Refreshing in a large cluster can affect performance and availability of the cluster.

ii) Restart the CLUSSDR channel (for example, using the START CHANNEL command).

iii) Restart the CLUSRCVR channel.

b) If the cluster is a publish/subscribe cluster, and the rejoining queue manager has subscriptions, issue the following command to ensure the proxy subscriptions are correctly synchronized across the cluster:

```
REFRESH QMGR TYPE(PROXYSUB)
```

5. Review the inventory application for message affinities.

   Before proceeding, ensure that the inventory application does not have any dependencies on the sequence of processing of messages and install the application on the system in Toronto.

6. Define the cluster queue INVENTQ.

   The INVENTQ queue, which is already hosted by the NEWYORK queue manager, is also to be hosted by TORONTO. Define it on the TORONTO queue manager as follows:

```
DEFINE QLOCAL(INVENTQ) CLUSTER(INVENTORY)
```

Results

Figure 36 on page 211 shows the INVENTORY cluster set up by this task.

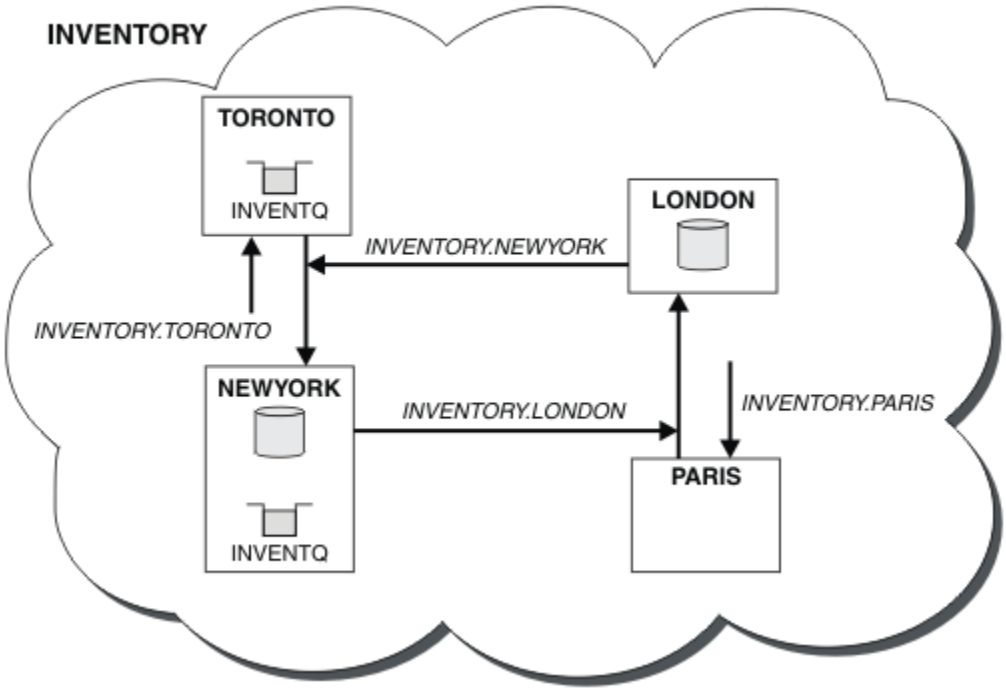


Figure 36. The INVENTORY cluster with four queue managers

The INVENTQ queue and the inventory application are now hosted on two queue managers in the cluster. This increases their availability, speeds up throughput of messages, and allows the workload to be distributed between the two queue managers. Messages put to INVENTQ by either TORONTO or NEWYORK are handled by the instance on the local queue manager whenever possible. Messages put by LONDON or PARIS are routed alternately to TORONTO or NEWYORK, so that the workload is balanced.

This modification to the cluster was accomplished without you having to alter the definitions on queue managers NEWYORK, LONDON, and PARIS. The full repositories in these queue managers are updated automatically with the information they need to be able to send messages to INVENTQ at TORONTO. The inventory application continues to function if one of the NEWYORK or the TORONTO queue manager

becomes unavailable, and it has sufficient capacity. The inventory application must be able to work correctly if it is hosted in both locations.

As you can see from the result of this task, you can have the same application running on more than one queue manager. You can clustering to distribution workload evenly.

An application might not be able to process records in both locations. For example, suppose that you decide to add a customer-account query and update application running in LONDON and NEWYORK. An account record can only be held in one place. You could decide to control the distribution of requests by using a data partitioning technique. You can split the distribution of the records. You could arrange for half the records, for example for account numbers 00000 - 49999, to be held in LONDON. The other half, in the range 50000 - 99999, are held in NEWYORK. You could then write a cluster workload exit program to examine the account field in all messages, and route the messages to the appropriate queue manager.

What to do next

Now that you have completed all the definitions, if you have not already done so start the channel initiator on IBM WebSphere MQ for z/OS. On all platforms, start a listener program on queue manager TORONTO. The listener program waits for incoming network requests and starts the cluster-receiver channel when it is needed.

Creating two-overlapping clusters with a gateway queue manager

Follow the instructions in the task to construct overlapping clusters with a gateway queue manager. Use the clusters as a starting point for the following examples of isolating messages to one application from messages to other applications in a cluster.

About this task

The example cluster configuration used to illustrate isolating cluster message traffic is shown in Figure 37 on page 213. The example is described in “Clustering: Application isolation using multiple cluster transmission queues” on page 277.

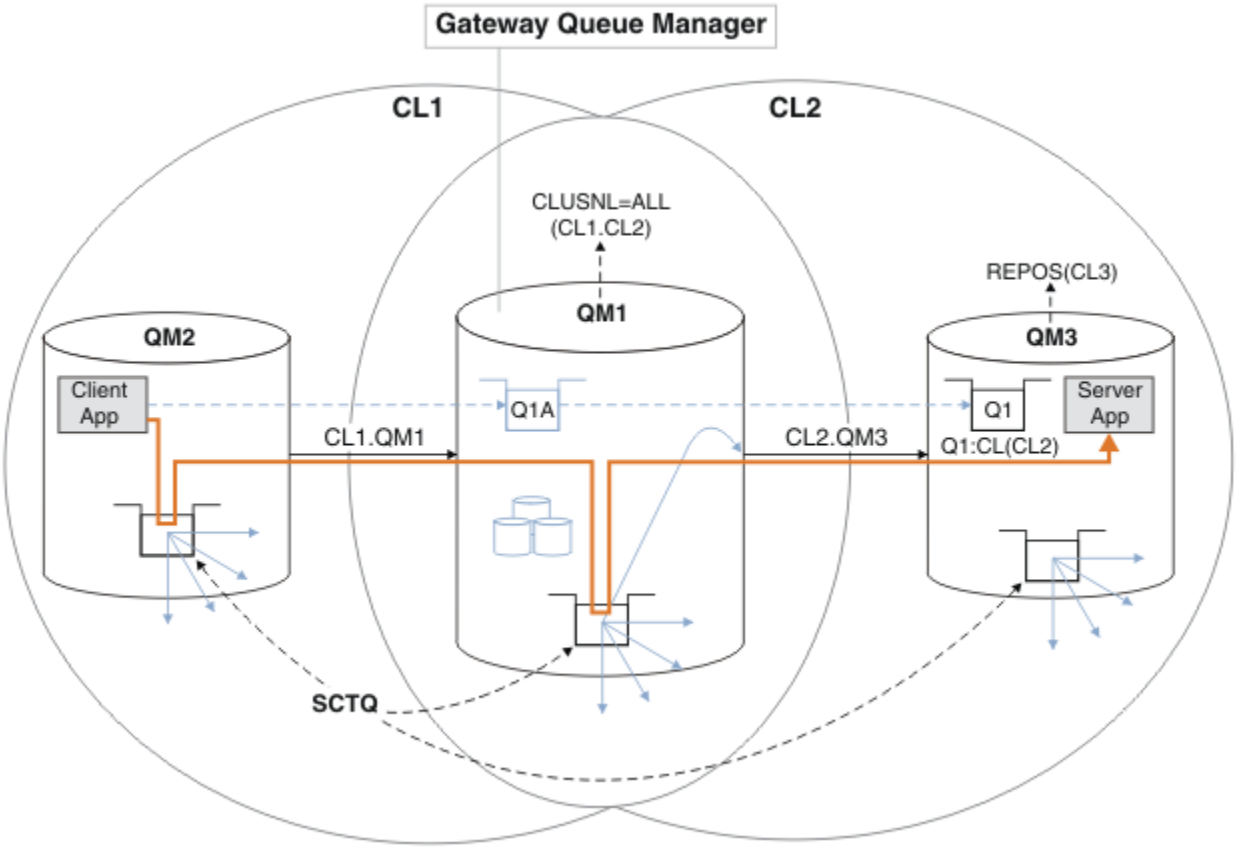


Figure 37. Client-server application deployed to hub and spoke architecture using IBM WebSphere MQ clusters

To make the number of steps to construct the example as few as possible, the configuration is kept simple, rather than realistic. The example might represent the integration of two clusters created by two separate organizations. For a more realistic scenario, see "Clustering: Planning how to configure cluster transmission queues" on page 281.

Follow the steps to construct the clusters. The clusters are used in the following examples of isolating the message traffic from the client application to the server application.

The instructions add a couple of extra queue managers so that each cluster has two repositories. The gateway queue manager is not used as a repository for performance reasons.

Procedure

1. Create and start the queue managers QM1, QM2, QM3, QM4, QM5.

```
crtmqm -sax -u SYSTEM.DEAD.LETTER.QUEUE QMn
strmqm QmgrName
```

   Note: QM4 and QM5 are the backup full repositories for the clusters.

2. Define and start listeners for each of the queue managers.

```
*... On QMn
DEFINE LISTENER(TCP141n) TRPTYPE(TCP) IPADDR(hostname) PORT(141n) CONTROL(QMGR) REPLACE
START LISTENER(TCP141n)
```

3. Create a cluster name list for all of the clusters.

```
*... On QM1
DEFINE NAMELIST(ALL) NAMES(CL1, CL2) REPLACE
```

4. Make QM2 and QM4 full repositories for CL1, QM3 and QM5 full repositories for CL2.

a) For CL1:

```
*... On QM2 and QM4
ALTER QMGR REPOS(CL1) DEFCLXQ(SCTQ)
```

b) For CL2:

```
*... On QM3 and QM5
ALTER QMGR REPOS(CL2) DEFCLXQ(SCTQ)
```

5. Add the cluster-sender and cluster-receiver channels for each queue manager and cluster.

   Run the following commands on QM2, QM3, QM4 and QM5, where *c*, *n*, and *m* take the values shown in Table 26 on page 214 for each queue manager:

Table 26. Parameter values for creating clusters 1 and 2

Queue manager	Cluster *c*	Other repository *n*	This repository *m*
QM2	1	4	2
QM4	1	2	4
QM3	2	5	3
QM5	2	3	5

```
*... On QMm
DEFINE CHANNEL(CLc.QMn) CHLTYPE(CLUSSDR)  CONNAME('localhost(141n)') CLUSTER(CLc) REPLACE
DEFINE CHANNEL(CLc.QMm) CHLTYPE(CLUSRCVR) CONNAME('localhost(141m)') CLUSTER(CLc) REPLACE
```

6. Add the gateway queue manager, QM1, to each of the clusters.

```
*... On QM1
DEFINE CHANNEL(CL1.QM2) CHLTYPE(CLUSSDR)  CONNAME('localhost(1412)') CLUSTER(CL1) REPLACE
DEFINE CHANNEL(CL1.QM1) CHLTYPE(CLUSRCVR) CONNAME('localhost(1411)') CLUSTER(CL1) REPLACE
DEFINE CHANNEL(CL2.QM3) CHLTYPE(CLUSSDR)  CONNAME('localhost(1413)') CLUSTER(CL2) REPLACE
DEFINE CHANNEL(CL2.QM1) CHLTYPE(CLUSRCVR) CONNAME('localhost(1411)') CLUSTER(CL2) REPLACE
```

7. Add the local queue Q1 to queue manager QM3 in cluster CL2.

```
*... On QM3
DEFINE QLOCAL(Q1) CLUSTER(CL2) REPLACE
```

8. Add the clustered queue manager alias Q1A to the gateway queue manager.

```
*... On QM1
DEFINE QALIAS(Q1A)  CLUSNL(ALL) TARGET(Q1) TARGTYPE(QUEUE) DEFBIND(NOTFIXED) REPLACE
```

   Note: Applications using the queue manager alias on any other queue manager but QM1, must specify DEFBIND(NOTFIXED) when they open the alias queue. **DEFBIND** specifies whether the routing information in the message header is fixed when the queue is opened by the application. If it is set to the default value, OPEN, messages are routed to Q1@QM1. Q1@QM1 does not exist, so messages from other queue managers end up on a dead letter queue. By setting the queue attribute to DEFBIND(NOTFIXED), applications such as **amqsput**, which default to the queue setting of **DEFBIND**, behave in the correct way.

9. Add the cluster queue manager alias definitions for all the clustered queue managers to the gateway queue manager, QM1.

```
*... On QM1
DEFINE QREMOTE(QM2) RNAME(' ') RQMNAME(QM2) CLUSNL(ALL) REPLACE
DEFINE QREMOTE(QM3) RNAME(' ') RQMNAME(QM3) CLUSNL(ALL) REPLACE
```

   Tip: The queue manager alias definitions on the gateway queue manager transfer messages that refer to a queue manager in another cluster; see Clustered queue manager aliases.

What to do next

1. Test the queue alias definition by sending a message from QM2 to Q1 on QM3 using the queue alias definition Q1A.

   a. Run the sample program **amqsput** on QM2 to put a message.

   ```
   C:\IBM\MQ>amqsput Q1A QM2
   Sample AMQSPUT0 start
   target queue is Q1A
   Sample request message from QM2 to Q1 using Q1A

   Sample AMQSPUT0 end
   ```

   b. Run the sample program **amqsget** to get the message from Q1 on QM3

   ```
   C:\IBM\MQ>amqsget Q1 QM3
   Sample AMQSGET0 start
   message <Sample request message from QM2 to Q1 using Q1A>
   no more messages
   Sample AMQSGET0 end
   ```

2. Test the queue manager alias definitions by sending a request message and receiving a reply message on a temporary-dynamic reply queue.

   The diagram shows the path taken by the reply message back to a temporary dynamic queue, which is called RQ. The server application, connected to QM3, opens the reply queue using the queue manager name QM2. The queue manager name QM2 is defined as a clustered queue manager alias on QM1. QM3 routes the reply message to QM1. QM1 routes the message to QM2.

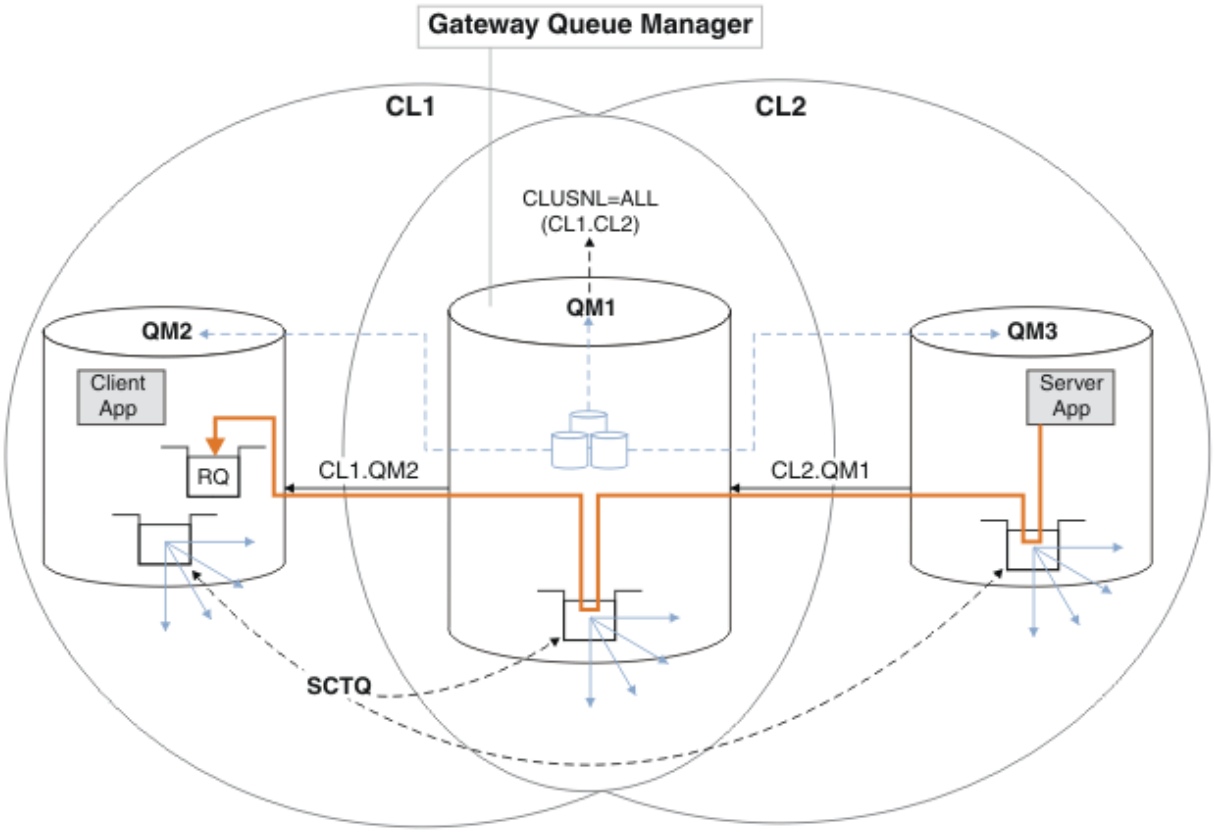


Figure 38. Using a queue manager alias to return the reply message to a different cluster

The way the routing works is as follows. Every queue manager in each cluster has a queue manager alias definition on QM1. The aliases are clustered in all the clusters. The grey dashed arrows from each

of the aliases to a queue manager show that each queue manager alias is resolved to a real queue manager in at least one of the clusters. In this case, the QM2 alias is clustered in both cluster CL1 and CL2, and is resolved to the real queue manager QM2 in CL1. The server application creates the reply message using the reply to queue name RQ, and reply to queue manager name QM2. The message is routed to QM1 because the queue manager alias definition QM2 is defined on QM1 in cluster CL2 and queue manager QM2 is not in cluster CL2. As the message cannot be sent to the target queue manager, it is sent to the queue manager that has the alias definition.

QM1 places the message on the cluster transmission queue on QM1 for transferal to QM2. QM1 routes the message to QM2 because the queue manager alias definition on QM1 for QM2 defines QM2 as the real target queue manager. The definition is not circular, because alias definitions can refer only to real definitions; the alias cannot point to itself. The real definition is resolved by QM1, because both QM1 and QM2 are in the same cluster, CL1. QM1 finds out the connection information for QM2 from the repository for CL1, and routes the message to QM2. For the message to be rerouted by QM1, the server application must have opened the reply queue with the option DEFBIND set to MQBND_BIND_NOT_FIXED. If the server application had opened the reply queue with the option MQBND_BIND_ON_OPEN, the message is not rerouted and ends up on a dead letter queue.

a. Create a clustered request queue with a trigger on QM3.

```
*... On QM3
DEFINE QLOCAL(QR) CLUSTER(CL2) TRIGGER INITQ(SYSTEM.DEFAULT.INITIATION.QUEUE)
PROCESS(ECHO) REPLACE
```

b. Create a clustered queue alias definition of QR on the gateway queue manager, QM1.

```
*... On QM1
DEFINE QALIAS(QRA) CLUSNL(ALL) TARGET(QR) TARGTYPE(QUEUE) DEFBIND(NOTFIXED) REPLACE
```

c. Create a process definition to start the sample echo program **amqsech** on QM3.

```
*... On QM3
DEFINE PROCESS(ECHO) APPLICID(AMQSECH) REPLACE
```

d. Create a model queue on QM2 for the sample program **amqsreq** to create the temporary-dynamic reply queue.

```
*... On QM2
DEFINE QMODEL(SYSTEM.SAMPLE.REPLY) REPLACE
```

e. Test the queue manager alias definition by sending a request from QM2 to QR on QM3 using the queue alias definition QRA.

i) Run the trigger monitor program on QM3.

```
runmqtrm -m QM3
```

The output is

```
C:\IBM\MQ>runmqtrm -m QM3
5724-H72 (C) Copyright IBM Corp. 1994, 2025. ALL RIGHTS RESERVED.
01/02/2012  16:17:15: WebSphere MQ trigger monitor started.


_________________________________________________
01/02/2012  16:17:15: Waiting for a trigger message
```

ii) Run the sample program **amqsreq** on QM2 to put a request and wait for a reply.

```
C:\IBM\MQ>amqsreq QRA QM2
Sample AMQSREQ0 start
server queue is QRA
replies to 4F2961C802290020
A request message from QM2 to QR on QM3
```

```
response <A request message from QM2 to QR on QM3>
no more replies
Sample AMQSREQ0 end
```

Related concepts

“Access control and multiple cluster transmission queues” on page 156
Choose between three modes of checking when an application puts messages to remote cluster queues. The modes are checking remotely against the cluster queue, checking locally against `SYSTEM.CLUSTER.TRANSMIT.QUEUE`, or checking against local profiles for the cluster queue, or cluster queue manager.

“Clustering: Application isolation using multiple cluster transmission queues” on page 277
You can isolate the message flows between queue managers in a cluster. You can place messages being transported by different cluster-sender channels onto different cluster transmission queues. You can use the approach in a single cluster or with overlapping clusters. The topic provides examples and some best practices to guide you in choosing an approach to use.

Related tasks

“Adding a queue manager to a cluster: separate transmission queues” on page 193
Follow these instructions to add a queue manager to the cluster you created. Messages to cluster queues and topics are transferred using multiple cluster transmission queues.

“Creating two-overlapping clusters with a gateway queue manager” on page 212
Follow the instructions in the task to construct overlapping clusters with a gateway queue manager. Use the clusters as a starting point for the following examples of isolating messages to one application from messages to other applications in a cluster.

“Adding a remote queue definition to isolate messages sent from a gateway queue manager” on page 195
Modify the configuration of overlapping clusters that use a gateway queue manager. After the modification messages are transferred to an application from the gateway queue manager without using the same transmission queue or channels as other cluster messages. The solution uses a clustered queue remote definition, and a separate sender channel and transmission queue.

“Changing the default to separate cluster transmission queues to isolate message traffic” on page 218
You can change the default way a queue manager stores messages for a clustered queue or topic on a transmission queue. Changing the default provides you with a way to isolate cluster messages on a gateway queue manager.

“Adding a cluster transmit queue to isolate cluster message traffic sent from a gateway queue manager” on page 198
Modify the configuration of overlapping clusters that use a gateway queue manager. After the modification messages are transferred to an application from the gateway queue manager without using the same transmission queue or channels as other cluster messages. The solution uses an additional cluster transmission queue to separate message traffic to a single queue manager in a cluster.

“Adding a cluster and a cluster transmit queue to isolate cluster message traffic sent from a gateway queue manager” on page 201
Modify the configuration of overlapping clusters that use a gateway queue manager. After the modification messages are transferred to an application from the gateway queue manager without using the same transmission queue or channels as other cluster messages. The solution uses an additional cluster to isolate the messages to a particular cluster queue.

“Clustering: Planning how to configure cluster transmission queues” on page 281

You are guided through the choices of cluster transmission queues. You can configure one common default queue, separate default queues, or manually defined queues. Configuring multiple cluster transmission queues applies to platforms other than z/OS.

Changing the default to separate cluster transmission queues to isolate message traffic

You can change the default way a queue manager stores messages for a clustered queue or topic on a transmission queue. Changing the default provides you with a way to isolate cluster messages on a gateway queue manager.

Before you begin

1. The gateway queue manager must be on Version 7.5, or later, and on a platform other than z/OS.
2. Construct the overlapping clusters shown in Figure 37 on page 213 in “Creating two-overlapping clusters with a gateway queue manager” on page 212 by following the steps in that task.

About this task

To implement the architecture with multiple clusters queue, your gateway queue manager must be on Version 7.5, or later. All you do to use multiple cluster transmission queues is to change the default cluster transmission queue type on the gateway queue manager. Change the value of the queue manager attribute **DEFCLXQ** on `QM1` from `SCTQ` to `CHANNEL`; see Figure 39 on page 219. The diagram shows one message flow. For flows to other queue managers, or to other clusters, the queue manager creates additional permanent dynamic cluster transmission queues. Each cluster-sender channel transfers messages from a different cluster transmission queue.

The change does not take effect immediately, unless you are connecting the gateway queue manager to clusters for the first time. The task includes steps for the typical case of managing a change to an existing configuration. To set up a queue manager to use separate cluster transmission queues when it first joins a cluster; see “Adding a queue manager to a cluster: separate transmission queues” on page 193.

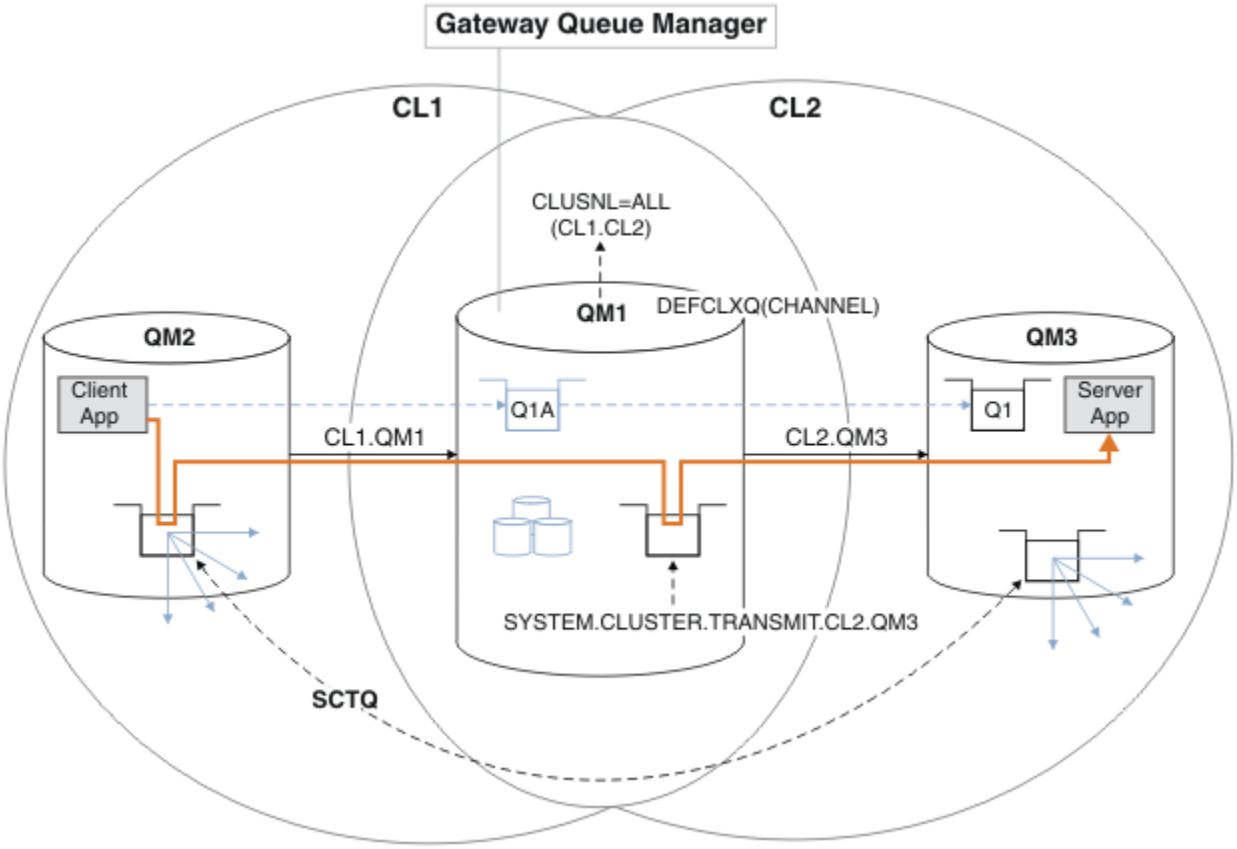


Figure 39. Client-server application deployed to hub and spoke architecture with separate cluster transmission queues on the gateway queue manager.

Procedure

1. Change the gateway queue manager to use separate cluster transmission queues.

```
*... On QM1
ALTER QMGR DEFCLXQ(CHANNEL)
```

2. Switch to the separate cluster transmission queues.

   Any cluster-sender channel that is not running switches to using separate cluster transmission queues when it next starts.

   To switch the running channels, either restart the queue manager, or follow these steps:

   a) List the cluster-sender channels that are running with `SYSTEM.CLUSTER.TRANSMIT.QUEUE`.

```
*... On QM1
DISPLAY CHSTATUS(*) WHERE(XMITQ EQ 'SYSTEM.CLUSTER.TRANSMIT.QUEUE')
```

   The response is list of channel status reports:

```
AMQ8417: Display Channel Status details.
   CHANNEL(CL1.QM2)            CHLTYPE(CLUSSDR)
   CONNAME(127.0.0.1(1412))    CURRENT
   RQMNAME(QM2)                STATUS(RUNNING)
   SUBSTATE(MQGET)             XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)
AMQ8417: Display Channel Status details.
   CHANNEL(CL2.QM3)            CHLTYPE(CLUSSDR)
   CONNAME(127.0.0.1(1413))    CURRENT
   RQMNAME(QM3)                STATUS(RUNNING)
```

```
   SUBSTATE(MQGET)             XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)
AMQ8417: Display Channel Status details.
   CHANNEL(CL2.QM5)            CHLTYPE(CLUSSDR)
   CONNAME(127.0.0.1(1415))    CURRENT
   RQMNAME(QM5)                STATUS(RUNNING)
   SUBSTATE(MQGET)             XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)
AMQ8417: Display Channel Status details.
   CHANNEL(CL1.QM4)            CHLTYPE(CLUSSDR)
   CONNAME(127.0.0.1(1414))    CURRENT
   RQMNAME(QM4)                STATUS(RUNNING)
   SUBSTATE(MQGET)             XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)
```

b) Stop the channels that are running

For each channel in the list, run the command:

```
*... On QM1
STOP CHANNEL(ChannelName)
```

Where *ChannelName* is each of `CL1.QM2, CL1.QM4, CL1.QM3, CL1.QM5`.

The response is that the command is accepted:

```
AMQ8019: Stop WebSphere MQ channel accepted.
```

c) Monitor which channels are stopped

```
*... On QM1
DISPLAY CHSTATUS(*) WHERE(XMITQ EQ 'SYSTEM.CLUSTER.TRANSMIT.QUEUE')
```

The response is a list of channels that are still running and channels that are stopped:

```
AMQ8417: Display Channel Status details.
   CHANNEL(CL1.QM2)            CHLTYPE(CLUSSDR)
   CONNAME(127.0.0.1(1412))    CURRENT
   RQMNAME(QM2)                STATUS(STOPPED)
   SUBSTATE( )                 XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)
AMQ8417: Display Channel Status details.
   CHANNEL(CL2.QM3)            CHLTYPE(CLUSSDR)
   CONNAME(127.0.0.1(1413))    CURRENT
   RQMNAME(QM3)                STATUS(STOPPED)
   SUBSTATE( )                 XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)
AMQ8417: Display Channel Status details.
   CHANNEL(CL2.QM5)            CHLTYPE(CLUSSDR)
   CONNAME(127.0.0.1(1415))    CURRENT
   RQMNAME(QM5)                STATUS(STOPPED)
   SUBSTATE( )                 XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)
AMQ8417: Display Channel Status details.
   CHANNEL(CL1.QM4)            CHLTYPE(CLUSSDR)
   CONNAME(127.0.0.1(1414))    CURRENT
   RQMNAME(QM4)                STATUS(STOPPED)
   SUBSTATE( )                 XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)
```

d) Start each stopped channel.

Do this step for all the channels that were running. If a channel does not stop, you can run the **STOP CHANNEL** command again with the FORCE option. An example of setting the FORCE option would be if the channel does not stop, and you cannot restart the other queue manager to synchronize the channel.

```
*... On QM1
START CHANNEL(CL2.QM5)
```

The response is that the command is accepted:

```
AMQ8018: Start WebSphere MQ channel accepted.
```

e) Monitor the transmission queues being switched.

Monitor the gateway queue manager error log for the message "AMQ7341 The `transmission queue for channel CL2.QM3 is SYSTEM.CLUSTER.TRANSMIT.`*QUEUE*|*CL2.QM3*".

f) Check that SYSTEM.CLUSTER.TRANSMIT.QUEUE is no longer used

```
*... On QM1
DISPLAY CHSTATUS(*) WHERE(XMITQ EQ 'SYSTEM.CLUSTER.TRANSMIT.QUEUE')
DISPLAY QUEUE(SYSTEM.CLUSTER.TRANSMIT.QUEUE) CURDEPTH
```

The response is list of channel status reports, and the depth of SYSTEM.CLUSTER.TRANSMIT.QUEUE:

```
AMQ8420: Channel Status not found.
AMQ8409: Display Queue details.
   QUEUE(SYSTEM.CLUSTER.TRANSMIT.QUEUE)    TYPE(QLOCAL)
   CURDEPTH(0)
```

g) Monitor which channels are started

```
*... On QM1
DISPLAY CHSTATUS(*) WHERE(XMITQ LK 'SYSTEM.CLUSTER.TRANSMIT.*')
```

The response is a list of the channels, in this case already running with the new default cluster transmission queues:

```
AMQ8417: Display Channel Status details.
   CHANNEL(CL1.QM2)                        CHLTYPE(CLUSSDR)
   CONNAME(127.0.0.1(1412))                CURRENT
   RQMNAME(QM2)                            STATUS(RUNNING)
   SUBSTATE(MQGET)
   XMITQ(SYSTEM.CLUSTER.TRANSMIT.CL1.QM2)
AMQ8417: Display Channel Status details.
   CHANNEL(CL2.QM3)                        CHLTYPE(CLUSSDR)
   CONNAME(127.0.0.1(1413))                CURRENT
   RQMNAME(QM3)                            STATUS(RUNNING)
   SUBSTATE(MQGET)
   XMITQ(SYSTEM.CLUSTER.TRANSMIT.CL2.QM3)
AMQ8417: Display Channel Status details.
   CHANNEL(CL2.QM5)                        CHLTYPE(CLUSSDR)
   CONNAME(127.0.0.1(1415))                CURRENT
   RQMNAME(QM5)                            STATUS(RUNNING)
   SUBSTATE(MQGET)
   XMITQ(SYSTEM.CLUSTER.TRANSMIT.CL2.QM5)
AMQ8417: Display Channel Status details.
   CHANNEL(CL1.QM4)                        CHLTYPE(CLUSSDR)
   CONNAME(127.0.0.1(1414))                CURRENT
   RQMNAME(QM4)                            STATUS(RUNNING)
   SUBSTATE(MQGET)
   XMITQ(SYSTEM.CLUSTER.TRANSMIT.CL1.QM4)
```

What to do next

1. Test the automatically defined cluster transmission queue by sending a message from QM2 to Q1 on QM3, resolving queue name with the queue alias definition Q1A

   a. Run the sample program **amqsput** on QM2 to put a message.

```
C:\IBM\MQ>amqsput Q1A QM2
Sample AMQSPUT0 start
```

```
target queue is Q1A
Sample request message from QM2 to Q1 using Q1A

Sample AMQSPUT0 end
```

b. Run the sample program **amqsget** to get the message from Q1 on QM3

```
C:\IBM\MQ>amqsget Q1 QM3
Sample AMQSGET0 start
message <Sample request message from QM2 to Q1 using Q1A>
no more messages
Sample AMQSGET0 end
```

2. Consider whether to reconfigure security, by configuring security for the cluster queues on the queue managers where messages for the cluster queues originate.

Related concepts

“Access control and multiple cluster transmission queues” on page 156
Choose between three modes of checking when an application puts messages to remote cluster queues. The modes are checking remotely against the cluster queue, checking locally against SYSTEM.CLUSTER.TRANSMIT.QUEUE, or checking against local profiles for the cluster queue, or cluster queue manager.

“Clustering: Application isolation using multiple cluster transmission queues” on page 277
You can isolate the message flows between queue managers in a cluster. You can place messages being transported by different cluster-sender channels onto different cluster transmission queues. You can use the approach in a single cluster or with overlapping clusters. The topic provides examples and some best practices to guide you in choosing an approach to use.

Related tasks

“Adding a queue manager to a cluster: separate transmission queues” on page 193
Follow these instructions to add a queue manager to the cluster you created. Messages to cluster queues and topics are transferred using multiple cluster transmission queues.

“Creating two-overlapping clusters with a gateway queue manager” on page 212
Follow the instructions in the task to construct overlapping clusters with a gateway queue manager. Use the clusters as a starting point for the following examples of isolating messages to one application from messages to other applications in a cluster.

“Adding a remote queue definition to isolate messages sent from a gateway queue manager” on page 195
Modify the configuration of overlapping clusters that use a gateway queue manager. After the modification messages are transferred to an application from the gateway queue manager without using the same transmission queue or channels as other cluster messages. The solution uses a clustered queue remote definition, and a separate sender channel and transmission queue.

“Changing the default to separate cluster transmission queues to isolate message traffic” on page 218
You can change the default way a queue manager stores messages for a clustered queue or topic on a transmission queue. Changing the default provides you with a way to isolate cluster messages on a gateway queue manager.

“Adding a cluster transmit queue to isolate cluster message traffic sent from a gateway queue manager” on page 198
Modify the configuration of overlapping clusters that use a gateway queue manager. After the modification messages are transferred to an application from the gateway queue manager without using the same transmission queue or channels as other cluster messages. The solution uses an additional cluster transmission queue to separate message traffic to a single queue manager in a cluster.

“Adding a cluster and a cluster transmit queue to isolate cluster message traffic sent from a gateway queue manager” on page 201
Modify the configuration of overlapping clusters that use a gateway queue manager. After the modification messages are transferred to an application from the gateway queue manager without using the same

transmission queue or channels as other cluster messages. The solution uses an additional cluster to isolate the messages to a particular cluster queue.

"Clustering: Planning how to configure cluster transmission queues" on page 281
You are guided through the choices of cluster transmission queues. You can configure one common default queue, separate default queues, or manually defined queues. Configuring multiple cluster transmission queues applies to platforms other than z/OS.

Removing a cluster queue from a queue manager

Disable the INVENTQ queue at Toronto. Send all the inventory messages to New York, and delete the INVENTQ queue at Toronto when it is empty.

Before you begin

Note: For changes to a cluster to be propagated throughout the cluster, at least one full repository must always be available. Ensure that your repositories are available before starting this task.

Scenario:

- The INVENTORY cluster has been set up as described in "Adding a queue manager that hosts a queue" on page 209. It contains four queue managers. LONDON and NEWYORK both hold full repositories. PARIS and TORONTO hold partial repositories. The inventory application runs on the systems in New York and Toronto and is driven by the arrival of messages on the INVENTQ queue.
- Because of reduced workload, you no longer want to run the inventory application in Toronto. You want to disable the INVENTQ queue hosted by the queue manager TORONTO, and have TORONTO feed messages to the INVENTQ queue in NEWYORK.
- Network connectivity exists between all four systems.
- The network protocol is TCP.

About this task

Follow these steps to remove a cluster queue.

Procedure

1. Indicate that the queue is no longer available.

   To remove a queue from a cluster, remove the cluster name from the local queue definition. Alter the INVENTQ on TORONTO so that it is not accessible from the rest of the cluster:

   ```
   ALTER QLOCAL(INVENTQ) CLUSTER(' ')
   ```

2. Check that the queue is no longer available.

   On a full repository queue manager, either LONDON or NEWYORK, check that the queue is no longer hosted by queue manager TORONTO by issuing the following command:

   ```
   DIS QCLUSTER (INVENTQ)
   ```

   TORONTO is not listed in the results, if the ALTER command has completed successfully.

3. Disable the queue.

   Disable the INVENTQ queue at TORONTO so that no further messages can be written to it:

   ```
   ALTER QLOCAL(INVENTQ) PUT(DISABLED)
   ```

   Now messages in transit to this queue using MQOO_BIND_ON_OPEN go to the dead-letter queue. You need to stop all applications from putting messages explicitly to the queue on this queue manager.

4. Monitor the queue until it is empty.

Monitor the queue using the DISPLAY QUEUE command, specifying the attributes IPPROCS, OPPROCS, and CURDEPTH, or use the **WRKMQMSTS** command on IBM i. When the number of input and output processes, and the current depth of the queuesare all zero, the queue is empty.

5. Monitor the channel to ensure there are no in-doubt messages.

   To be sure that there are no in-doubt messages on the channel INVENTORY.TORONTO, monitor the cluster-sender channel called INVENTORY.TORONTO on each of the other queue managers. Issue the DISPLAY CHSTATUS command specifying the INDOUBT parameter from each queue manager:

   ```
   DISPLAY CHSTATUS(INVENTORY.TORONTO) INDOUBT
   ```

   If there are any in-doubt messages, you must resolve them before proceeding. For example, you might try issuing the RESOLVE channel command or stopping and restarting the channel.

6. Delete the local queue.

   When you are satisfied that there are no more messages to be delivered to the inventory application at TORONTO, you can delete the queue:

   ```
   DELETE QLOCAL(INVENTQ)
   ```

7. You can now remove the inventory application from the system in Toronto

   Removing the application avoids duplication and saves space on the system.

Results

The cluster set up by this task is like that setup by the previous task. The difference is the INVENTQ queue is no longer available at queue manager TORONTO.

When you took the queue out of service in step 1, the TORONTO queue manager sent a message to the two full repository queue managers. It notified them of the change in status. The full repository queue managers pass on this information to other queue managers in the cluster that have requested updates to information concerning the INVENTQ.

When a queue manager puts a message on the INVENTQ queue the updated partial repository indicates that the INVENTQ queue is available only at the NEWYORK queue manager. The message is sent to the NEWYORK queue manager.

What to do next

In this task, there was only one queue to remove and only one cluster to remove it from.

Suppose that there are many queues referring to a namelist containing many cluster names. For example, the TORONTO queue manager might host not only the INVENTQ , but also the PAYROLLQ, SALESQ , and PURCHASESQ. TORONTO makes these queues available in all the appropriate clusters, INVENTORY , PAYROLL, SALES, and PURCHASES . Define a namelist of the cluster names on the TORONTO queue manager:

```
DEFINE NAMELIST(TOROLIST)
       DESCR('List of clusters TORONTO is in')
       NAMES(INVENTORY, PAYROLL, SALES, PURCHASES)
```

Add the namelist to each queue definition:

```
DEFINE QLOCAL(INVENTQ) CLUSNL(TOROLIST)
DEFINE QLOCAL(PAYROLLQ) CLUSNL(TOROLIST)
DEFINE QLOCAL(SALESQ) CLUSNL(TOROLIST)
DEFINE QLOCAL(PURCHASESQ) CLUSNL(TOROLIST)
```

Now suppose that you want to remove all those queues from the SALES cluster, because the SALES operation is to be taken over by the PURCHASES operation. All you need to do is alter the TOROLIST namelist to remove the name of the SALES cluster from it.

If you want to remove a single queue from one of the clusters in the namelist, create a namelist, containing the remaining list of cluster names. Then alter the queue definition to use the new namelist. To remove the PAYROLLQ from the INVENTORY cluster:

1. Create a namelist:

```
DEFINE NAMELIST(TOROSHORTLIST)
       DESCR('List of clusters TORONTO is in other than INVENTORY')
       NAMES(PAYROLL, SALES, PURCHASES)
```

2. Alter the PAYROLLQ queue definition:

```
ALTER QLOCAL(PAYROLLQ) CLUSNL(TOROSHORTLIST)
```

Moving a full repository to another queue manager

Move a full repository from one queue manager to another, building up the new repository from information held at the second repository.

Before you begin

Note: For changes to a cluster to be propagated throughout the cluster, at least one full repository must always be available. Ensure that your repositories are available before starting this task.

Scenario:

- The INVENTORY cluster has been set up as described in "Adding a queue manager to a cluster" on page 191.
- For business reasons you now want to remove the full repository from queue manager LONDON, and replace it with a full repository at queue manager PARIS. The NEWYORK queue manager is to continue holding a full repository.

About this task

Follow these steps to move a full repository to another queue manager.

Procedure

1. Alter PARIS to make it a full repository queue manager.

   On PARIS, issue the following command:

```
ALTER QMGR REPOS(INVENTORY)
```

2. Add a CLUSSDR channel on PARIS

   PARIS currently has a cluster-sender channel pointing to LONDON. LONDON is no longer to hold a full repository for the cluster. PARIS must have a new cluster-sender channel that points to NEWYORK, where the other full repository is now held.

```
DEFINE CHANNEL(INVENTORY.NEWYORK) CHLTYPE(CLUSSDR) TRPTYPE(TCP)
CONNAME(NEWYORK.CHSTORE.COM) CLUSTER(INVENTORY)
DESCR('Cluster-sender channel from PARIS to repository at NEWYORK')
```

3. Define a CLUSSDR channel on NEWYORK that points to PARIS

   Currently NEWYORK has a cluster-sender channel pointing to LONDON. Now that the other full repository has moved to PARIS, you need to add a new cluster-sender channel at NEWYORK that points to PARIS.

```
DEFINE CHANNEL(INVENTORY.PARIS) CHLTYPE(CLUSSDR) TRPTYPE(TCP)
CONNAME(PARIS.CHSTORE.COM) CLUSTER(INVENTORY)
DESCR('Cluster-sender channel from NEWYORK to repository at PARIS')
```

When you add the cluster-sender channel to PARIS, PARIS learns about the cluster from NEWYORK. It builds up its own full repository using the information from NEWYORK.

4. Check that queue manager PARIS now has a full repository

   Check that queue manager PARIS has built its own full repository from the full repository on queue manager NEWYORK. Issue the following commands:

   ```
   DIS QCLUSTER(*) CLUSTER (INVENTORY)
   DIS CLUSQMGR(*) CLUSTER (INVENTORY)
   ```

   Check that these commands show details of the same resources in this cluster as on NEWYORK.

   Note: If queue manager NEWYORK is not available, this building of information cannot complete. Do not move on to the next step until the task is complete.

5. Alter the queue-manager definition on LONDON

   Finally alter the queue manager at LONDON so that it no longer holds a full repository for the cluster. On LONDON, issue the command:

   ```
   ALTER QMGR REPOS(' ')
   ```

   The queue manager no longer receives any cluster information. After 30 days the information that is stored in its full repository expires. The queue manager LONDON now builds up its own partial repository.

6. Remove or change any outstanding definitions.

   When you are sure that the new arrangement of your cluster is working as expected, remove or change manually defined CLUSSDR definitions that are no longer correct.

   - On the PARIS queue manager, you must stop and delete the cluster-sender channel to LONDON, and then issue the start channel command so that the cluster can use the automatic channels again:

     ```
     STOP CHANNEL(INVENTORY.LONDON)
     DELETE CHANNEL(INVENTORY.LONDON)
     START CHANNEL(INVENTORY.LONDON)
     ```

   - On the NEWYORK queue manager, you must stop and delete the cluster-sender channel to LONDON, and then issue the start channel command so that the cluster can use the automatic channels again:

     ```
     STOP CHANNEL(INVENTORY.LONDON)
     DELETE CHANNEL(INVENTORY.LONDON)
     START CHANNEL(INVENTORY.LONDON)
     ```

   - Replace all other cluster-sender channels in the cluster that point to LONDON with channels that point to either NEWYORK or PARIS. In this small example, there are no others. To check whether there are any others that you have forgotten, issue the DISPLAY CHANNEL command from each queue manager, specifying TYPE(CLUSSDR). For example:

     ```
     DISPLAY CHANNEL(*) TYPE(CLUSSDR)
     ```

   It is important that you perform this task as soon as possible after moving the full repository from LONDON to PARIS. In the time before you perform this task, queue managers that have manually defined CLUSSDR channels named INVENTORY.LONDON might send requests for information using this channel.

   After LONDON has ceased to be a full repository, if it receives such requests it will write error messages to its queue manager error log. The following examples show which error messages might be seen on LONDON:

   - AMQ9428: Unexpected publication of a cluster queue object received
   - AMQ9432: Query received by a non-repository queue manager

The queue manager LONDON does not respond to the requests for information because it is no longer a full repository. The queue managers requesting information from LONDON must rely on NEWYORK for cluster information until their manually defined CLUSSDR definitions are corrected to point to PARIS. This situation must not be tolerated as a valid configuration in the long term.

Results

Figure 40 on page 227 shows the cluster set up by this task.

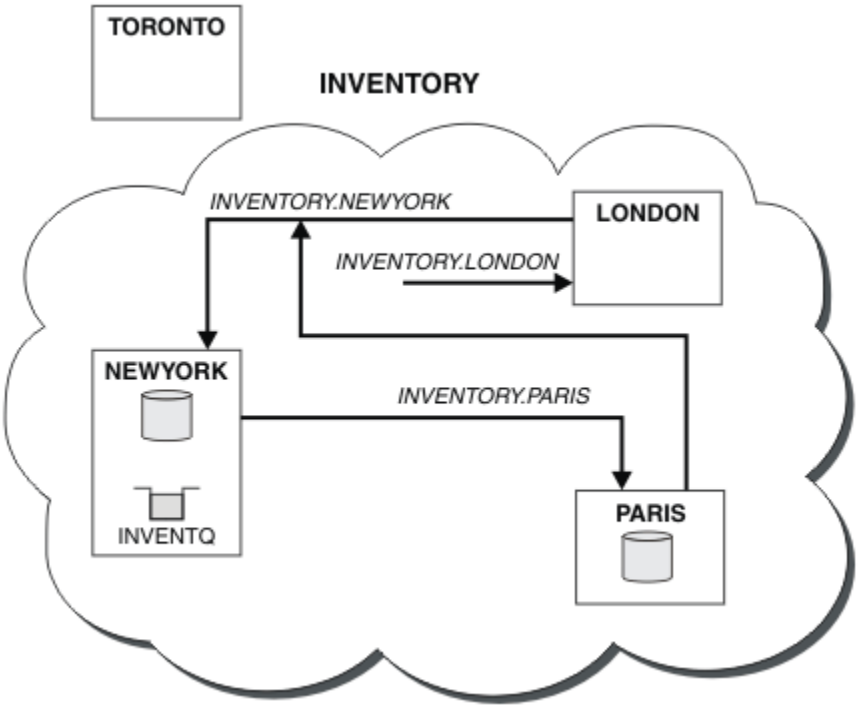


Figure 40. The INVENTORY cluster with the full repository moved to PARIS

Converting an existing network into a cluster

Convert an existed distributed queuing network to a cluster and add an additional queue manager to increase capacity.

Before you begin

In "Setting up a new cluster" on page 181 through "Moving a full repository to another queue manager" on page 225 you created and extended a new cluster. The next two tasks explore a different approach: that of converting an existing network of queue managers into a cluster.

Note: For changes to a cluster to be propagated throughout the cluster, at least one full repository must always be available. Ensure that your repositories are available before starting this task.

Scenario:

- An IBM WebSphere MQ network is already in place, connecting the nationwide branches of a chain store. It has a hub and spoke structure: all the queue managers are connected to one central queue manager. The central queue manager is on the system on which the inventory application runs. The application is driven by the arrival of messages on the INVENTQ queue, for which each queue manager has a remote-queue definition.

  This network is illustrated in Figure 41 on page 228.

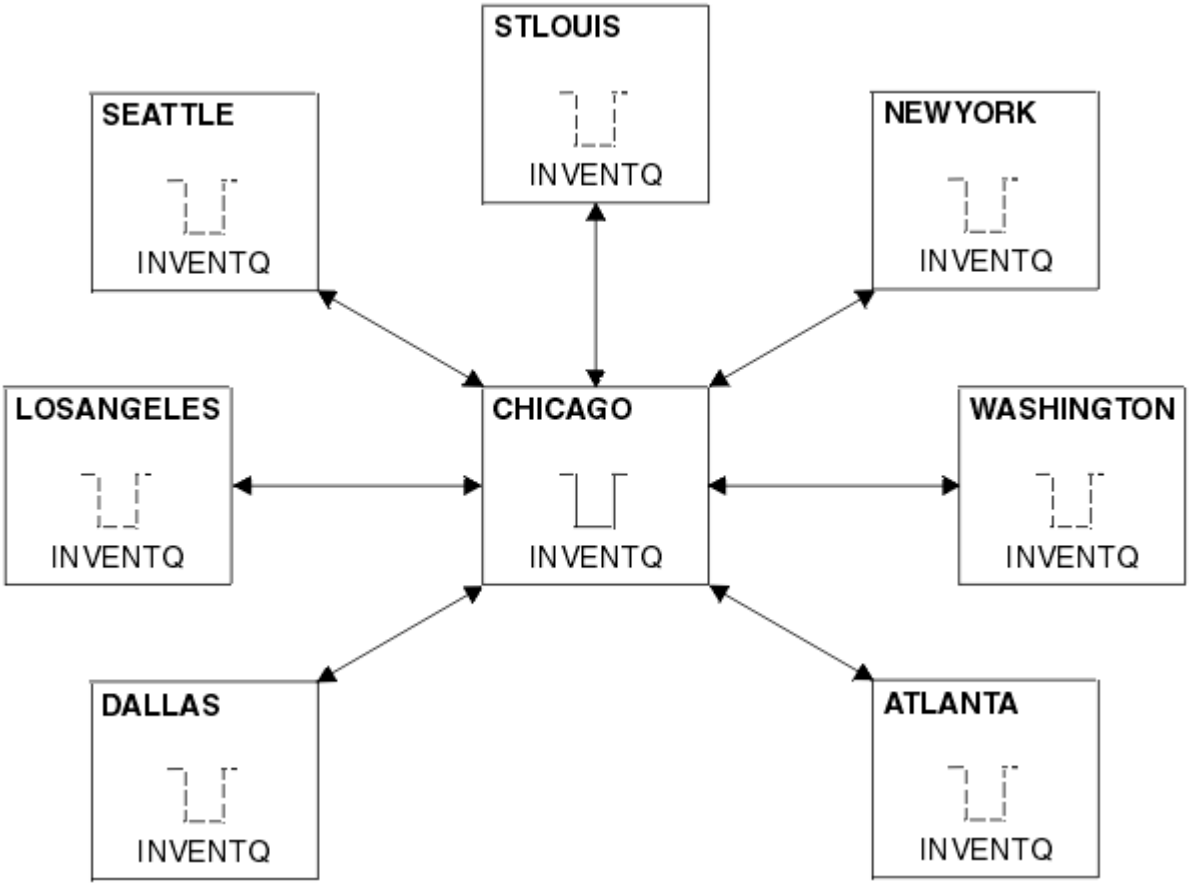


Figure 41. A hub and spoke network

- To ease administration you are going to convert this network into a cluster and create another queue manager at the central site to share the workload.

  The cluster name is `CHNSTORE`.

  Note: The cluster name `CHNSTORE` was selected to allow cluster-receiver channel names to be created using names in the format *`cluster-name.queue-manager`* that do not exceed the maximum length of 20 characters, for example `CHNSTORE.WASHINGTON`.
- Both the central queue managers are to host full repositories and be accessible to the inventory application.
- The inventory application is to be driven by the arrival of messages on the `INVENTQ` queue hosted by either of the central queue managers.
- The inventory application is to be the only application running in parallel and accessible by more than one queue manager. All other applications continue to run as before.
- All the branches have network connectivity to the two central queue managers.
- The network protocol is TCP.

About this task

Follow these steps to convert an existing network into a cluster.

Procedure

1. Review the inventory application for message affinities.

   Before proceeding ensure that the application can handle message affinities. Message affinities are the relationship between conversational messages that are exchanged between two applications, where

the messages must be processed by a particular queue manager or in a particular sequence. For more information on message affinities, see: “Handling message affinities” on page 270

2. Alter the two central queue managers to make them full repository queue managers.

   The two queue managers CHICAGO and CHICAGO2 are at the hub of this network. You have decided to concentrate all activity associated with the chain store cluster on to those two queue managers. As well as the inventory application and the definitions for the INVENTQ queue, you want these queue managers to host the two full repositories for the cluster. At each of the two queue managers, issue the following command:

   ```
   ALTER QMGR REPOS(CHNSTORE)
   ```

3. Define a CLUSRCVR channel on each queue manager.

   At each queue manager in the cluster, define a cluster-receiver channel and a cluster-sender channel. It does not matter which channel you define first.

   Make a CLUSRCVR definition to advertise each queue manager, its network address, and other information, to the cluster. For example, on queue manager ATLANTA:

   ```
   DEFINE CHANNEL(CHNSTORE.ATLANTA) CHLTYPE(CLUSRCVR) TRPTYPE(TCP)
   CONNAME(ATLANTA.CHSTORE.COM) CLUSTER(CHNSTORE)
   DESCR('Cluster-receiver channel')
   ```

4. Define a CLUSSDR channel on each queue manager

   Make a CLUSSDR definition at each queue manager to link that queue manager to one or other of the full repository queue managers. For example, you might link ATLANTA to CHICAGO2:

   ```
   DEFINE CHANNEL(CHNSTORE.CHICAGO2) CHLTYPE(CLUSSDR) TRPTYPE(TCP)
   CONNAME(CHICAGO2.CHSTORE.COM) CLUSTER(CHNSTORE)
   DESCR('Cluster-sender channel to repository queue manager')
   ```

5. Install the inventory application on CHICAGO2.

   You already have the inventory application on queue manager CHICAGO. Now you need to make a copy of this application on queue manager CHICAGO2.

6. Define the INVENTQ queue on the central queue managers.

   On CHICAGO, modify the local queue definition for the queue INVENTQ to make the queue available to the cluster. Issue the command:

   ```
   ALTER QLOCAL(INVENTQ) CLUSTER(CHNSTORE)
   ```

   On CHICAGO2, make a definition for the same queue:

   ```
   DEFINE QLOCAL(INVENTQ) CLUSTER(CHNSTORE)
   ```

   On z/OS, you can use the MAKEDEF option of the COMMAND function of **CSQUTIL** to make an exact copy on CHICAGO2 of the INVENTQ on CHICAGO.

   When you make these definitions, a message is sent to the full repositories at CHICAGO and CHICAGO2 and the information in them is updated. The queue manager finds out from the full repositories when it puts a message to the INVENTQ, that there is a choice of destinations for the messages.

7. Check that the cluster changes have been propagated.

   Check that the definitions you created in the previous step have been propagated though the cluster. Issue the following command on a full repository queue manager:

   ```
   DIS QCLUSTER(INVENTQ)
   ```

Adding a new, interconnected cluster

Add a new cluster that shares some queue managers with an existing cluster.

Before you begin

Note:

1. For changes to a cluster to be propagated throughout the cluster, at least one full repository must always be available. Ensure that your repositories are available before starting this task.
2. Before starting this task, check for queue-name clashes and understand the consequences. You might need to rename a queue, or set up queue aliases before you can proceed.

Scenario:

- A WebSphere MQ cluster has been set up as described in “Converting an existing network into a cluster” on page 227 .
- A new cluster called MAILORDER is to be implemented. This cluster comprises four of the queue managers that are in the CHNSTORE cluster; CHICAGO, CHICAGO2 , SEATTLE, and ATLANTA, and two additional queue managers; HARTFORD and OMAHA . The MAILORDER application runs on the system at Omaha, connected to queue manager OMAHA. It is driven by the other queue managers in the cluster putting messages on the MORDERQ queue.
- The full repositories for the MAILORDER cluster are maintained on the two queue managers CHICAGO and CHICAGO2.
- The network protocol is TCP.

About this task

Follow these steps to add a new, interconnected cluster.

Procedure

1. Create a namelist of the cluster names.

   The full repository queue managers at CHICAGO and CHICAGO2 are now going to hold the full repositories for both of the clusters CHNSTORE and MAILORDER . First, create a namelist containing the names of the clusters. Define the namelist on CHICAGO and CHICAGO2 , as follows:

   ```
   DEFINE NAMELIST(CHAINMAIL)
          DESCR('List of cluster names')
          NAMES(CHNSTORE, MAILORDER)
   ```

2. Alter the two queue-manager definitions.

   Now alter the two queue-manager definitions at CHICAGO and CHICAGO2. Currently these definitions show that the queue managers hold full repositories for the cluster CHNSTORE. Change that definition to show that the queue managers hold full repositories for all clusters listed in the CHAINMAIL namelist. Alter the CHICAGO and CHICAGO2 queue manager definitions:

   ```
   ALTER QMGR REPOS(' ') REPOSNL(CHAINMAIL)
   ```

3. Alter the CLUSRCVR channels on CHICAGO and CHICAGO2.

   The CLUSRCVR channel definitions at CHICAGO and CHICAGO2 show that the channels are available in the cluster CHNSTORE. You need to change the cluster-receiver definition to show that the channels are available to all clusters listed in the CHAINMAIL namelist. Change the cluster-receiver definition at CHICAGO:

   ```
   ALTER CHANNEL(CHNSTORE.CHICAGO) CHLTYPE(CLUSRCVR)
         CLUSTER(' ') CLUSNL(CHAINMAIL)
   ```

At CHICAGO2, enter the command:

```
ALTER CHANNEL(CHNSTORE.CHICAGO2) CHLTYPE(CLUSRCVR)
      CLUSTER(' ') CLUSNL(CHAINMAIL)
```

4. Alter the CLUSSDR channels on CHICAGO and CHICAGO2.

   Change the two CLUSSDR channel definitions to add the namelist. At CHICAGO, enter the command:

```
ALTER CHANNEL(CHNSTORE.CHICAGO2) CHLTYPE(CLUSSDR)
      CLUSTER(' ') CLUSNL(CHAINMAIL)
```

   At CHICAGO2, enter the command:

```
ALTER CHANNEL(CHNSTORE.CHICAGO) CHLTYPE(CLUSSDR)
      CLUSTER(' ') CLUSNL(CHAINMAIL)
```

5. Create a namelist on SEATTLE and ATLANTA.

   Because SEATTLE and ATLANTA are going to be members of more than one cluster, you must create a namelist containing the names of the clusters. Define the namelist on SEATTLE and ATLANTA , as follows:

```
DEFINE NAMELIST(CHAINMAIL)
       DESCR('List of cluster names')
       NAMES(CHNSTORE, MAILORDER)
```

6. Alter the CLUSRCVR channels on SEATTLE and ATLANTA.

   The CLUSRCVR channel definitions at SEATTLE and ATLANTA show that the channels are available in the cluster CHNSTORE. Change the cluster-receive channel definitions to show that the channels are available to all clusters listed in the CHAINMAIL namelist. At SEATTLE, enter the command:

```
ALTER CHANNEL(CHNSTORE.SEATTLE) CHLTYPE(CLUSRCVR)
      CLUSTER(' ') CLUSNL(CHAINMAIL)
```

   At ATLANTA, enter the command:

```
ALTER CHANNEL(CHNSTORE.ATLANTA) CHLTYPE(CLUSRCVR)
      CLUSTER(' ') CLUSNL(CHAINMAIL)
```

7. Alter the CLUSSDR channels on SEATTLE and ATLANTA.

   Change the two CLUSSDR channel definitions to add the namelist. At SEATTLE, enter the command:

```
ALTER CHANNEL(CHNSTORE.CHICAGO) CHLTYPE(CLUSSDR)
      CLUSTER(' ') CLUSNL(CHAINMAIL)
```

   At ATLANTA, enter the command:

```
ALTER CHANNEL(CHNSTORE.CHICAGO2) CHLTYPE(CLUSSDR)
      CLUSTER(' ') CLUSNL(CHAINMAIL)
```

8. Define CLUSRCVR and CLUSSDR channels on HARTFORD and OMAHA .

   On the two new queue managers HARTFORD and OMAHA, define cluster-receiver and cluster-sender channels. It does not matter in which sequence you make the definitions. At HARTFORD, enter:

```
DEFINE CHANNEL(MAILORDER.HARTFORD) CHLTYPE(CLUSRCVR) TRPTYPE(TCP)
CONNAME(HARTFORD.CHSTORE.COM) CLUSTER(MAILORDER)
DESCR('Cluster-receiver channel for HARTFORD')

DEFINE CHANNEL(MAILORDER.CHICAGO) CHLTYPE(CLUSSDR) TRPTYPE(TCP)
CONNAME(CHICAGO.CHSTORE.COM) CLUSTER(MAILORDER)
DESCR('Cluster-sender channel from HARTFORD to repository at CHICAGO')
```

   At OMAHA, enter:

```
DEFINE CHANNEL(MAILORDER.OMAHA) CHLTYPE(CLUSRCVR) TRPTYPE(TCP)
CONNAME(OMAHA.CHSTORE.COM) CLUSTER(MAILORDER)
```

```
DESCR('Cluster-receiver channel for OMAHA')

DEFINE CHANNEL(MAILORDER.CHICAGO) CHLTYPE(CLUSSDR) TRPTYPE(TCP)
CONNAME(CHICAGO.CHSTORE.COM) CLUSTER(MAILORDER)
DESCR('Cluster-sender channel from OMAHA to repository at CHICAGO')
```

9. Define the MORDERQ queue on OMAHA .

   The final step to complete this task is to define the queue MORDERQ on the queue manager OMAHA . At OMAHA, enter:

   ```
   DEFINE QLOCAL(MORDERQ) CLUSTER(MAILORDER)
   ```

10. Check that the cluster changes have been propagated.

    Check that the definitions you created with the previous steps have been propagated though the cluster. Issue the following commands on a full repository queue manager:

    ```
    DIS QCLUSTER (MORDERQ)
    DIS CLUSQMGR
    ```

11.

Results

The cluster set up by this task is shown in Figure 42 on page 233.

Now we have two overlapping clusters. The full repositories for both clusters are held at CHICAGO and CHICAGO2. The mail order application that runs on OMAHA is independent of the inventory application that runs at CHICAGO. However, some of the queue managers that are in the CHNSTORE cluster are also in the MAILORDER cluster, and so they can send messages to either application. Before carrying out this task to overlap two clusters, be aware of the possibility of queue-name clashes.

Suppose that on NEWYORK in cluster CHNSTORE and on OMAHA in cluster MAILORDER, there is a queue called ACCOUNTQ . If you overlap the clusters and then an application on SEATTLE puts a message to the queue ACCOUNTQ , the message can go to either instance of the ACCOUNTQ .

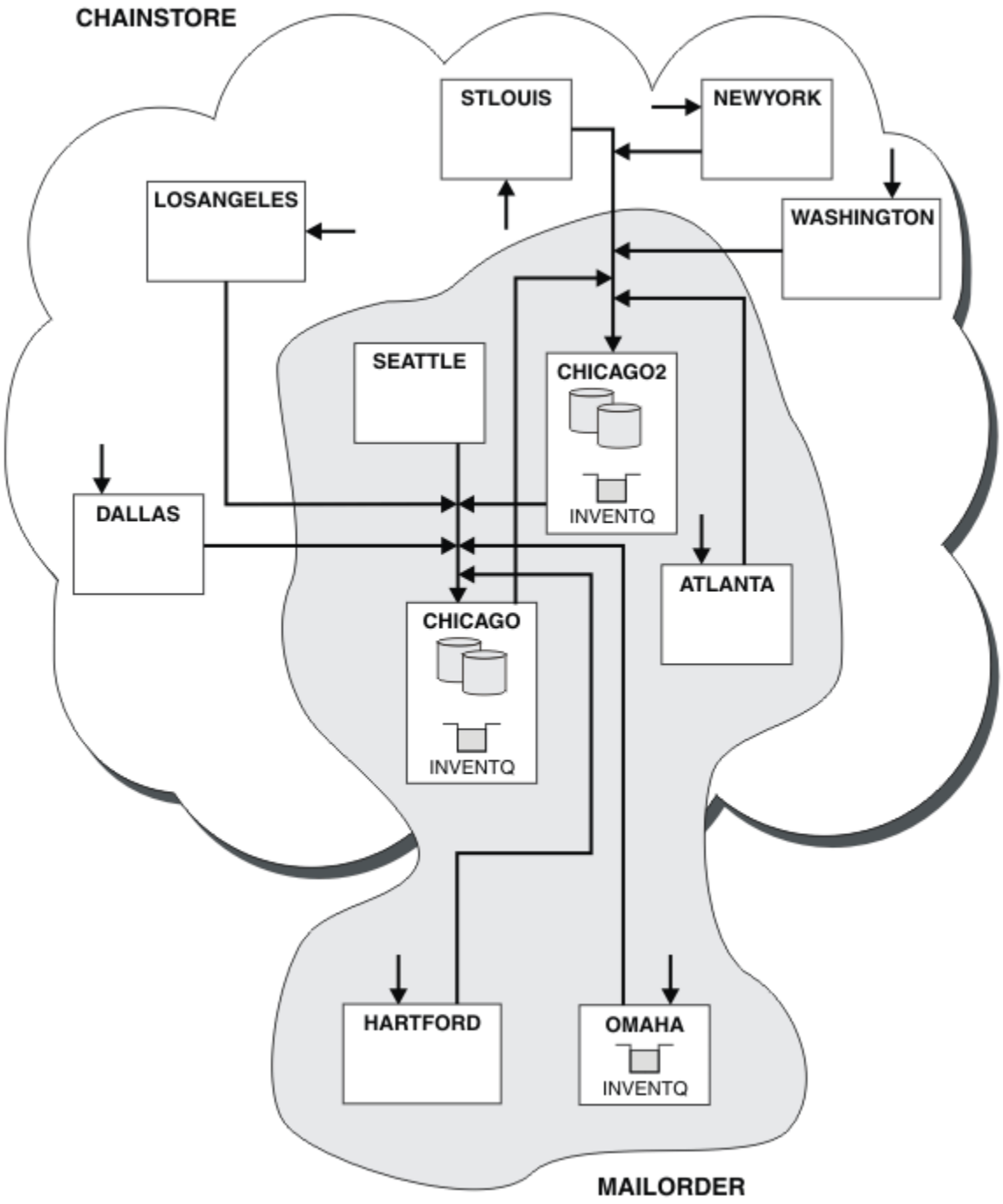


Figure 42. Interconnected clusters

What to do next

Suppose you decide to merge the MAILORDER cluster with the CHNSTORE cluster to form one large cluster called CHNSTORE.

To merge the MAILORDER cluster with the CHNSTORE cluster, such that CHICAGO and CHICAGO2 hold the full repositories:

- Alter the queue manager definitions for CHICAGO and CHICAGO2, removing the REPOSNL attribute, which specifies the namelist (CHAINMAIL), and replacing it with a REPOS attribute specifying the cluster name (CHNSTORE). For example:

```
ALTER QMGR(CHICAGO) REPOSNL(' ') REPOS(CHNSTORE)
```

- On each queue manager in the MAILORDER cluster, alter all the channel definitions and queue definitions to change the value of the CLUSTER attribute from MAILORDER to CHNSTORE. For example, at HARTFORD , enter:

```
ALTER CHANNEL(MAILORDER.HARTFORD) CLUSTER(CHNSTORE)
```

  At OMAHA enter:

```
ALTER QLOCAL(MORDERQ) CLUSTER(CHNSTORE)
```

- Alter all definitions that specify the cluster namelist CHAINMAIL, that is, the CLUSRCVR and CLUSSDR channel definitions at CHICAGO, CHICAGO2, SEATTLE, and ATLANTA , to specify instead the cluster CHNSTORE.

From this example, you can see the advantage of using namelists. Instead of altering the queue manager definitions for CHICAGO and CHICAGO2 you can alter the value of the namelist CHAINMAIL. Similarly, instead of altering the CLUSRCVR and CLUSSDR channel definitions at CHICAGO, CHICAGO2 , SEATTLE, and ATLANTA, you can achieve the required result by altering the namelist.

Removing a cluster network

Remove a cluster from a network and restore the distributed queuing configuration.

Before you begin

Note: For changes to a cluster to be propagated throughout the cluster, at least one full repository must always be available. Ensure that your repositories are available before starting this task.

Scenario:

- An IBM WebSphere MQ cluster has been set up as described in “Converting an existing network into a cluster” on page 227.
- This cluster is now to be removed from the system. The network of queue managers is to continue functioning as it did before the cluster was implemented.

About this task

Follow these steps to remove a cluster network.

Procedure

1. Remove cluster queues from the CHNSTORE cluster.

   On both CHICAGO and CHICAGO2, modify the local queue definition for the queue INVENTQ to remove the queue from the cluster. Issue the command:

   ```
   ALTER QLOCAL(INVENTQ) CLUSTER(' ')
   ```

   When you alter the queue, the information in the full repositories is updated and propagated throughout the cluster. Active applications using MQOO_BIND_NOT_FIXED, and applications using MQOO_BIND_AS_Q_DEF where the queue has been defined with DEFBIND(NOTFIXED), fail on the next attempted MQPUT or MQPUT1 call. The reason code MQRC_UNKNOWN_OBJECT_NAME is returned.

   You do not have to perform Step 1 first, but if you do not, perform it instead after Step 4.
2. Stop all applications that have access to cluster queue.

Stop all applications that have access to cluster queues. If you do not, some cluster information might remain on the local queue manager when you refresh the cluster in Step 5. This information is removed when all applications have stopped and the cluster channels have disconnected.

3. Remove the repository attribute from the full repository queue managers.

   On both CHICAGO and CHICAGO2, modify the queue manager definitions to remove the repository attribute. To do this issue the command:

   ```
   ALTER QMGR REPOS(' ')
   ```

   The queue managers inform the other queue managers in the cluster that they no longer hold the full repositories. When the other queue managers receive this information, you see a message indicating that the full repository has ended. You also see one or more messages indicating that there are no longer any repositories available for the cluster CHNSTORE .

4. Remove cluster channels.

   On CHICAGO remove the cluster channels:

   ```
   ALTER CHANNEL(CHNSTORE.CHICAGO2) CHLTYPE(CLUSSDR) CLUSTER(' ')
   ALTER CHANNEL(CHNSTORE.CHICAGO) CHLTYPE(CLUSRCVR) CLUSTER(' ')
   ```

   Note: It is important to issue the CLUSSDR command first, then CLUSRCVR command. Do not issue the CLUSRCVR command first, then the CLUSSDR command. Doing so, creates indoubt channels that have a STOPPED status. You then need to issue a START CHANNEL command to recover the stopped channels; for example, START CHANNEL(CHNSTORE.CHICAGO).

   You see messages indicating that there are no repositories for the cluster CHNSTORE.

   If you did not remove the cluster queues as described in Step 1, do so now.

5. Stop cluster channels.

   On CHICAGO stop the cluster channels with the following commands:

   ```
   STOP CHANNEL(CHNSTORE.CHICAGO2)
   STOP CHANNEL(CHNSTORE.CHICAGO)
   ```

6. Repeat steps 4 and 5 for each queue manager in the cluster.
7. Stop the cluster channels, then remove all definitions for the cluster channels and cluster queues from each queue manager.
8. Optional: Clear the cached cluster information held by the queue manager.

   Although the queue managers are no longer members of the cluster, they each retain a cached copy of information about the cluster. If you want to remove this data, see task “Restoring a queue manager to its pre-cluster state” on page 239.

9. Replace the remote-queue definitions for the INVENTQ

   So that the network can continue to function, replace the remote queue definition for the INVENTQ at every queue manager.

10. Tidy up the cluster.

    Delete any queue or channel definitions no longer required.

Removing a queue manager from a cluster

Remove a queue manager from a cluster, in scenarios where the queue manager can communicate normally with at least one full repository in the cluster.

Before you begin

This method is the best practice for scenarios in which at least one full repository is available, and can be contacted by the queue manager that is being removed. This method involves the least manual

intervention, and allows the queue manager to negotiate a controlled withdrawal from the cluster. If the queue manager that is being removed cannot contact a full repository, see “Removing a queue manager from a cluster: Alternative method” on page 237.

Before you remove the queue manager from the cluster, you must ensure that the queue manager is no longer hosting resources that are needed by the cluster:

- If the queue manager hosts a full repository, complete steps 1-4 from “Moving a full repository to another queue manager” on page 225.
- If the queue manager hosts cluster queues, complete steps 1-7 from “Removing a cluster queue from a queue manager” on page 223.
- If the queue manager hosts cluster topics, either delete the topics (for example by using the DELETE TOPIC command), or move them to other hosts.

  Note: If you remove a queue manager from a cluster, and the queue manager still hosts a cluster topic, then the queue manager might continue to attempt to deliver publications to the queue managers that are left in the cluster until the topic is deleted.

About this task

This example task removes the queue manager LONDON from the INVENTORY cluster. The INVENTORY cluster is set up as described in “Adding a queue manager to a cluster” on page 191, and modified as described in “Removing a cluster queue from a queue manager” on page 223.

The process for removing a queue manager from a cluster is more complicated than the process of adding a queue manager.

When a queue manager joins a cluster, the existing members of the cluster have no knowledge of the new queue manager and so have no interactions with it. New sender and receiver channels must be created on the joining queue manager so that it can connect to a full repository.

When a queue manager is removed from a cluster, it is likely that applications connected to the queue manager are using objects such as queues that are hosted elsewhere in the cluster. Also, applications that are connected to other queue managers in the cluster might be using objects hosted on the target queue manager. As a result of these applications, the current queue manager might create additional sender channels to establish communication with cluster members other than the full repository that it used to join the cluster. Every queue manager in the cluster has a cached copy of data that describes other cluster members. This might include the one that is being removed.

Procedure

1. Alter the manually defined cluster receiver channels to remove them from the cluster, on queue manager LONDON:

   ```
   ALTER CHANNEL(INVENTORY.LONDON) CHLTYPE(CLUSRCVR) CLUSTER(' ')
   ```

2. Alter the manually defined cluster sender channels to remove them from the cluster, on queue manager LONDON:

   ```
   ALTER CHANNEL(INVENTORY.PARIS) CHLTYPE(CLUSSDR) CLUSTER(' ')
   ```

   The other queue managers in the cluster learn that this queue manager and its cluster resources are no longer part of the cluster.

3. Monitor the cluster transmit queue, on queue manager LONDON, until there are no messages that are waiting to flow to any full repository in the cluster.

   ```
   DISPLAY CHSTATUS(INVENTORY.LONDON) XQMSGSA
   ```

   If messages remain on the transmit queue, determine why they are not being sent to the PARIS and NEWYORK full repositories before continuing.

Results

The queue manager LONDON is no longer part of the cluster. However, it can still function as an independent queue manager.

What to do next

The result of these changes can be confirmed by issuing the following command on the remaining members of the cluster:

```
DISPLAY CLUSQMGR(LONDON)
```

The queue manager continues to be displayed until the auto-defined cluster sender channels to it have stopped. You can wait for this to happen, or, continue to monitor for active instances by issuing the following command:

```
DISPLAY CHANNEL(INVENTORY.LONDON)
```

When you are confident that no more messages are being delivered to this queue manager, you can stop the cluster sender channels to LONDON by issuing the following command on the remaining members of the cluster:

```
STOP CHANNEL(INVENTORY.LONDON) STATUS(INACTIVE)
```

After the changes are propagated throughout the cluster, and no more messages are being delivered to this queue manager, stop and delete the CLUSRCVR channel on LONDON:

```
STOP CHANNEL(INVENTORY.LONDON)
DELETE CHANNEL(INVENTORY.LONDON)
```

The removed queue manager can be added back into the cluster at a later point as described in “Adding a queue manager to a cluster” on page 191. The removed queue manager continues to cache knowledge of the remaining members of the cluster for up to 90 days. If you prefer not to wait until this cache expires, it can be forcibly removed as described in “Restoring a queue manager to its pre-cluster state” on page 239.

Removing a queue manager from a cluster: Alternative method

Remove a queue manager from a cluster, in scenarios where, because of a significant system or configuration issue, the queue manager cannot communicate with any full repository in the cluster.

Before you begin

This alternative method of removing a queue manager from a cluster manually stops and deletes all cluster channels linking the removed queue manager to the cluster, and forcibly removes the queue manager from the cluster. This method is used in scenarios where the queue manager that is being removed cannot communicate with any of the full repositories. This might be (for example) because the queue manager has stopped working, or because there has been a prolonged communications failure between the queue manager and the cluster. Otherwise,use the most common method: “Removing a queue manager from a cluster” on page 235.

Before you remove the queue manager from the cluster, you must ensure that the queue manager is no longer hosting resources that are needed by the cluster:

- If the queue manager hosts a full repository, complete steps 1-4 from “Moving a full repository to another queue manager” on page 225.
- If the queue manager hosts cluster queues, complete steps 1-7 from “Removing a cluster queue from a queue manager” on page 223.

- If the queue manager hosts cluster topics, either delete the topics (for example by using the DELETE TOPIC command), or move them to other hosts.

  Note: If you remove a queue manager from a cluster, and the queue manager still hosts a cluster topic, then the queue manager might continue to attempt to deliver publications to the queue managers that are left in the cluster until the topic is deleted.

About this task

This example task removes the queue manager LONDON from the INVENTORY cluster. The INVENTORY cluster is set up as described in "Adding a queue manager to a cluster" on page 191, and modified as described in "Removing a cluster queue from a queue manager" on page 223.

The process for removing a queue manager from a cluster is more complicated than the process of adding a queue manager.

When a queue manager joins a cluster, the existing members of the cluster have no knowledge of the new queue manager and so have no interactions with it. New sender and receiver channels must be created on the joining queue manager so that it can connect to a full repository.

When a queue manager is removed from a cluster, it is likely that applications connected to the queue manager are using objects such as queues that are hosted elsewhere in the cluster. Also, applications that are connected to other queue managers in the cluster might be using objects hosted on the target queue manager. As a result of these applications, the current queue manager might create additional sender channels to establish communication with cluster members other than the full repository that it used to join the cluster. Every queue manager in the cluster has a cached copy of data that describes other cluster members. This might include the one that is being removed.

This procedure might be appropriate in an emergency, when it is not possible to wait for the queue manager to leave the cluster gracefully.

Procedure

1. Stop all channels used to communicate with other queue managers in the cluster. Use MODE(FORCE) to stop the CLUSRCVR channel, on queue manager LONDON. Otherwise you might need to wait for the sender queue manager to stop the channel:

   ```
   STOP CHANNEL(INVENTORY.LONDON) MODE(FORCE)
   STOP CHANNEL(INVENTORY.TORONTO)
   STOP CHANNEL(INVENTORY.PARIS)
   STOP CHANNEL(INVENTORY.NEWYORK)
   ```

2. Monitor the channel states, on queue manager LONDON, until the channels stop:

   ```
   DISPLAY CHSTATUS(INVENTORY.LONDON)
   DISPLAY CHSTATUS(INVENTORY.TORONTO)
   DISPLAY CHSTATUS(INVENTORY.PARIS)
   DISPLAY CHSTATUS(INVENTORY.NEWYORK)
   ```

   No more application messages are sent to or from the other queue managers in the cluster after the channels stop.

3. Delete the manually defined cluster channels, on queue manager LONDON:

   ```
   DELETE CHANNEL(INVENTORY.NEWYORK)
   DELETE CHANNEL(INVENTORY.TORONTO)
   ```

4. The remaining queue managers in the cluster still retain knowledge of the removed queue manager, and might continue to send messages to it. To purge the knowledge from the remaining queue managers, reset the removed queue manager from the cluster on one of the full repositories:

   ```
   RESET CLUSTER(INVENTORY) ACTION(FORCEREMOVE) QMNAME(LONDON) QUEUES(YES)
   ```

   If there might be another queue manager in the cluster that has the same name as the removed queue manager, specify the **QMID** of the removed queue manager.

Results

The queue manager LONDON is no longer part of the cluster. However, it can still function as an independent queue manager.

What to do next

The result of these changes can be confirmed by issuing the following command on the remaining members of the cluster:

```
DISPLAY CLUSQMGR(LONDON)
```

The queue manager continues to be displayed until the auto-defined cluster sender channels to it have stopped. You can wait for this to happen, or, continue to monitor for active instances by issuing the following command:

```
DISPLAY CHANNEL(INVENTORY.LONDON)
```

After the changes are propagated throughout the cluster, and no more messages are being delivered to this queue manager, delete the CLUSRCVR channel on LONDON:

```
DELETE CHANNEL(INVENTORY.LONDON)
```

The removed queue manager can be added back into the cluster at a later point as described in “Adding a queue manager to a cluster” on page 191. The removed queue manager continues to cache knowledge of the remaining members of the cluster for up to 90 days. If you prefer not to wait until this cache expires, it can be forcibly removed as described in “Restoring a queue manager to its pre-cluster state” on page 239.

Restoring a queue manager to its pre-cluster state

When a queue manager is removed from a cluster, it retains knowledge of the remaining cluster members. This knowledge eventually expires and is deleted automatically. However, if you prefer to delete it immediately, you can use the steps in this topic.

Before you begin

It is assumed that the queue manager has been removed from the cluster, and is no longer performing any work in the cluster. For example, its queues are no longer receiving messages from the cluster, and no applications are waiting for messages to arrive in these queues.

Important: If you remove a queue manager from a cluster and refresh it using REPOS(YES), you will not be able to add it back in again by simply altering its CLUSRCVR's CLUSTER attribute. After altering its CLUSRCVR's CLUSTER attribute to be nonblank (that is, the clustername), you will additionally need to issue refresh cluster with REPOS(NO), at which point the internal sequence numbers on the CLUSRCVR will be brought up to date. Then the queue manager will be successful in re-introducing itself to the Full Repositories and the rest of the cluster members. (Note the REPOS(NO) version of the command must be run after the CLUSRCVR channel has been given the correct cluster name.)

This restriction applies to IBM WebSphere MQ Version 7.5 only.

About this task

When a queue manager is removed from a cluster, it retains knowledge of the remaining cluster members for up to 90 days. This can have system benefits, particularly if the queue manager quickly rejoins the cluster. When this knowledge eventually expires, it is deleted automatically. However, there are reasons why you might prefer to delete this information manually. For example:

- You might want to confirm that you have stopped every application on this queue manager that previously used cluster resources. Until the knowledge of the remaining cluster members expires, any such application continues to write to a transmit queue. After the cluster knowledge is deleted, the system generates an error message when such an application tries to use cluster resources.
- When you display status information for the queue manager, you might prefer not to see expiring information about remaining cluster members.

This task uses the INVENTORY cluster as an example. The LONDON queue manager has been removed from the INVENTORY cluster as described in "Removing a queue manager from a cluster" on page 235. To delete knowledge of the remaining members of the cluster, issue the following commands on the LONDON queue manager.

Procedure

1. Remove all memory of the other queue managers in the cluster from this queue manager:

```
REFRESH CLUSTER(INVENTORY) REPOS(YES)
```

2. Monitor the queue manager until all the cluster resources are gone:

```
DISPLAY CLUSQMGR(*) CLUSTER(INVENTORY)
DISPLAY QCLUSTER(*) CLUSTER(INVENTORY)
DISPLAY TOPIC(*) CLUSTER(INVENTORY)
```

Related concepts

Clusters

"Comparison of clustering and distributed queuing" on page 157
Compare the components that need to be defined to connect queue managers using distributed queuing and clustering.

"Components of a cluster" on page 159
Clusters are composed of queue managers, cluster repositories, cluster channels, and cluster queues.

"Managing IBM WebSphere MQ clusters" on page 180
You can create, extend, and maintain IBM WebSphere MQ clusters.

Maintaining a queue manager

Suspend and resume a queue manager from a cluster to perform maintenance.

About this task

From time to time, you might need to perform maintenance on a queue manager that is part of a cluster. For example, you might need to take backups of the data in its queues, or apply fixes to the software. If the queue manager hosts any queues, its activities must be suspended. When the maintenance is complete, its activities can be resumed.

Procedure

1. Suspend a queue manager, by issuing the SUSPEND QMGR **runmqsc** command:

```
SUSPEND QMGR CLUSTER(SALES)
```

   The SUSPEND **runmqsc** command notifies the queue managers in the SALES cluster that this queue manager has been suspended.

   The purpose of the SUSPEND QMGR command is only to advise other queue managers to avoid sending messages to this queue manager if possible. It does not mean that the queue manager is disabled. Some messages that have to be handled by this queue manager are still sent to it, for example when this queue manager is the only host of a clustered queue.

While the queue manager is suspended the workload management routines avoid sending messages to it. Messages that have to be handled by that queue manager include messages sent by the local queue manager.

WebSphere MQ uses a workload balancing algorithm to determine which destinations are suitable, rather than selecting the local queue manager whenever possible.

a) Enforce the suspension of a queue manager by using the FORCE option on the SUSPEND QMGR command:

```
SUSPEND QMGR CLUSTER(SALES) MODE(FORCE)
```

MODE(FORCE) forcibly stops all inbound channels from other queue managers in the cluster. If you do not specify MODE(FORCE), the default MODE(QUIESCE) applies.

2. Do whatever maintenance tasks are necessary.
3. Resume the queue manager by issuing the RESUME QMGR **runmqsc** command:

```
RESUME QMGR CLUSTER(SALES)
```

Results

The RESUME **runmqsc** command notifies the full repositories that the queue manager is available again. The full repository queue managers disseminate this information to other queue managers that have requested updates to information concerning this queue manager.

Maintaining the cluster transmission queue

Make every effort to keep cluster transmission queues available. They are essential to the performance of clusters.

Before you begin

- Make sure that the cluster transmission queue does not become full.
- Take care not to issue an ALTER **runmqsc** command to set it either get disabled or put disabled accidentally.
- Make sure that the medium the cluster transmission queue is stored on does not become full.

Refreshing a cluster queue manager

You can remove auto-defined channels and auto-defined cluster objects from the local repository using the REFRESH CLUSTER command. No messages are lost.

Before you begin

You might be asked to use the command by your IBM Support Center. Do not use the command without careful consideration. For example, for large clusters use of the **REFRESH CLUSTER** command can be disruptive to the cluster while it is in progress, and again at 27 day intervals thereafter when the cluster objects automatically send status updates to all interested queue managers. See “Clustering: Using REFRESH CLUSTER best practices” on page 300.

About this task

A queue manager can make a fresh start in a cluster. In normal circumstances, you do not need to use the REFRESH CLUSTER command.

Procedure

Issue the REFRESH CLUSTER **MQSC** command from a queue manager to remove auto-defined cluster queue-manager and queue objects from the local repository.

The command only removes objects that refer to other queue managers, it does not remove objects relating to the local queue manager. The command also removes auto-defined channels. It removes channels that do not have messages on the cluster transmission queue and are not attached to a full repository queue manager.

Results

Effectively, the REFRESH CLUSTER command allows a queue manager to be cold-started with respect to its full repository content. IBM WebSphere MQ ensures that no data is lost from your queues.

Related concepts

"Clustering: Using REFRESH CLUSTER best practices" on page 300
You use the **REFRESH CLUSTER** command to discard all locally held information about a cluster and rebuild that information from the full repositories in the cluster. You should not need to use this command, except in exceptional circumstances. If you do need to use it, there are special considerations about how you use it. This information is a guide based on testing and feedback from customers.

Recovering a queue manager

Bring the cluster information about a queue manager up to date using the REFRESH CLUSTER **runmqsc** command. Follow this procedure after recovering a queue manager from a point-in-time backup.

Before you begin

You have restored a cluster queue manager from a point-in-time backup.

About this task

To recover a queue manager in a cluster, restore the queue manager, and then bring the cluster information up to date using the REFRESH CLUSTER **runmqsc** command.

Note: For large clusters, use of the **REFRESH CLUSTER** command can be disruptive to the cluster while it is in progress, and again at 27 day intervals thereafter when the cluster objects automatically send status updates to all interested queue managers. See Refreshing in a large cluster can affect performance and availability of the cluster.

Procedure

Issue the REFRESH CLUSTER command on the restored queue manager for all clusters in which the queue manager participates.

What to do next

There is no need to issue the REFRESH CLUSTER command on any other queue manager.

Related concepts

"Clustering: Using REFRESH CLUSTER best practices" on page 300
You use the **REFRESH CLUSTER** command to discard all locally held information about a cluster and rebuild that information from the full repositories in the cluster. You should not need to use this command, except in exceptional circumstances. If you do need to use it, there are special considerations about how you use it. This information is a guide based on testing and feedback from customers.

Configuring cluster channels for availability

Follow good configuration practices to keep cluster channels running smoothly if there are intermittent network stoppages.

Before you begin

Clusters relieve you of the need to define channels, but you still need to maintain them. The same channel technology is used for communication between queue managers in a cluster as is used in distributed queuing. To understand about cluster channels, you need to be familiar with matters such as:

- How channels operate
- How to find their status
- How to use channel exits

About this task

You might want to give some special consideration to the following points:

Procedure

Consider the following points when configuring cluster channels

- Choose values for HBINT or KAINT on cluster-sender channels and cluster-receiver channels that do not burden the network with lots of heartbeat or keep alive flows. An interval less than about 10 seconds gives false failures, if your network sometimes slows down and introduces delays of this length.
- Set the BATCHHB value to reduce the window for causing a marooned message because it is indoubt on a failed channel. An indoubt batch on a failed channel is more likely to occur if the batch is given longer to fill. If the message traffic along the channel is sporadic with long periods of time between bursts of messages a failed batch is more likely.
- A problem arises if the cluster-sender end of a channel fails and then tries to restart before the heartbeat or keep alive has detected the failure. The channel-sender restart is rejected if the cluster-receiver end of the channel has remained active. To avoid the failure, arrange for the cluster-receiver channel to be terminated and restarted when a cluster-sender channel attempts to restart.

  On platforms other than z/OS

  Control the problem of the cluster-receiver end of the channel remaining active using the AdoptNewMCA, AdoptNewMCATimeout, and AdoptNewMCACheck attributes in the qm.ini file or the Windows NT Registry.

Routing messages to and from clusters

Use queue aliases, queue manager aliases, and remote queue definitions to connect clusters to external queue managers and other clusters.

For details on routing messages to and from clusters, see the following subtopics:

Related concepts

Clusters

How clusters work

“Comparison of clustering and distributed queuing” on page 157

Compare the components that need to be defined to connect queue managers using distributed queuing and clustering.

“Components of a cluster” on page 159

Clusters are composed of queue managers, cluster repositories, cluster channels, and cluster queues.

“Managing IBM WebSphere MQ clusters” on page 180

You can create, extend, and maintain IBM WebSphere MQ clusters.

"Queue-manager aliases and clusters" on page 254
Use queue-manager aliases to hide the name of queue managers when sending messages into or out of a cluster, and to workload balance messages sent to a cluster.

"Queue aliases and clusters" on page 256
Use queue aliases to hide the name of a cluster queue, to cluster a queue, adopt different attributes, or adopt different access controls.

"Reply-to queue aliases and clusters" on page 256
A reply-to queue alias definition is used to specify alternative names for reply information. Reply-to queue alias definitions can be used with clusters just the same as in a distributed queuing environment.

Related tasks

"Configuring a queue manager cluster" on page 154
Use the links in this topic to find out how clusters work, how to design a cluster configuration, and to get an example of how to set up a simple cluster.

"Setting up a new cluster" on page 181
Follow these instructions to set up the example cluster. Separate instructions describe setting up the cluster on TCP/IP, LU 6.2, and with a single transmission queue or multiple transmission queues. Test the cluster works by sending a message from one queue manager to the other.

Configuring request/reply to a cluster

Configure a request/reply message path from a queue manager outside a cluster. Hide the inner details of the cluster by using a gateway queue manager as the communication path to and from the cluster.

Before you begin

Figure 43 on page 245 shows a queue manager called QM3 that is outside the cluster called DEMO. QM3 could be a queue manager on a WebSphere MQ product that does not support clusters. QM3 hosts a queue called Q3, which is defined as follows:

```
DEFINE QLOCAL(Q3)
```

Inside the cluster are two queue managers called QM1 and QM2. QM2 hosts a cluster queue called Q2, which is defined as follows:

```
DEFINE QLOCAL(Q2) CLUSTER(DEMO)
```

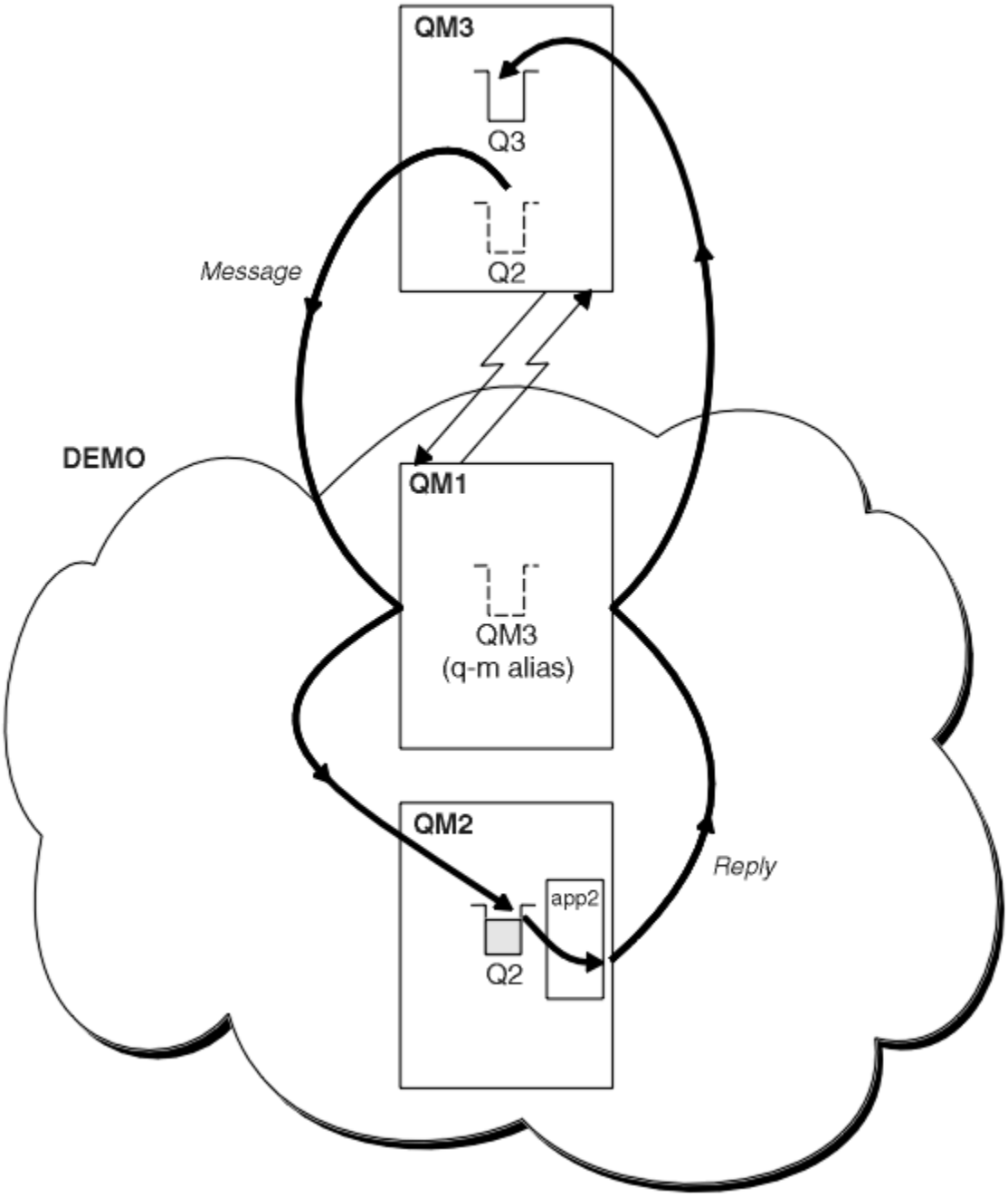


Figure 43. Putting from a queue manager outside the cluster

About this task

Follow the advice in the procedure to set up the path for the request and reply messages.

Procedure

1. Send the request message to the cluster.

   Consider how the queue manager that is outside the cluster puts a message to the queue Q2 at QM2, that is inside the cluster. A queue manager outside the cluster must have a QREMOTE definition for each queue in the cluster that it puts messages to.

   a) Define a remote queue for Q2 on QM3.

   ```
   DEFINE QREMOTE(Q2) RNAME(Q2) RQMNAME(QM2) XMITQ(QM1)
   ```

Because QM3 is not part of a cluster, it must communicate using distributed queuing techniques. Therefore, it must also have a sender-channel and a transmission queue to QM1. QM1 needs a corresponding receiver channel. The channels and transmission queues are not shown explicitly in Figure 43 on page 245.

In the example, an application at QM3 issues an MQPUT call to put a message to Q2. The QREMOTE definition causes the message to be routed to Q2 at QM2 using the sender-channel that is getting messages from the QM1 transmission queue.

2. Receive the reply message from the cluster.

   Use a queue manager alias to create a return path for replies to a queue manager outside the cluster. The gateway, QM1, advertises a queue-manager alias for the queue manager that is outside the cluster, QM3. It advertises QM3 to the queue managers inside the cluster by adding the cluster attribute to a queue manager alias definition for QM3. A queue manager alias definition is like a remote queue definition, but with a blank RNAME.

   a) Define a queue manager alias for QM3 on QM1.

   ```
   DEFINE QREMOTE(QM3) RNAME(' ') RQMNAME(QM3) CLUSTER(DEMO)
   ```

   We must consider the choice of name for the transmission queue used to forward replies back from QM1 to QM3. Implicit in the QREMOTE definition, by the omission of the XMITQ attribute, is the name of the transmission queue is QM3. But QM3 is the same name as we expect to advertise to the rest of the cluster using the queue manager alias. WebSphere MQ does not allow you to give both the transmission queue and the queue manager alias the same name. One solution is to create a transmission queue to forward messages to QM3 with a different name to the queue manager alias.

   b) Provide the transmission queue name in the QREMOTE definition.

   ```
   DEFINE QREMOTE(QM3) RNAME(' ') RQMNAME(QM3) CLUSTER(DEMO) XMITQ(QM3.XMIT)
   ```

   The new queue manager alias couples the new transmission queue called QM3.XMIT with the QM3 queue manager alias. It is a simple and correct solution, but not wholly satisfactory. It has broken the naming convention for transmission queues that they are given the same name as the target queue manager. Are there any alternative solutions that preserve the transmission queue naming convention?

   The problem arises because the requester defaulted to passing QM3 as the reply-to queue manager name in the request message that is sent from QM3. The server on QM2 uses the QM3 reply-to queue manager name to address QM3 in its replies. The solution required QM1 to advertise QM3 as the queue manager alias to return reply messages to and prevented QM1 from using QM3 as the name of the transmission queue.

   Instead of defaulting to providing QM3 as the reply-to queue manager name, applications on QM3 need to pass a reply-to queue manager alias to QM1 for reply messages. The gateway queue manager QM1 advertises the queue manager alias for replies to QM3 rather than QM3 itself, avoiding the conflict with the name of the transmission queue.

   c) Define a queue manager alias for QM3 on QM1.

   ```
   DEFINE QREMOTE(QM3.ALIAS) RNAME(' ') RQMNAME(QM3) CLUSTER(DEMO)
   ```

   Two changes to the configuration commands are required.

   i) The QREMOTE at QM1 now advertises our queue manager alias QM3.ALIAS to the rest of the cluster, coupling it to the name of the real queue manager QM3. QM3 is again the name of the transmission queue to send reply queues back to QM3

   ii) The client application must provide QM3.ALIAS as the name of the reply-to queue manager when it constructs the request message. You can provide QM3.ALIAS to the client application in one of two ways.

   - Code QM3.ALIAS in the reply-to queue manager name field constructed by MQPUT in the MQMD. You must do it this way if you are using a dynamic queue for replies.

- Use a reply-to queue alias, `Q3.ALIAS`, rather than a reply-to queue when providing the reply-to queue name.

```
DEFINE QREMOTE(Q3.ALIAS) RNAME(Q3) RQMNAME(QM3.ALIAS)
```

What to do next

Note: You cannot demonstrate the use of reply-to queue aliases with **AMQSREQ0**. It opens the reply-to queue using the queue name provided in parameter 3, or the default `SYSTEM.SAMPLE.REPLY` model queue. You need to modify the sample providing another parameter containing the reply-to queue alias to name the reply-to queue manager alias for `MQPUT`.

Related tasks

"Hiding the name of a cluster target queue manager" on page 247
Route a message to a cluster queue that is defined on any queue manager in a cluster without naming the queue manager.

Hiding the name of a cluster target queue manager

Route a message to a cluster queue that is defined on any queue manager in a cluster without naming the queue manager.

Before you begin

- Avoid revealing the names of queue managers that are inside the cluster to queue managers that are outside the cluster.
  - Resolving references to the queue manager hosting a queue inside the cluster removes the flexibility to do workload balancing.
  - It also makes it difficult for you to change a queue manager hosting a queue in the cluster.
  - The alternative is to replace `RQMNAME` with a queue manager alias provided by the cluster administrator.
  - "Hiding the name of a cluster target queue manager" on page 247 describes using a queue manager alias to decouple a queue manager outside a cluster from the management of queue managers inside a cluster.
- However, the suggested way to name transmission queues is to give them the name of the target queue manager. The name of the transmission queue reveals the name of a queue manager in the cluster. You have to choose which rule to follow. You might choose to name the transmission queue using either the queue manager name or the cluster name:

  Name the transmission queue using the gateway queue manager name
  Disclosure of the gateway queue manager name to queue managers outside a cluster is a reasonable exception to the rule of hiding cluster queue manager names.

  Name the transmission queue using the name of the cluster
  If you are not following the convention of naming transmission queues with the name of the target queue manager, use the cluster name.

About this task

Modify the task "Configuring request/reply to a cluster" on page 244, to hide the name of the target queue manager inside the cluster.

Procedure

In the example, see Figure 44 on page 248 , define a queue manager alias on the gateway queue manager `QM1` called `DEMO`:

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

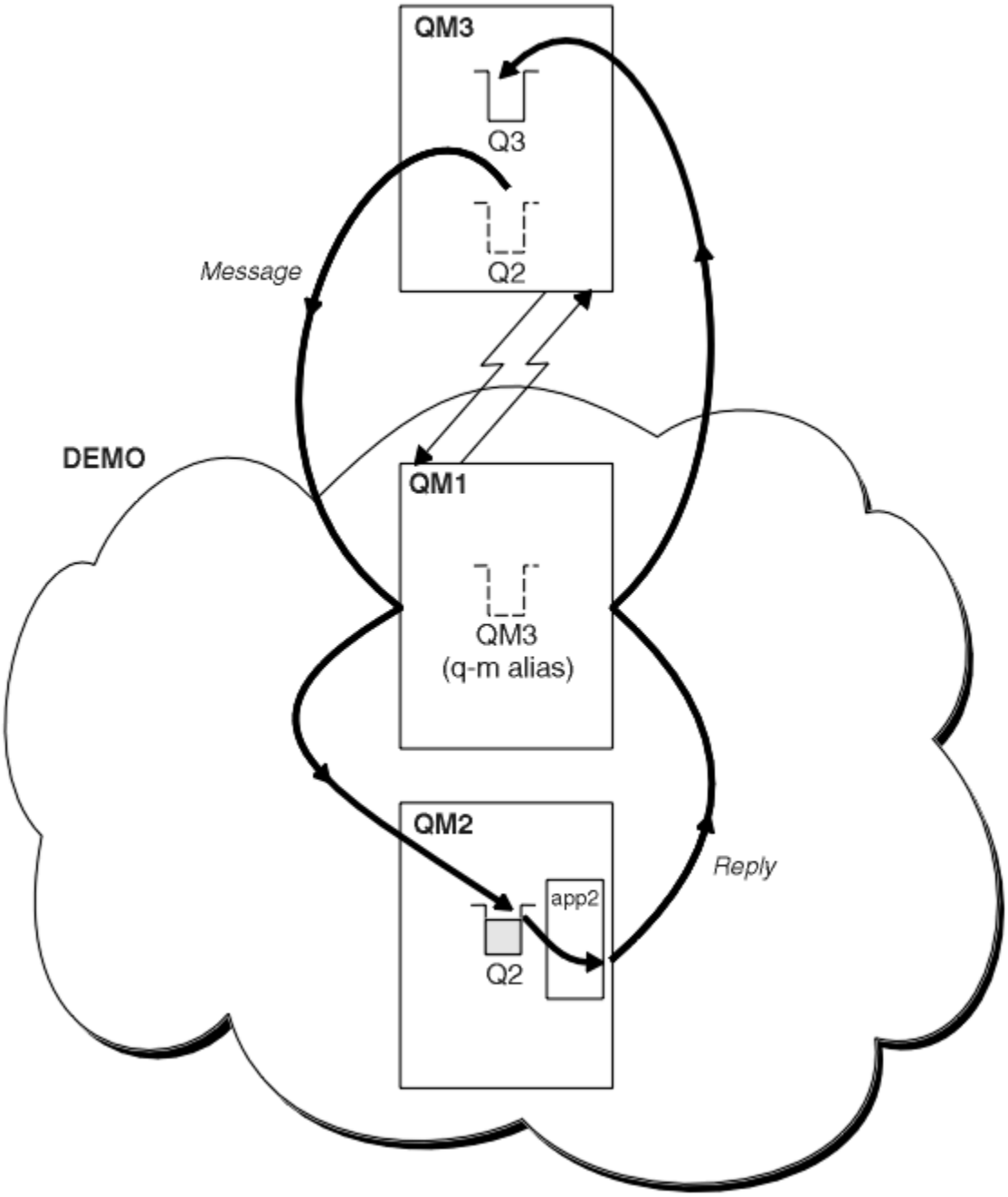


Figure 44. Putting from a queue manager outside the cluster

The QREMOTE definition on QM1 makes the queue manager alias DEMO known to the gateway queue manager. QM3, the queue manager outside the cluster, can use the queue manager alias DEMO to send messages to cluster queues on DEMO, rather than having to use an actual queue manager name.

If you adopt the convention of using the cluster name to name the transmission queue connecting to a cluster, then the remote queue definition for Q2 becomes:

```
DEFINE QREMOTE(Q2) RNAME(Q2) RQMNAME(DEMO)
```

Results

Messages destined for Q2 on DEMO are placed on the DEMO transmission queue. From the transmission queue they are transferred by the sender-channel to the gateway queue manager, QM1. The gateway queue manager routes the messages to any queue manager in the cluster that hosts the cluster queue Q2.

Configuring request/reply from a cluster

Configure a request/reply message path from a cluster to a queue manager outside the cluster. Hide the details of how a queue manager inside the cluster communicates outside the cluster by using a gateway queue manager.

Before you begin

Figure 45 on page 250 shows a queue manager, QM2, inside the cluster DEMO. It sends a request to a queue, Q3, hosted on queue manager outside the cluster. The replies are returned to Q2 at QM2 inside the cluster.

To communicate with the queue manager outside the cluster, one or more queue managers inside the cluster act as a gateway. A gateway queue manager has a communication path to the queue managers outside the cluster. In the example, QM1 is the gateway.

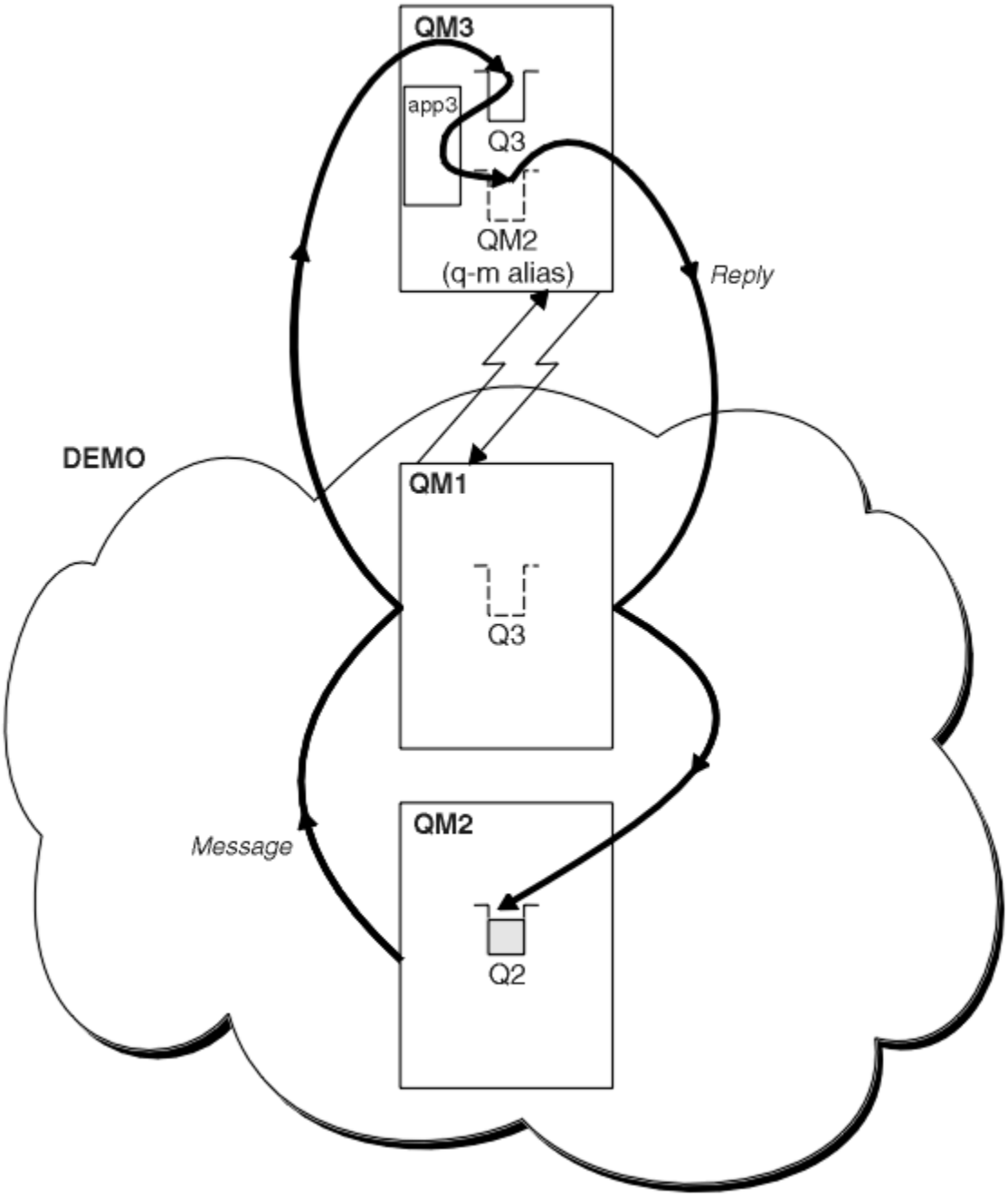


Figure 45. Putting to a queue manager outside the cluster

About this task

Follow the instructions to set up the path for the request and reply messages

Procedure

1. Send the request message from the cluster.

   Consider how the queue manager, QM2, which is inside the cluster puts a message to the queue Q3 at QM3, which is outside the cluster.

   a) Create a QREMOTE definition on QM1 that advertises the remote queue Q3 to the cluster

   ```
   DEFINE QREMOTE(Q3) RNAME(Q3) RQMNAME(QM3) CLUSTER(DEMO)
   ```

It also has a sender-channel and a transmission queue to the queue manager that is outside the cluster. QM3 has a corresponding receiver-channel. The channels are not shown in Figure 45 on page 250.

An application on QM2 issues an MQPUT call specifying the target queue and the queue to which replies are to be sent. The target queue is Q3 and the reply-to queue is Q2.

The message is sent to QM1, which uses its remote-queue definition to resolve the queue name to Q3 at QM3.

2. Receive the reply message from the queue manager outside the cluster.

   A queue manager outside the cluster must have a queue manager alias for each queue manager in the cluster to which it send a message. The queue-manager alias must also specify the name of the transmission queue to the gateway queue manager. In this example, QM3 needs a queue manager alias definition for QM2:

   a) Create a queue manager alias QM2 on QM3

   ```
   DEFINE QREMOTE(QM2) RNAME(' ') RQMNAME(QM2) XMITQ(QM1)
   ```

   QM3 also needs a sender-channel and transmission queue to QM1 and QM1 needs a corresponding receiver-channel.

   The application, **app3**, on QM3 can then send replies to QM2, by issuing an MQPUT call and specifying the queue name, Q2 and the queue manager name, QM2.

What to do next

You can define more than one route out of a cluster.

Configuring workload balancing from outside a cluster

Configure a message path from a queue manager outside a cluster to any copy of a cluster queue. The result is to workload balance requests from outside the cluster to each instance of a cluster queue.

Before you begin

Configure the example, as shown in Figure 43 on page 245 in "Configuring request/reply to a cluster" on page 244.

About this task

In this scenario, the queue manager outside the cluster, QM3 in Figure 46 on page 252, sends requests to the queue Q2. Q2 is hosted on two queue managers within cluster DEMO to use workload balancing. A queue named Q2 is defined on the queue managers QM2 and QM4 but not on the gateway queue manager QM1. The requests from QM3, the queue manager outside the cluster, are sent to either instance of Q2.

QM3 is not part of a cluster and communicates using distributed queuing techniques. It must have a sender-channel and a transmission queue to QM1. QM1 needs a corresponding receiver-channel. The channels and transmission queues are not shown explicitly in Figure 46 on page 252.

The procedure extends the example in Figure 43 on page 245 in "Configuring request/reply to a cluster" on page 244.

Procedure

1. Define a local queue called Q2 on each of QM2 and QM4.

   ```
   DEFINE QLOCAL(Q2) CLUSTER(DEMO) DEFBIND(NOTFIXED)
   ```

2. Create a QREMOTE definition for Q2 on QM3.

   ```
   DEFINE QREMOTE(Q2) RNAME(Q2) RQMNAME(Q3) XMITQ(QM1)
   ```

Create a QREMOTE definition for each queue in the cluster that QM3 puts messages to.

3. Create a queue-manager alias Q3 on QM3.

```
DEFINE QREMOTE(Q3) RNAME(' ') RQMNAME(' ') CLUSTER(DEMO) DEFBIND(NOTFIXED)
```

   Q3 is not a real queue manager name. It is the name of a queue manager alias definition in the cluster that equates the queue manager alias name Q3 with blank, ' '

4. QM1, the gateway queue manager, has no special definitions.

Results

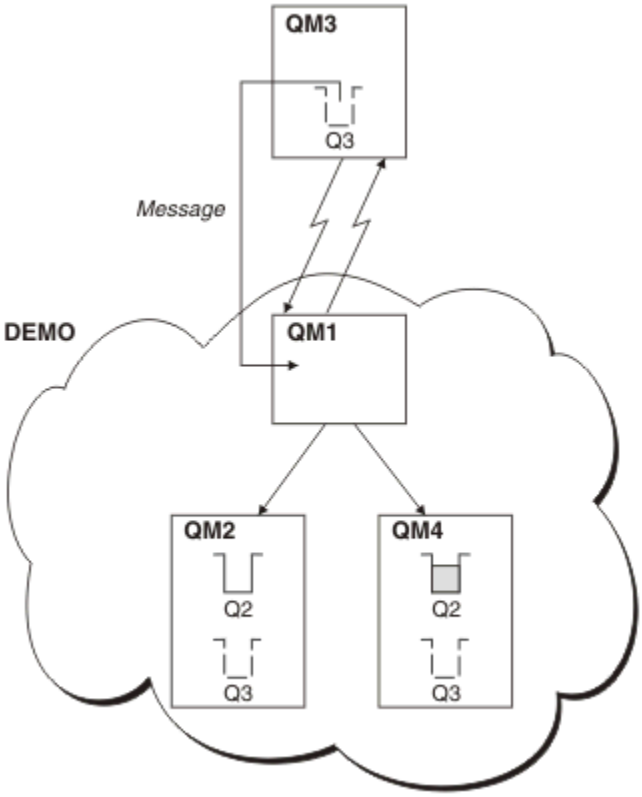


Figure 46. Putting from a queue manager outside the cluster

When an application at QM3 issues an MQPUT call to put a message to Q2, the QREMOTE definition causes the message to be routed through the gateway queue manager QM1. QM1 uses workload balancing to distribute messages targeted to Q2 between the queues called Q2 on the two queue managers, QM2 and QM4, which have cluster queue manager aliases for Q3.

Configuring message paths between clusters

Connect clusters together using a gateway queue manager. Make queues or queue managers visible to all the clusters by defining cluster queue or cluster queue manager aliases on the gateway queue manager.

About this task

Instead of grouping all your queue managers together in one large cluster, you can have many smaller clusters. Each cluster has one or more queue managers in acting as a bridge. The advantage of this is that you can restrict the visibility of queue and queue-manager names across the clusters. See “Overlapping clusters” on page 175. Use aliases to change the names of queues and queue managers to avoid name conflicts or to comply with local naming conventions.

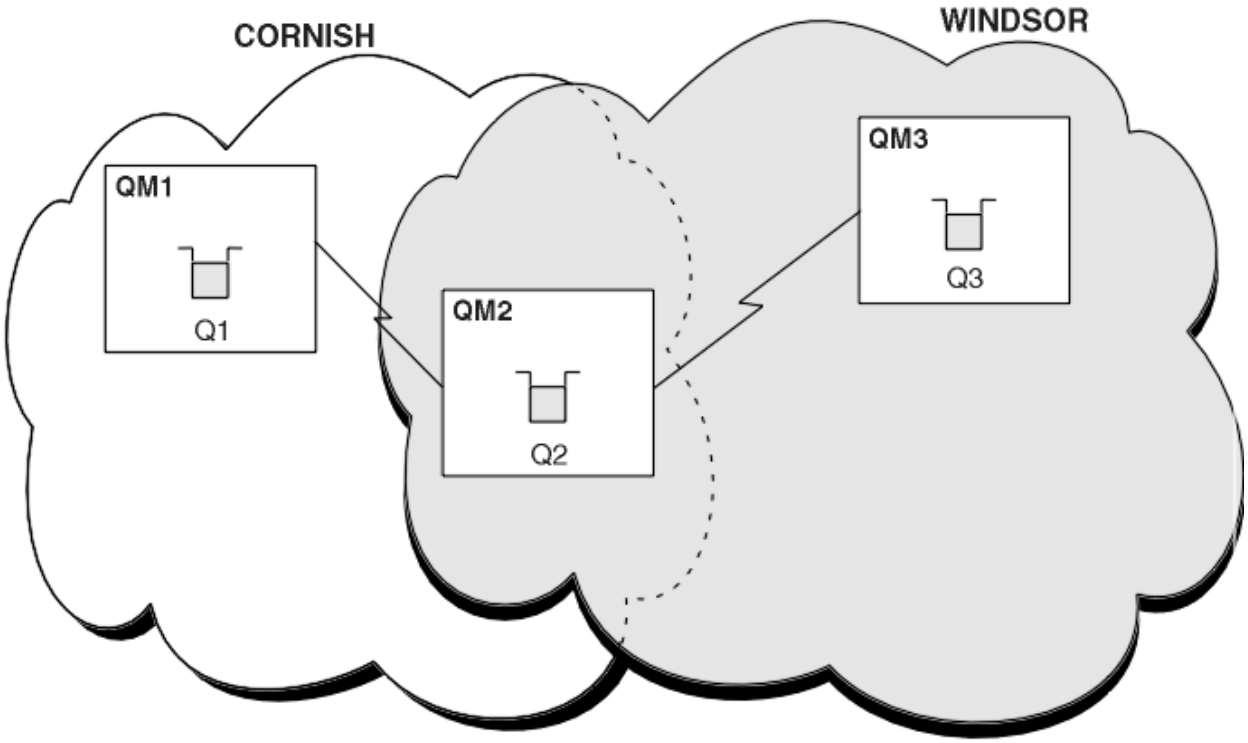


Figure 47. Bridging across clusters

Figure 47 on page 253 shows two clusters with a bridge between them. There could be more than one bridge.

Configure the clusters using the following procedure:

Procedure

1. Define a cluster queue, Q1 on QM1.

```
DEFINE QLOCAL(Q1) CLUSTER(CORNISH)
```

2. Define a cluster queue, Q3 on QM3.

```
DEFINE QLOCAL(Q3) CLUSTER(WINDSOR)
```

3. Create a namelist called CORNISHWINDSOR on QM2, containing the names of both clusters.

```
DEFINE NAMELIST(CORNISHWINDSOR) DESCR('CornishWindsor namelist')
       NAMES(CORNISH, WINDSOR)
```

4. Define a cluster queue, Q2 on QM2

```
DEFINE QLOCAL(Q2) CLUSNL(CORNISHWINDSOR)
```

What to do next

QM2 is a member of both clusters and is the bridge between them. For each queue that you want to make visible across the bridge, you need a QALIAS definition on the bridge. For example in Figure 47 on page 253, on QM2, you need:

```
DEFINE QALIAS(MYQ3) TARGET(Q3) CLUSTER(CORNISH) DEFBIND(NOTFIXED)
```

Using the queue alias, an application connected to a queue manager in CORNISH, for example QM1, can put a message to Q3. It refers to Q3 as MYQ3. The message is routed to Q3 at QM3.

When you open a queue, you need to set DEFBIND to either NOTFIXED or QDEF. If DEFBIND is left as the default, OPEN, the queue manager resolves the alias definition to the bridge queue manager that hosts it. The bridge does not forward the message.

For each queue manager that you want to make visible, you need a queue-manager alias definition. For example, on QM2 you need:

```
DEFINE QREMOTE(QM1) RNAME(' ') RQMNAME(QM1) CLUSTER(WINDSOR)
```

An application connected to any queue manager in WINDSOR, for example QM3, can put a message to any queue on QM1, by naming QM1 explicitly on the MQOPEN call.

Queue-manager aliases and clusters

Use queue-manager aliases to hide the name of queue managers when sending messages into or out of a cluster, and to workload balance messages sent to a cluster.

Queue-manager aliases, which are created using a remote-queue definition with a blank RNAME, have five uses:

Remapping the queue-manager name when sending messages

A queue-manager alias can be used to remap the queue-manager name specified in an MQOPEN call to another queue manager. It can be a cluster queue manager. For example, a queue manager might have the queue-manager alias definition:

```
DEFINE QREMOTE(YORK) RNAME(' ') RQMNAME(CLUSQM)
```

YORK can be used as an alias for the queue manager called CLUSQM. When an application on the queue manager that made this definition puts a message to queue manager YORK, the local queue manager resolves the name to CLUSQM. If the local queue manager is not called CLUSQM, it puts the message on the cluster transmission queue to be moved to CLUSQM. It also changes the transmission header to say CLUSQM instead of YORK.

Note: The definition applies only on the queue manager that makes it. To advertise the alias to the whole cluster, you need to add the CLUSTER attribute to the remote-queue definition. Then messages from other queue managers that were destined for YORK are sent to CLUSQM .

Altering or specifying the transmission queue when sending messages

Aliasing can be used to join a cluster to a non-cluster system. For example, queue managers in the cluster ITALY could communicate with the queue manager called PALERMO , which is outside the cluster. To communicate, one of the queue managers in the cluster must act as a gateway. From the gateway queue manager, issue the command:

```
DEFINE QREMOTE(ROME) RNAME(' ') RQMNAME(PALERMO) XMITQ(X) CLUSTER(ITALY)
```

The command is a queue-manager alias definition. It defines and advertises ROME as a queue manager over which messages from any queue manager in the cluster ITALY can multi-hop to reach their destination at PALERMO. Messages put to a queue opened with the queue-manager name set to ROME are sent to the gateway queue manager with the queue manager alias definition. Once there, the messages are put on the transmission queue X and moved by non-cluster channels to the queue manager PALERMO .

The choice of the name ROME in this example is not significant. The values for QREMOTE and RQMNAME could both be the same.

Determining the destination when receiving messages

When a queue manager receives a message, it extracts the name of the destination queue and queue manager from the transmission header. It looks for a queue-manager alias definition with the same name as the queue manager in the transmission header. if it finds one, it substitutes the RQMNAME from the queue-manager alias definition for the queue manager name in the transmission header.

There are two reasons for using a queue-manager alias in this way:

- To direct messages to another queue manager
- To alter the queue manager name to be the same as the local queue manager

Using queue manager aliases in a gateway queue manager to route messages between queue managers in different clusters.

An application can send a message to a queue in a different cluster using a queue manager alias. The queue does not have to be a cluster queue. The queue is defined in one cluster. The application is connected to a queue manager in a different cluster. A gateway queue manager connects the two clusters. If the queue is not defined as clustered, for the correct routing to take place, the application must open the queue using the queue name and a clustered queue manager alias name. For an example of a configuration, see "Creating two-overlapping clusters with a gateway queue manager" on page 212, from which the reply message flow illustrated in figure 1, is taken.

The diagram shows the path taken by the reply message back to a temporary dynamic queue, which is called RQ. The server application, connected to QM3, opens the reply queue using the queue manager name QM2. The queue manager name QM2 is defined as a clustered queue manager alias on QM1. QM3 routes the reply message to QM1. QM1 routes the message to QM2.

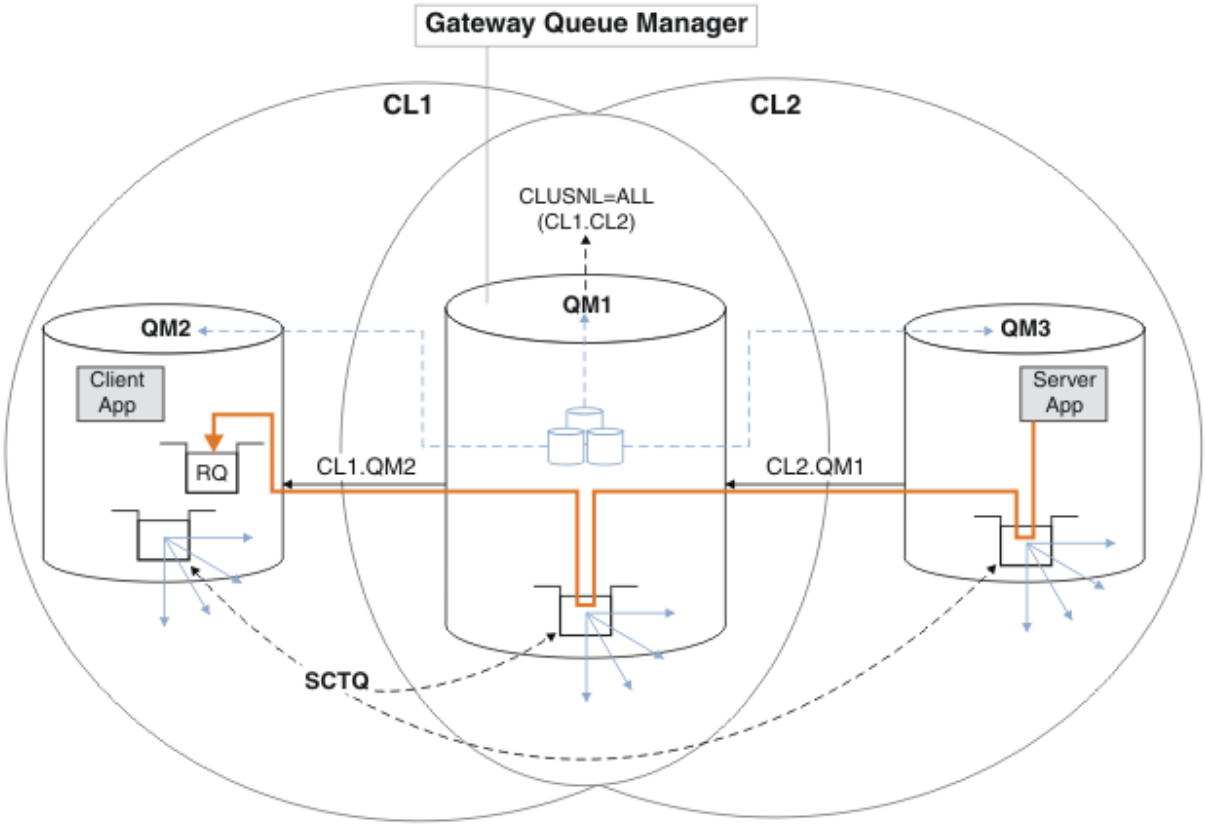


Figure 48. Using a queue manager alias to return the reply message to a different cluster

The way the routing works is as follows. Every queue manager in each cluster has a queue manager alias definition on QM1. The aliases are clustered in all the clusters. The grey dashed arrows from each of the aliases to a queue manager show that each queue manager alias is resolved to a real queue manager in at least one of the clusters. In this case, the QM2 alias is clustered in both cluster CL1 and CL2, and is resolved to the real queue manager QM2 in CL1. The server application creates the reply message using the reply to queue name RQ, and reply to queue manager name QM2. The message is routed to QM1 because the queue manager alias definition QM2 is defined on QM1 in cluster CL2 and queue manager QM2 is not in cluster CL2. As the message cannot be sent to the target queue manager, it is sent to the queue manager that has the alias definition.

QM1 places the message on the cluster transmission queue on QM1 for transferal to QM2. QM1 routes the message to QM2 because the queue manager alias definition on QM1 for QM2 defines QM2 as

the real target queue manager. The definition is not circular, because alias definitions can refer only to real definitions; the alias cannot point to itself. The real definition is resolved by QM1, because both QM1 and QM2 are in the same cluster, CL1. QM1 finds out the connection information for QM2 from the repository for CL1, and routes the message to QM2. For the message to be rerouted by QM1, the server application must have opened the reply queue with the option DEFBIND set to MQBND_BIND_NOT_FIXED. If the server application had opened the reply queue with the option MQBND_BIND_ON_OPEN, the message is not rerouted and ends up on a dead letter queue.

Using a queue manager as a gateway into the cluster to workload balance messages from coming from outside the cluster.

You define a queue called EDINBURGH on more than one queue manager in the cluster. You want the clustering mechanism to balance the workload for messages coming to that queue from outside the cluster.

A queue manager from outside the cluster needs a transmit queue and sender-channel to one queue manager in the cluster. This queue is called a gateway queue manager. To take advantage of the default workload balancing mechanism, one of the following rules must apply:

- The gateway queue manager must not contain an instance of the EDINBURGH queue.
- The gateway queue manager specifies CLWLUSEQ(ANY) on ALTER QMGR.

For an example of workload balancing from outside a cluster, see "Configuring workload balancing from outside a cluster" on page 251

Reply-to queue aliases and clusters

A reply-to queue alias definition is used to specify alternative names for reply information. Reply-to queue alias definitions can be used with clusters just the same as in a distributed queuing environment.

For example:

- An application at queue manager VENICE sends a message to queue manager PISA using the MQPUT call. The application provides the following reply-to queue information in the message descriptor:

```
ReplyToQ='QUEUE'
ReplyToQMgr=''
```

- In order that replies sent to QUEUE can be received on OTHERQ at PISA, create a remote-queue definition on VENICE that is used as a reply-to queue alias. The alias is effective only on the system on which it was created.

```
DEFINE QREMOTE(QUEUE) RNAME(OTHERQ) RQMNAME(PISA)
```

RQMNAME and QREMOTE can specify the same names, even if RQMNAME is itself a cluster queue manager.

Queue aliases and clusters

Use queue aliases to hide the name of a cluster queue, to cluster a queue, adopt different attributes, or adopt different access controls.

A QALIAS definition is used to create an alias by which a queue is to be known. You might create an alias for a number of reasons:

- You want to start using a different queue but you do not want to change your applications.
- You do not want applications to know the real name of the queue to which they are putting messages.
- You might have a naming convention that differs from the one where the queue is defined.
- Your applications might not be authorized to access the queue by its real name but only by its alias.

Create a QALIAS definition on a queue manager using the DEFINE QALIAS command. For example, run the command:

```
DEFINE QALIAS(PUBLIC) TARGET(LOCAL) CLUSTER(C)
```

The command advertises a queue called PUBLIC to the queue managers in cluster C. PUBLIC is an alias that resolves to the queue called LOCAL. Messages sent to PUBLIC are routed to the queue called LOCAL.

You can also use a queue alias definition to resolve a queue name to a cluster queue. For example, run the command:

```
DEFINE QALIAS(PRIVATE) TARGET(PUBLIC)
```

The command enables a queue manager to use the name PRIVATE to access a queue advertised elsewhere in the cluster by the name PUBLIC. Because this definition does not include the CLUSTER attribute it applies only to the queue manager that makes it.

Using clusters for workload management

By defining multiple instances of a queue on different queue managers in a cluster you can spread the work of servicing the queue over multiple servers. There are several factors that can prevent messages being requeued to a different queue manager in the event of failure.

As well as setting up clusters to reduce system administration, you can create clusters in which more than one queue manager hosts an instance of the same queue.

You can organize your cluster such that the queue managers in it are clones of each other. Each queue manager is able to run the same applications and have local definitions of the same queues. You can spread the workload between your queue managers by having several instances of an application. Each instance of the application receives messages and runs independently of each other.

The advantages of using clusters in this way are:

- Increased availability of your queues and applications
- Faster throughput of messages
- More even distribution of workload in your network

Any one of the queue managers that hosts an instance of a particular queue can handle messages destined for that queue. Applications do not name a queue manager when sending messages. A workload management algorithm determines which queue manager handles the message.

See the following subtopics for more information about cluster configurations for workload management:

Related concepts

Clusters

How clusters work

“Comparison of clustering and distributed queuing” on page 157
Compare the components that need to be defined to connect queue managers using distributed queuing and clustering.

“Components of a cluster” on page 159
Clusters are composed of queue managers, cluster repositories, cluster channels, and cluster queues.

“Managing IBM WebSphere MQ clusters” on page 180
You can create, extend, and maintain IBM WebSphere MQ clusters.

“Routing messages to and from clusters” on page 243
Use queue aliases, queue manager aliases, and remote queue definitions to connect clusters to external queue managers and other clusters.

Related tasks

“Configuring a queue manager cluster” on page 154
Use the links in this topic to find out how clusters work, how to design a cluster configuration, and to get an example of how to set up a simple cluster.

“Setting up a new cluster” on page 181

Follow these instructions to set up the example cluster. Separate instructions describe setting up the cluster on TCP/IP, LU 6.2, and with a single transmission queue or multiple transmission queues. Test the cluster works by sending a message from one queue manager to the other.

Writing and compiling cluster workload exits

Example of a cluster with more than one instance of a queue

In this example of a cluster with more than one instance of a queue, messages are routed to different instances of the queue. You can force a message to a specific instance of the queue, and you can choose to send a sequence of messages to one of either of the queue managers.

Figure 49 on page 259 shows a cluster in which there is more than one definition for the queue Q3. If an application at QM1 puts a message to Q3, it does not necessarily know which instance of Q3 is going to process its message. If an application is running on QM2 or QM4, where there are local instances of Q3, the local instance of Q3 is opened by default. By setting the CLWLUSEQ queue attribute, the local instance of the queue can be treated the same as a remote instance of the queue.

The MQOPEN option DefBind controls whether the target queue manager is chosen when the MQOPEN call is issued, or when the message is transferred from the transmission queue.

If you set DefBind to MQBND_BIND_NOT_FIXED the message can be sent to an instance of the queue that is available when the message is transmitted. This avoids the following problems:

- The target queue is unavailable when the message arrives at the target queue manager.
- The state of the queue has changed.
- The message has been put using a cluster queue alias, and no instance of the target queue exists on the queue manager where the instance of the cluster queue alias is defined.

If any if these problems are discovered at transmission time, another available instance of the target queue is sought and the message is rerouted. If no instances of the queue are available, the message is placed on the dead-letter queue.

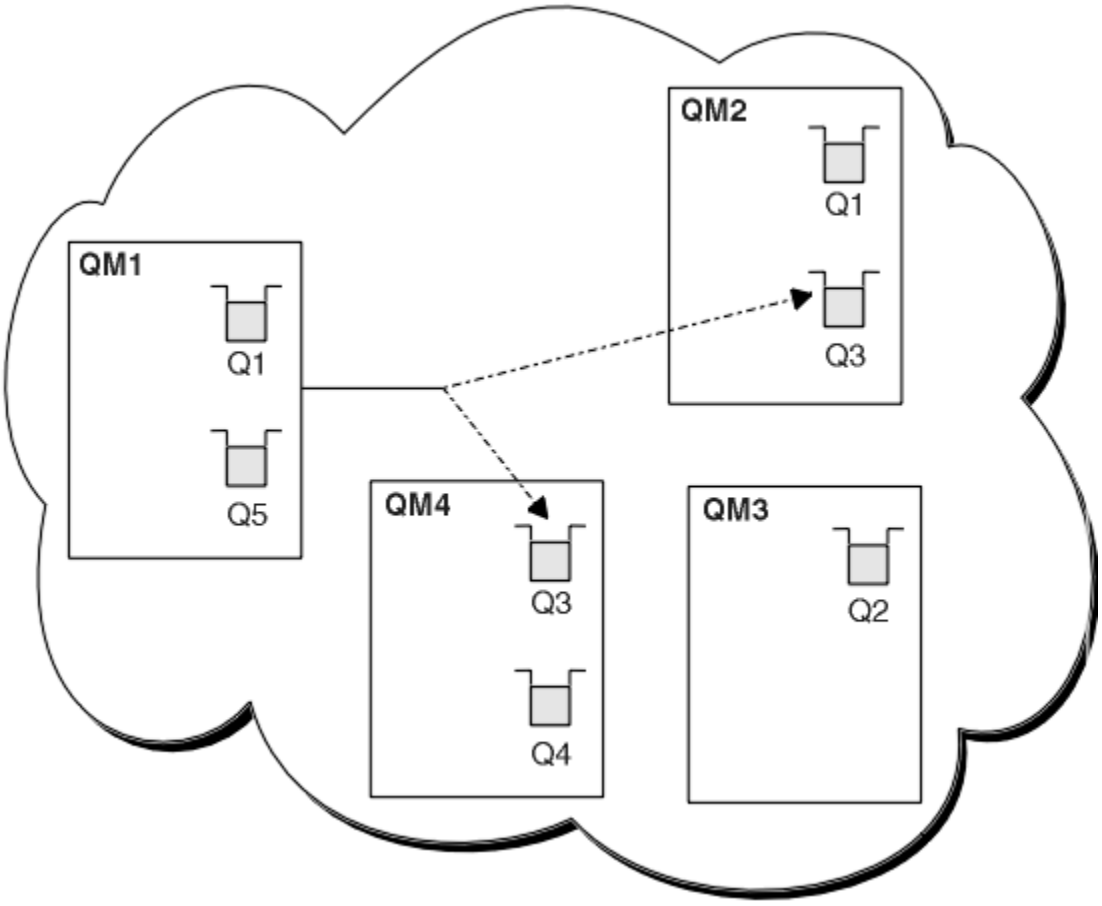


Figure 49. A cluster with multiple instances of the same queue

One factor that can prevent messages being rerouted is if messages have been assigned to a fixed queue manager or channel with MQBND_BIND_ON_OPEN. Messages bound on MQOPEN are never reallocated to another channel. Note also that message reallocation only takes place when a cluster channel is actually failing. Reallocation does not occur if the channel has already failed.

The system attempts to reroute a message if the destination queue manager goes out of service. In so doing, it does not affect the integrity of the message by running the risk of losing it or by creating a duplicate. If a queue manager fails and leaves a message in doubt, that message is not rerouted.

Adding a queue manager that hosts a queue locally

Follow these instructions to add an instance of INVENTQ to provide additional capacity to run the inventory application system in Paris and New York.

Before you begin

Note: For changes to a cluster to be propagated throughout the cluster, at least one full repository must always be available. Ensure that your repositories are available before starting this task.

Scenario:

- The INVENTORY cluster has been set up as described in Adding a new queue manager to a cluster. It contains three queue managers; LONDON and NEWYORK both hold full repositories, PARIS holds a partial repository. The inventory application runs on the system in New York, connected to the NEWYORK queue manager. The application is driven by the arrival of messages on the INVENTQ queue.
- We want to add an instance of INVENTQ to provide additional capacity to run the inventory application system in Paris and New York.

About this task

Follow these steps to add a queue manager that hosts a queue locally.

Procedure

1. Alter the PARIS queue manager.

   For the application in Paris to use the INVENTQ in Paris and the one in New York, we must inform the queue manager. On PARIS issue the following command:

   ```
   ALTER QMGR CLWLUSEQ(ANY)
   ```

2. Review the inventory application for message affinities.

   Before proceeding, ensure that the inventory application does not have any dependencies on the sequence of processing of messages. For more information, see "Handling message affinities" on page 270.

3. Install the inventory application on the system in Paris.
4. Define the cluster queue INVENTQ.

   The INVENTQ queue which is already hosted by the NEWYORK queue manager is also to be hosted by PARIS. Define it on the PARIS queue manager as follows:

   ```
   DEFINE QLOCAL(INVENTQ) CLUSTER(INVENTORY)
   ```

   Now that you have completed all the definitions, if you have not already done so, start the channel initiator on WebSphere MQ for z/OS. On all platforms, start a listener program on queue manager PARIS. The listener listens for incoming network requests and starts the cluster-receiver channel when it is needed.

Results

Figure 50 on page 260 shows the cluster set up by this task.

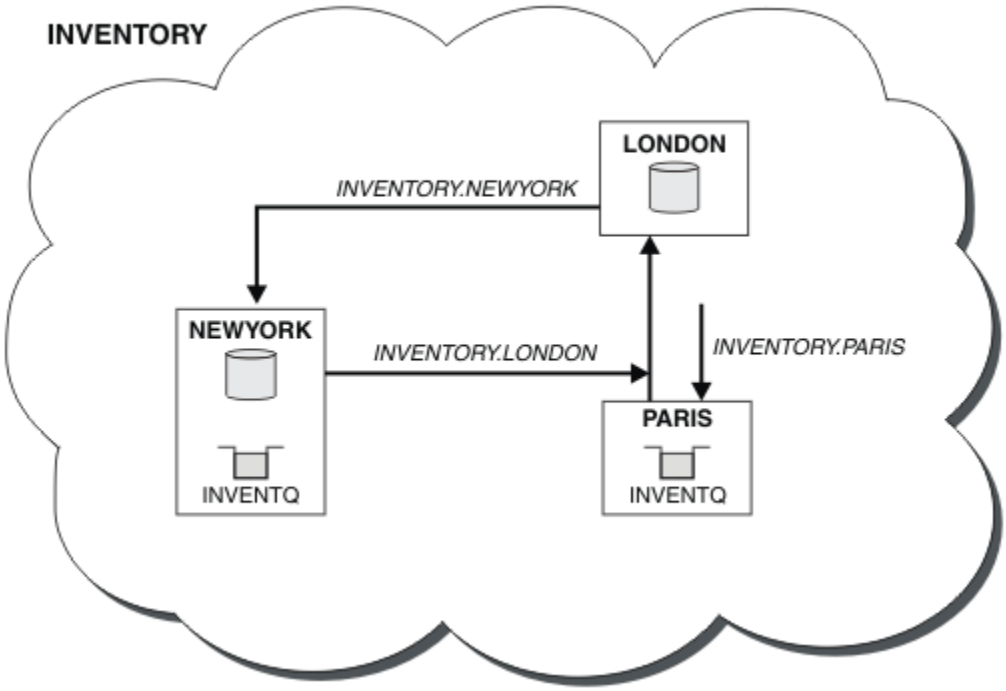


Figure 50. The INVENTORY cluster, with three queue managers

The modification to this cluster was accomplished without you altering the queue managers NEWYORK or LONDON. The full repositories in these queue managers are updated automatically with the information they need to be able to send messages to INVENTQ at PARIS.

What to do next

The INVENTQ queue and the inventory application are now hosted on two queue managers in the cluster. This increases their availability, speeds up throughput of messages, and allows the workload to be distributed between the two queue managers. Messages put to INVENTQ by any of the queue managers LONDON, NEWYORK, PARIS are routed alternately to PARIS or NEWYORK, so that the workload is balanced.

Using two networks in a cluster

Follow these instructions to add a new store in TOKYO where there are two different networks. Both need to be available for use to communicate with the queue manager in Tokyo.

Before you begin

Note: For changes to a cluster to be propagated throughout the cluster, at least one full repository must always be available. Ensure that your repositories are available before starting this task.

Scenario:

- The INVENTORY cluster has been set up as described in "Adding a queue manager to a cluster". It contains three queue managers; LONDON and NEWYORK both hold full repositories, PARIS holds a partial repository. The inventory application runs on the system in New York, connected to the NEWYORK queue manager. The application is driven by the arrival of messages on the INVENTQ queue.
- A new store is being added in TOKYO where there are two different networks. Both need to be available for use to communicate with the queue manager in Tokyo.

About this task

Follow these steps to use two networks in a cluster.

Procedure

1. Decide which full repository TOKYO refers to first.

   Every queue manager in a cluster must refer to one or other of the full repositories to gather information about the cluster. It builds up its own partial repository. It is of no particular significance which repository you choose. In this example, NEWYORK is chosen. Once the new queue manager has joined the cluster it communicates with both of the repositories.

2. Define the CLUSRCVR channels.

   Every queue manager in a cluster needs to define a cluster-receiver on which it can receive messages. This queue manager needs to be able to communicate on each network.

   ```
   DEFINE CHANNEL(INVENTORY.TOKYO.NETB) CHLTYPE(CLUSRCVR) TRPTYPE(TCP)
   CONNAME('TOKYO.NETB.CMSTORE.COM') CLUSTER(INVENTORY) DESCR('Cluster-receiver
   channel using network B for TOKYO')
   ```

   ```
   DEFINE CHANNEL(INVENTORY.TOKYO.NETA) CHLTYPE(CLUSRCVR) TRPTYPE(TCP)
   CONNAME('TOKYO.NETA.CMSTORE.COM') CLUSTER(INVENTORY) DESCR('Cluster-receiver
   channel using network A for TOKYO')
   ```

3. Define a CLUSSDR channel on queue manager TOKYO.

   Every queue manager in a cluster needs to define one cluster-sender channel on which it can send messages to its first full repository. In this case we have chosen NEWYORK, so TOKYO needs the following definition:

```
DEFINE CHANNEL(INVENTORY.NEWYORK) CHLTYPE(CLUSSDR) TRPTYPE(TCP)
CONNAME(NEWYORK.CHSTORE.COM) CLUSTER(INVENTORY) DESCR('Cluster-sender
channel from TOKYO to repository at NEWYORK')
```

Now that you have completed all the definitions, if you have not already done so start the channel initiator on WebSphere MQ for z/OS. On all platforms, start a listener program on queue manager PARIS. The listener program listens for incoming network requests and starts the cluster-receiver channel when it is needed.

Results

Figure 51 on page 262 shows the cluster set up by this task.

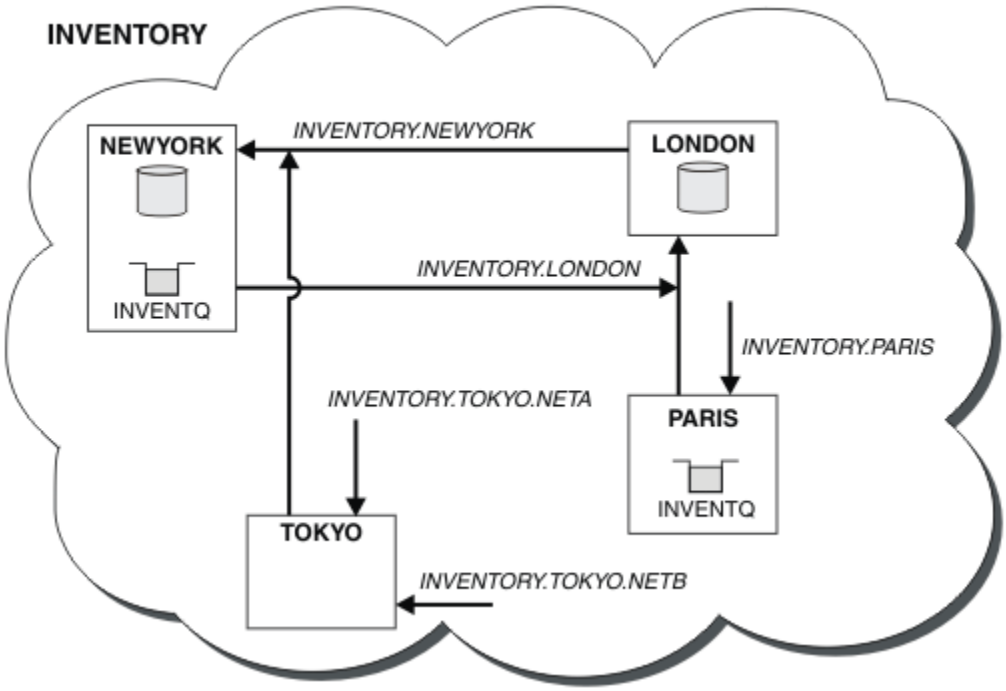


Figure 51. The INVENTORY cluster, with four queue managers

By making only three definitions, we have added the queue manager TOKYO to the cluster with two different network routes available to it.

Related tasks

"Adding a queue manager to a cluster" on page 191
Follow these instructions to add a queue manager to the cluster you created. Messages to cluster queues and topics are transferred using the single cluster transmission queue SYSTEM.CLUSTER.TRANSMIT.QUEUE.

Using a primary and a secondary network in a cluster

Follow these instructions to make one network the primary network, and another network the backup network. Use the backup network if there is a problem with the primary network.

Before you begin

Note: For changes to a cluster to be propagated throughout the cluster, at least one full repository must always be available. Ensure that your repositories are available before starting this task.

Scenario:

- The INVENTORY cluster has been set up as described in “Using two networks in a cluster” on page 261. It contains four queue managers; LONDON and NEWYORK both hold full repositories; PARIS and TOKYO hold partial repositories. The inventory application runs on the system in New York, connected to the queue manager NEWYORK. The TOKYO queue manager has two different networks that it can communicate on.
- You want to make one of the networks the primary network, and another of the networks the backup network. You plan to use the backup network if there is a problem with the primary network.

About this task

Use the NETPRTY attribute to configure a primary and a secondary network in a cluster.

Procedure

Alter the existing CLUSRCVR channels on TOKYO.

To indicate that the network A channel is the primary channel, and the network B channel is the secondary channel, use the following commands:

a) `ALTER CHANNEL(INVENTORY.TOKYO.NETA) CHLTYPE(CLUSRCVR) NETPRTY(2) DESCR('Main cluster-receiver channel for TOKYO')`

b) `ALTER CHANNEL(INVENTORY.TOKYO.NETB) CHLTYPE(CLUSRCVR) NETPRTY(1) DESCR('Backup cluster-receiver channel for TOKYO')`

What to do next

By configuring the channel with different network priorities, you have now defined to the cluster that you have a primary network and a secondary network. The queue managers in the cluster that use these channels automatically use the primary network whenever it is available. The queue managers failover to use the secondary network when the primary network is not available.

Adding a queue to act as a backup

Follow these instructions to provide a backup in Chicago for the inventory system that now runs in New York. The Chicago system is only used when there is a problem with the New York system.

Before you begin

Note: For changes to a cluster to be propagated throughout the cluster, at least one full repository must always be available. Ensure that your repositories are available before starting this task.

Scenario:

- The INVENTORY cluster has been set up as described in “Adding a queue manager to a cluster” on page 191. It contains three queue managers; LONDON and NEWYORK both hold full repositories, PARIS holds a partial repository. The inventory application runs on the system in New York, connected to the NEWYORK queue manager. The application is driven by the arrival of messages on the INVENTQ queue.
- A new store is being set up in Chicago to provide a backup for the inventory system that now runs in New York. The Chicago system only used when there is a problem with the New York system.

About this task

Follow these steps to add a queue to act as a backup.

Procedure

1. Decide which full repository CHICAGO refers to first.

   Every queue manager in a cluster must refer to one or other of the full repositories to gather information about the cluster. It builds up its own partial repository. It is of no particular significance

which repository you choose for any particular queue manager. In this example, NEWYORK is chosen. Once the new queue manager has joined the cluster it communicates with both of the repositories.

2. Define the CLUSRCVR channel.

   Every queue manager in a cluster needs to define a cluster-receiver on which it can receive messages. On CHICAGO, define:

   ```
   DEFINE CHANNEL(INVENTORY.CHICAGO) CHLTYPE(CLUSRCVR) TRPTYPE(TCP)
   CONNAME(CHICAGO.CMSTORE.COM) CLUSTER(INVENTORY) DESCR('Cluster-receiver
   channel for CHICAGO')
   ```

3. Define a CLUSSDR channel on queue manager CHICAGO.

   Every queue manager in a cluster needs to define one cluster-sender channel on which it can send messages to its first full repository. In this case we have chosen NEWYORK, so CHICAGO needs the following definition:

   ```
   DEFINE CHANNEL(INVENTORY.NEWYORK) CHLTYPE(CLUSSDR) TRPTYPE(TCP)
   CONNAME(NEWYORK.CHSTORE.COM) CLUSTER(INVENTORY) DESCR('Cluster-sender
   channel from CHICAGO to repository at NEWYORK')
   ```

4. Alter the existing cluster queue INVENTQ.

   The INVENTQ which is already hosted by the NEWYORK queue manager is the main instance of the queue.

   ```
   ALTER QLOCAL(INVENTQ) CLWLPRTY(2)
   ```

5. Review the inventory application for message affinities.

   Before proceeding, ensure that the inventory application does not have any dependencies on the sequence of processing of messages.

6. Install the inventory application on the system in CHICAGO.

7. Define the backup cluster queue INVENTQ

   The INVENTQ which is already hosted by the NEWYORK queue manager, is also to be hosted as a backup by CHICAGO. Define it on the CHICAGO queue manager as follows:

   ```
   DEFINE QLOCAL(INVENTQ) CLUSTER(INVENTORY) CLWLPRTY(1)
   ```

   Now that you have completed all the definitions, if you have not already done so start the channel initiator on WebSphere MQ for z/OS. On all platforms, start a listener program on queue manager CHICAGO. The listener program listens for incoming network requests and starts the cluster-receiver channel when it is needed.

Results

Figure 52 on page 265 shows the cluster set up by this task.

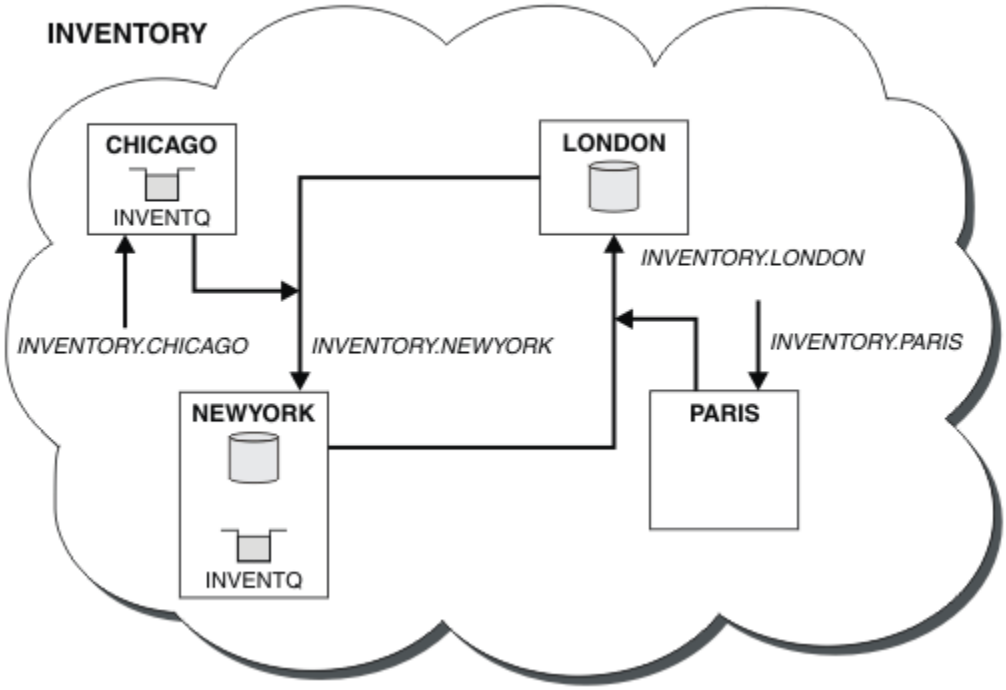


Figure 52. The INVENTORY cluster, with four queue managers

The INVENTQ queue and the inventory application are now hosted on two queue managers in the cluster. The CHICAGO queue manager is a backup. Messages put to INVENTQ are routed to NEWYORK unless it is unavailable when they are sent instead to CHICAGO.

Note:

The availability of a remote queue manager is based on the status of the channel to that queue manager. When channels start, their state changes several times, with some of the states being less preferential to the cluster workload management algorithm. In practice this means that lower priority (backup) destinations can be chosen while the channels to higher priority (primary) destinations are starting.

If you need to ensure that no messages go to a backup destination, do not use CLWLPRTY. Consider using separate queues, or CLWLRANK with a manual switch over from the primary to backup.

Restricting the number of channels used

Follow these instructions to restrict the number of active channels each server runs when a price check application is installed on various queue managers.

Before you begin

Note: For changes to a cluster to be propagated throughout the cluster, at least one full repository must always be available. Ensure that your repositories are available before starting this task.

Scenario:

- A price check application is to be installed on various queue managers. To keep the number of channels being used to a low number, the number of active channels each server runs is restricted. The application is driven by the arrival of messages on the PRICEQ queue.
- Four server queue managers host the price check application. Two query queue managers send messages to the PRICEQ to query a price. Two more queue managers are configured as full repositories.

About this task

Follow these steps to restrict the number of channels used.

Procedure

1. Choose two full repositories.

   Choose two queue managers to be the full repositories for your price check cluster. They are called REPOS1 and REPOS2.

   Issue the following command:

   ```
   ALTER QMGR REPOS(PRICECHECK)
   ```

2. Define a CLUSRCVR channel on each queue manager.

   At each queue manager in the cluster, define a cluster-receiver channel and a cluster-sender channel. It does not matter which is defined first.

   ```
   DEFINE CHANNEL(PRICECHECK.SERVE1) CHLTYPE(CLUSRCVR) TRPTYPE(TCP)
   CONNAME(SERVER1.COM) CLUSTER(PRICECHECK) DESCR('Cluster-receiver channel')
   ```

3. Define a CLUSSDR channel on each queue manager.

   Make a CLUSSDR definition at each queue manager to link that queue manager to one or other of the full repository queue managers.

   ```
   DEFINE CHANNEL(PRICECHECK.REPOS1) CHLTYPE(CLUSSDR) TRPTYPE(TCP)
   CONNAME(REPOS1.COM) CLUSTER(PRICECHECK) DESCR('Cluster-sender channel to
   repository queue manager')
   ```

4. Install the price check application.
5. Define the PRICEQ queue on all the server queue managers.

   Issue the following command on each:

   ```
   DEFINE QLOCAL(PRICEQ) CLUSTER(PRICECHECK)
   ```

6. Restrict the number of channels used by queries

   On the query queue managers we restrict the number of active channels used, by issuing the following commands on each:

   ```
   ALTER QMGR CLWLMRUC(2)
   ```

7. If you have not already done so, start the channel initiator on WebSphere MQ for z/OS. On all platforms, start a listener program.

   The listener program listens for incoming network requests and starts the cluster-receiver channel when it is needed.

Results

Figure 53 on page 267 shows the cluster set up by this task.

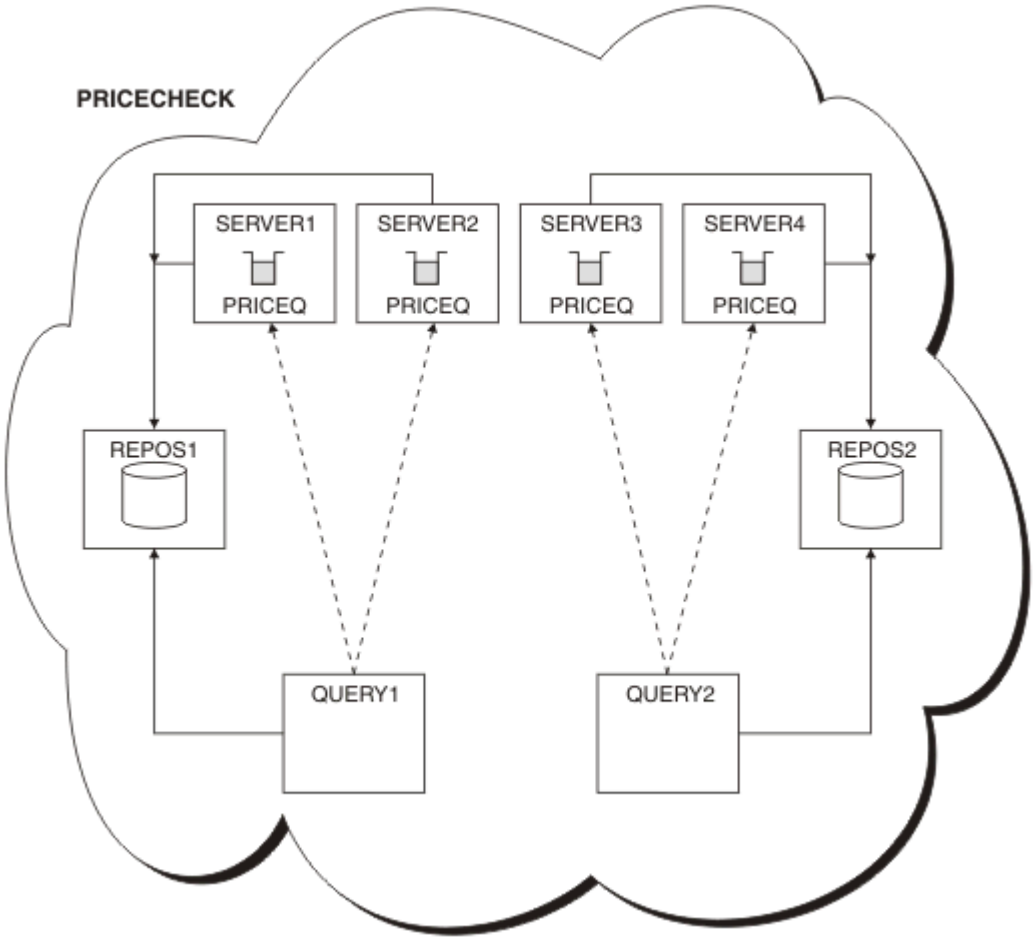


Figure 53. The PRICECHECK cluster, with four server queue managers, two repositories, and two query queue managers

Although there are four instances of the PRICEQ queue available in the PRICECHECK cluster, each querying queue manager only uses two of two of them. For example, the QUERY1 queue manager only has active channels to the SERVER1 and SERVER2 queue managers. If SERVER1 became unavailable, the QUERY1 queue manager would then begin to use another queue manager, for example SERVER3.

What to do next

Although there are four instances of the PRICEQ queue available in the PRICECHECK cluster, each querying queue manager only uses two of two of them. For example, the QUERY1 queue manager only has active channels to the SERVER1 and SERVER2 queue managers. If SERVER1 became unavailable, the QUERY1 queue manager would then begin to use another queue manager, for example SERVER3.

Adding a more powerful queue manager that hosts a queue

Follow these instructions to provide additional capacity by running the inventory system in Los Angeles as well as New York, where Los Angeles can handle twice the number of messages as New York.

Before you begin

Note: For changes to a cluster to be propagated throughout the cluster, at least one full repository must always be available. Ensure that your repositories are available before starting this task.

Scenario:

- The INVENTORY cluster has been set up as described in “Adding a queue manager to a cluster” on page 191. It contains three queue managers: LONDON and NEWYORK both hold full repositories, PARIS holds a partial repository and puts messages from INVENTQ. The inventory application runs on the system in New York connected to the NEWYORK queue manager. The application is driven by the arrival of messages on the INVENTQ queue.
- A new store is being set up in Los Angeles. To provide additional capacity, you want to run the inventory system in Los Angeles as well as New York. The new queue manager can process twice as many messages as New York.

About this task

Follow these steps to add a more powerful queue manager that hosts a queue.

Procedure

1. Decide which full repository LOSANGELES refers to first.
2. Every queue manager in a cluster must refer to one or other of the full repositories to gather information about the cluster. It builds up its own partial repository. It is of no particular significance which repository you choose. In this example, NEWYORK is chosen. Once the new queue manager has joined the cluster it communicates with both of the repositories.

   ```
   DEFINE CHANNEL(INVENTORY.NEWYORK) CHLTYPE(CLUSSDR) TRPTYPE(TCP)
   CONNAME(NEWYORK.CHSTORE.COM) CLUSTER(INVENTORY)
   DESCR('Cluster-sender channel from LOSANGELES to repository at NEWYORK')
   ```

3. Define the CLUSRCVR channel on queue manager LOSANGELES.

   Every queue manager in a cluster must define a cluster-receiver channel on which it can receive messages. On LOSANGELES, define:

   ```
   DEFINE CHANNEL(INVENTORY.LOSANGELES) CHLTYPE(CLUSRCVR) TRPTYPE(TCP)
   CONNAME(LOSANGELES.CHSTORE.COM) CLUSTER(INVENTORY)
   DESCR('Cluster-receiver channel for queue manager LOSANGELES')
   CLWLWGHT(2)
   ```

   The cluster-receiver channel advertises the availability of the queue manager to receive messages from other queue managers in the cluster INVENTORY. Setting CLWLWGHT to two ensures that the Los Angeles queue manager gets twice as many of the inventory messages as New York (when the channel for NEWYORK is set to one).
4. Alter the CLUSRCVR channel on queue manager NEWYORK.

   Ensure that the Los Angeles queue manager gets twice as many of the inventory messages as New York. Alter the definition of the cluster-receiver channel.

   ```
   ALTER CHANNEL(INVENTORY.NEWYORK) CHLTYPE(CLUSRCVR) CLWLWGHT(1)
   ```

5. Review the inventory application for message affinities.

   Before proceeding, ensure that the inventory application does not have any dependencies on the sequence of processing of messages.
6. Install the inventory application on the system in Los Angeles
7. Define the cluster queue INVENTQ.

   The INVENTQ queue, which is already hosted by the NEWYORK queue manager, is also to be hosted by LOSANGELES. Define it on the LOSANGELES queue manager as follows:

   ```
   DEFINE QLOCAL(INVENTQ) CLUSTER(INVENTORY)
   ```

   Now that you have completed all the definitions, if you have not already done so start the channel initiator on WebSphere MQ for z/OS. On all platforms, start a listener program on queue manager LOSANGELES. The listener program listens for incoming network requests and starts the cluster-receiver channel when it is needed.

Results

“Adding a more powerful queue manager that hosts a queue” on page 267 shows the cluster set up by this task.

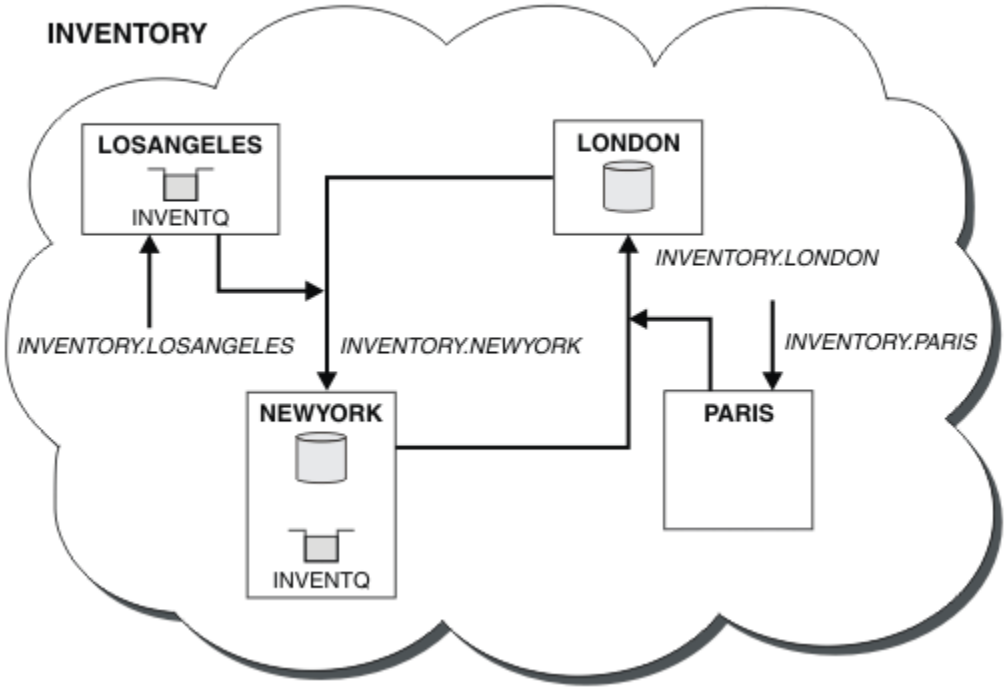


Figure 54. The INVENTORY cluster with four queue managers

This modification to the cluster was accomplished without you having to alter the queue managers LONDON and PARIS. The repositories in these queue managers are updated automatically with the information they need to be able to send messages to INVENTQ at LOSANGELES.

What to do next

The INVENTQ queue and inventory application are hosted on two queue managers in the cluster. The configuration increases their availability, speeds up throughput of messages, and allows the workload to be distributed between the two queue managers. Messages put to INVENTQ by either LOSANGELES or NEWYORK are handled by the instance on the local queue manager whenever possible. Messages put by LONDON or PARIS are routed to LOSANGELES or NEWYORK, with twice as many messages being sent to LOSANGELES.

Application programming and clusters

You do not need to make any programming changes to take advantage of multiple instances of the same queue. However, some programs do not work correctly unless a sequence of messages is sent to the same instance of a queue.

Applications can open a queue using the MQOPEN call. Applications use the MQPUT call to put messages onto an open queue. Applications can put a single message onto a queue that is not already open, using the MQPUT1 call.

If you set up clusters that have multiple instances of the same queue, there are no specific application programming considerations. However, to benefit from the workload management aspects of clustering, you might need to modify your applications. If you set up a network in which there are multiple definitions of the same queue, review your applications for message affinities.

Suppose for example, you have two applications that rely on a series of messages flowing between them in the form of questions and answers. You probably want answers to go back to the same queue manager that sent a question. It is important that the workload management routine does not send the messages to any queue manager that hosts a copy of the reply queue.

You might have applications that require messages to be processed in sequence (for example, a database replication application that sends batches of messages that must be retrieved in sequence). The use of segmented messages can also cause an affinity problem.

Opening a local or remote version of the target queue

Be aware of how the queue manager chooses whether use a local or remote version of the target queue.

1. The queue manager opens the local version of the target queue to read messages, or to set the attributes of the queue.
2. The queue manager opens any instance of the target queue to write messages to, if at least one of the following conditions is true:
   - A local version of the target queue does not exist.
   - The queue manager specifies `CLWLUSEQ(ANY)` on `ALTER QMGR`.
   - The queue on the queue manager specifies `CLWLUSEQ(ANY)`.

Handling message affinities

Message affinities are rarely part of good programming design. You need to remove message affinities to use clustering fully. If you cannot remove message affinities, you can force related messages to be delivered using the same channel and to the same queue manager.

If you have applications with message affinities, remove the affinities before starting to use clusters.

Removing message affinities improves the availability of applications. An application sends a batch of messages that has message affinities to a queue manager. The queue manager fails after receiving only part of the batch. The sending queue manager must wait for it to recover and process the incomplete message batch before it can send any more messages.

Removing messages affinities also improves the scalability of applications. A batch of messages with affinities can lock resources at the destination queue manager while waiting for subsequent messages. These resources might remain locked for long periods of time, preventing other applications from doing their work.

Furthermore, message affinities prevent the cluster workload management routines from making the best choice of queue manager.

To remove affinities, consider the following possibilities:

- Carrying state information in the messages
- Maintaining state information in nonvolatile storage accessible to any queue manager, for example in a Db2 database
- Replicating read-only data so that it is accessible to more than one queue manager

If it is not appropriate to modify your applications to remove message affinities, there are a number of possible solutions to the problem.

Name a specific destination on the `MQOPEN` call

Specify the remote-queue name and the queue manager name on each `MQOPEN` call, and all messages put to the queue using that object handle go to the same queue manager, which might be the local queue manager.

Specifying the remote-queue name and the queue manager name on each `MQOPEN` call has disadvantages:

- No workload balancing is carried out. You do not take advantage of the benefits of cluster workload balancing.
- If the target queue manager is remote and there is more than one channel to it, the messages might take different routes and the sequence of messages is still not preserved.
- If your queue manager has a definition for a transmission queue with the same name as the destination queue manager, messages go on that transmission queue rather than on the cluster transmission queue.

Return the queue-manager name in the reply-to queue manager field

Allow the queue manager that receives the first message in a batch to return its name in its response. It does this using the `ReplyToQMgr` field of the message descriptor. The queue manager at the sending end can then extract the reply-to queue manager name and specify it on all subsequent messages.

Using the `ReplyToQMgr` information from the response has disadvantages:

- The requesting queue manager must wait for a response to its first message
- You must write additional code to find and use the `ReplyToQMgr` information before sending subsequent messages
- If there is more than one route to the queue manager, the sequence of the messages might not be preserved

Set the MQOO_BIND_ON_OPEN option on the MQOPEN call

Force all your messages to be put to the same destination using the `MQOO_BIND_ON_OPEN` option on the `MQOPEN` call. Either `MQOO_BIND_ON_OPEN` or `MQOO_BIND_ON_GROUP` must be specified when using message groups with clusters to ensure that all messages in the group are processed at the same destination.

By opening a queue and specifying `MQOO_BIND_ON_OPEN`, you force all messages that are sent to this queue to be sent to the same instance of the queue. `MQOO_BIND_ON_OPEN` binds all messages to the same queue manager and also to the same route. For example, if there is an IP route and a NetBIOS route to the same destination, one of these is selected when the queue is opened and this selection is honored for all messages put to the same queue using the object handle obtained.

By specifying `MQOO_BIND_ON_OPEN` you force all messages to be routed to the same destination. Therefore applications with message affinities are not disrupted. If the destination is not available, the messages remain on the transmission queue until it becomes available again.

`MQOO_BIND_ON_OPEN` also applies when the queue manager name is specified in the object descriptor when you open a queue. There might be more than one route to the named queue manager. For example, there might be multiple network paths or another queue manager might have defined an alias. If you specify `MQOO_BIND_ON_OPEN`, a route is selected when the queue is opened.

Note: This is the recommended technique. However, it does not work in a multi-hop configuration in which a queue manager advertises an alias for a cluster queue. Nor does it help in situations in which applications use different queues on the same queue manager for different groups of messages.

An alternative to specifying `MQOO_BIND_ON_OPEN` on the `MQOPEN` call, is to modify your queue definitions. On your queue definitions, specify `DEFBIND(OPEN)`, and allow the `DefBind` option on the `MQOPEN` call to default to `MQOO_BIND_AS_Q_DEF`.

Set the MQOO_BIND_ON_GROUP option on the MQOPEN call

Force all your messages in a group to be put to the same destination using the `MQOO_BIND_ON_GROUP` option on the `MQOPEN` call. Either `MQOO_BIND_ON_OPEN` or `MQOO_BIND_ON_GROUP` must be specified when using message groups with clusters to ensure that all messages in the group are processed at the same destination.

By opening a queue and specifying `MQOO_BIND_ON_GROUP`, you force all messages in a group that are sent to this queue to be sent to the same instance of the queue. `MQOO_BIND_ON_GROUP` binds all

messages in a group to the same queue manager, and also to the same route. For example, if there is an IP route and a NetBIOS route to the same destination, one of these is selected when the queue is opened and this selection is honored for all messages in a group put to the same queue using the object handle obtained.

By specifying MQOO_BIND_ON_GROUP you force all messages in a group to be routed to the same destination. Therefore applications with message affinities are not disrupted. If the destination is not available, the messages remain on the transmission queue until it becomes available again.

MQOO_BIND_ON_GROUP also applies when the queue manager name is specified in the object descriptor when you open a queue. There might be more than one route to the named queue manager. For example, there might be multiple network paths or another queue manager might have defined an alias. If you specify MQOO_BIND_ON_GROUP, a route is selected when the queue is opened.

For MQOO_BIND_ON_GROUP to be effective you must include the MQPMO_LOGICAL_ORDER put option on MQPUT. You can set **GroupId** in the MQMD of the message to MQGI_NONE, and you must include the following message flags within the MQMD **MsgFlags** field of the messages:

- Last message in group: MQMF_LAST_MSG_IN_GROUP
- All other messages in group: MQMF_MSG_IN_GROUP

If MQOO_BIND_ON_GROUP is specified but the messages are not grouped, the behavior is equivalent to MQOO_BIND_NOT_FIXED.

Note: This is the recommended technique for ensuring that messages in a group are sent to the same destination. However, it does not work in a multi-hop configuration in which a queue manager advertises an alias for a cluster queue.

An alternative to specifying MQOO_BIND_ON_GROUP on the MQOPEN call, is to modify your queue definitions. On your queue definitions, specify DEFBIND(GROUP), and allow the DefBind option on the MQOPEN call to default to MQOO_BIND_AS_Q_DEF.

Write a customized cluster workload exit program

Instead of modifying your applications you can circumvent the message affinities problem by writing a cluster workload exit program. Writing a cluster workload exit program is not easy and is not a recommended solution. The program would have to be designed to recognize the affinity by inspecting the content of messages. Having recognized the affinity, the program would have to force the workload management utility to route all related messages to the same queue manager.

Clustering: Best practices

Clusters provide a mechanism for interconnecting queue managers. The best practices described in this section are based on testing and feedback from customers.

A successful cluster setup is dependent on good planning and a thorough understanding of IBM WebSphere MQ fundamentals, such as good application management and network design. Ensure that you are familiar with the information in the related topics listed below before continuing.

Related concepts

Clustering

Concepts of intercommunication

How clusters work

Clustering: Special considerations for overlapping clusters

This topic provides guidance for planning and administering IBM WebSphere MQ clusters. This information is a guide based on testing and feedback from customers.

Cluster ownership

Familiarize yourself with overlapping clusters before reading the following information. See “Overlapping clusters” on page 175 and “Configuring message paths between clusters” on page 252 for the necessary information.

When configuring and managing a system that consists of overlapping clusters, it is best to adhere to the following:

- Although IBM WebSphere MQ clusters are 'loosely coupled' as previously described, it is useful to consider a cluster as a single unit of administration. This concept is used because the interaction between definitions on individual queue managers is critical to the smooth functioning of the cluster. For example: When using workload balanced cluster queues it is important that a single administrator or team understand the full set of possible destinations for messages, which depends on definitions spread throughout the cluster. More trivially, cluster sender/receiver channel pairs must be compatible throughout.
- Considering this previous concept; where multiple clusters meet (which are to be administered by separate teams / individuals), it is important to have clear policies in place controlling administration of the gateway queue managers.
- It is useful to treat overlapping clusters as a single namespace: Channel names and queue manager names must be unique throughout a single cluster. Administration is much easier when unique throughout the entire topology. It is best to follow a suitable naming convention, possible conventions are described in “Cluster naming conventions” on page 174.
- Sometimes administrative and system management cooperation is essential/unavoidable: For example, cooperation between organizations that own different clusters that need to overlap. A clear understanding of who owns what, and enforceable rules/conventions helps clustering run smoothly when overlapping clusters.

Overlapping clusters: Gateways

In general, a single cluster is easier to administer than multiple clusters. Therefore creating large numbers of small clusters (one for every application for example) is something to be avoided generally.

However, to provide classes of service, you can implement overlapping clusters. For example:

- If you have concentric clusters where the smaller one is for Publish/Subscribe. See How to size systems for more information.
- If some queue managers are to be administered by different teams. See the previous section “Cluster ownership” on page 273for more information.
- If it makes sense from an organizational or geographical point of view.
- If equivalent clusters work with name resolution, for example when implementing SSL or TLS in an existing cluster.

There is no security benefit from overlapping clusters; allowing clusters administered by two different teams to overlap, effectively joins the teams as well as the topology. Any:

- Name advertised in such a cluster is accessible to the other cluster.
- Name advertised in one cluster can be advertised in the other to draw off eligible messages.
- Non-advertised object on a queue manager adjacent to the gateway can be resolved from any clusters of which the gateway is a member.

The namespace is the union of both clusters and must be treated as a single namespace. Therefore, ownership of an overlapping cluster is shared amongst all the administrators of both clusters.

When a system contains multiple clusters, there might be a requirement to route messages from queue managers in one cluster to queues on queue managers in another cluster. In this situation, the multiple clusters must be interconnected in some way: A good pattern to follow is the use of gateway queue managers between clusters. This arrangement avoids building up a difficult-to-manage mesh of point-to-point channels, and provides a good place to manage such issues as security policies. There are two distinct ways of achieving this arrangement:

1. Place one (or more) queue managers in both clusters using a second cluster receiver definition. This arrangement involves fewer administrative definitions but, as previously stated, means that ownership of an overlapping cluster is shared amongst all the administrators of both clusters.
2. Pair a queue manager in cluster one with a queue manager in cluster teo using traditional point-to-point channels.

In either of these cases, various tools can be used to route traffic appropriately. In particular, queue or queue manager aliases can be used to route into the other cluster, and a queue manager alias with blank **RQMNAME** property re-drives workload balancing where it is wanted.

Related concepts

"Cluster naming conventions" on page 174
Consider naming queue managers in the same cluster using a naming convention that identifies the cluster to which the queue manager belongs. Use a similar naming convention for channel names and extend it to describe the channel characteristics.

Clustering: Topology design considerations

This topic provides guidance for planning and administering IBM WebSphere MQ clusters. This information is a guide based on testing and feedback from customers.

By thinking about where user applications and internal administrative processes are going to be located in advance, many problems can either be avoided, or minimized at a later date. This topic contains information about design decisions that can improve performance, and simplify maintenance tasks as the cluster scales.

Performance of the clustering infrastructure

When an application tries to open a queue on a queue manager in a cluster, the queue manager registers its interest with the full repositories for that queue so that it can learn where the queue exists in the cluster. Any updates to the queue location or configuration are automatically sent by the full repositories to the interested queue manager. This registering of interest is internally known as a subscription (these subscriptions are not the same as IBM WebSphere MQ subscriptions used for publish/subscribe messaging in IBM WebSphere MQ)

All information about a cluster goes through every full repository. Full repositories are therefore always being used in a cluster for administrative message traffic. The high usage of system resources when managing these subscriptions, and the transmission of them and the resulting configuration messages, can cause a considerable load on the clustering infrastructure. There are a number of things to consider when ensuring that this load is understood and minimized wherever possible:

- The more individual queue managers using a cluster queue, the more subscriptions are in the system, and thus the bigger the administrative overhead when changes occur and interested subscribers need to be notified, especially on the full repository queue managers. One way to minimize unnecessary traffic and full repository load is by connecting similar applications (that is, those applications that work with the same queues) to a smaller number of queue managers.

- In addition to the number of subscriptions in the system affecting the performance the rate of change in the configuration of clustered objects can affect performance, for example the frequent changing of a clustered queue configuration.
- When a queue manager is a member of multiple clusters (that is, it is part of an overlapping cluster system) any interest made in a queue results in a subscription for each cluster it is a member of, even if the same queue managers are the full repositories for more than one of the clusters. This arrangement increases the load on the system, and is one reason to consider whether multiple overlapping clusters are necessary, rather than a single cluster.
- Application message traffic (that is, the messages being sent byIBM WebSphere MQ applications to the cluster queues) does not go via the full repositories to reach the destination queue managers. This message traffic is sent directly between the queue manager where the message enters the cluster, and the queue manager where the cluster queue exists. It is not therefore necessary to accommodate high rates of application message traffic with respect to the full repository queue managers, unless the full repository queue managers happen to be either of those two queue managers mentioned. For that reason, it is recommended that full repository queue managers are not used for application message traffic in clusters where the clustering infrastructure load is significant.

Full repositories

A repository is a collection of information about the queue managers that are members of a cluster. A queue manager that hosts a complete set of information about every queue manager in the cluster has a full repository. For more information about full repositories and partial repositories, see “Cluster repository” on page 160.

Full repositories must be held on servers that are reliable and as highly available as possible and single points of failure must be avoided. The cluster design must always have two full repositories. If there is a failure of a full repository, the cluster can still operate.

Details of any updates to cluster resources made by a queue manager in a cluster; for example, clustered queues, are sent from that queue manager to two full repositories at most in that cluster (or to one if there is only one full repository queue manager in the cluster). Those full repositories hold the information and propagate it to any queue managers in the cluster that show an interest in it (that is, they subscribe to it). To ensure that each member of the cluster has an up-to-date view of the cluster resources there, each queue manager must be able to communicate with at least one full repository queue manager at any one time.

If, for any reason a queue manager cannot communicate with any full repositories, it can continue to function in the cluster based on its already cached level of information for a period time, but no new updates or access to previously unused cluster resources are available.

For this reason, you must aim to keep the two full repositories available at all times. However, this arrangement does not mean that extreme measures must be taken because the cluster functions adequately for a short while without a full repository.

There is another reason that a cluster must have two full repository queue managers, other than the availability of cluster information: This reason is to ensure that the cluster information held in the full repository cache exists in two places for recovery purposes. If there is only one full repository, and it loses its information about the cluster, then manual intervention on all queue managers within the cluster is required in order to get the cluster working again. If there are two full repositories however, then because information is always published to and subscribed for from two full repositories, the failed full repository can be recovered with the minimum of effort.

- It is possible to perform maintenance on full repository queue managers in a two full repository cluster design without impacting users of that cluster: The cluster continues to function with only one repository, so where possible bring the repositories down, apply the maintenance, and back up again one at a time. Even if there is an outage on the second full repository, running applications are unaffected for a minimum of three days.
- Unless there is a good reason for using a third repository, such as using a geographically local full repository for geographical reasons, use the two repository design. Having three full repositories means

that you never know which are the two that are currently in use, and there might be administrative problems caused by interactions between multiple workload management parameters. It is not recommend to have more than two full repositories.

- If you still need better availability, consider hosting the full repository queue managers as multi-instance queue managers or using platform specific high availability support to improve their availability.
- You must fully interconnect all the full repository queue managers with manually defined cluster sender channels. Particular care must be taken when the cluster does have, for some justifiable reason, more than two full repositories. In this situation it is often possible to miss one or more channels and for it not to be immediately apparent. When full interconnection does not occur, hard to diagnose problems often arise. They are hard to diagnose because some full repositories not holding all repository data and therefore resulting in queue managers in the cluster having different views of the cluster depending on the full repositories that they connect to.

Should applications use queues on full repositories?

A full repository is in most ways exactly like any other queue manager, and it is therefore possible to host application queues on the full repository and connect applications directly to these queue managers. Should applications use queues on full repositories?

The commonly accepted answer is "No?. Although this configuration is possible, many customers prefer to keep these queue managers dedicated to maintaining the full repository cluster cache. Points to consider when deciding on either option are described here, but ultimately the cluster architecture must be appropriate to the particular demands of the environment.

- Upgrades: Usually, in order to use new cluster features in new releases of IBM WebSphere MQ the full repository queue managers of that cluster must be upgraded first. When an application in the cluster wants to use new features, it might be useful to be able to update the full repositories (and some subset of partial repositories) without testing a number of co-located applications.
- Maintenance: In a similar way if you must apply urgent maintenance to the full repositories, they can be restarted or refreshed with the **REFRESH** command without touching applications.
- Performance: As clusters grow and demands on the full repository cluster cache maintenance become greater, keeping applications separate reduces risk of this affecting application performance through contention for system resources.
- Hardware requirements: Typically, full repositories do not need to be powerful; for example, a simple UNIX server with a good expectation of availability is sufficient. Alternatively, for very large or constantly changing clusters, the performance of the full repository computer must be considered.
- Software requirements: Requirements are usually the main reason for choosing to host application queues on a full repository. In a small cluster, collocation might mean a requirement for fewer queue managers/servers over all.

Managing channel definitions

Even within a single cluster, multiple channel definitions can exist giving multiple routes between two queue managers.

There is sometimes an advantage to having parallel channels within a single cluster, but this design decision must be considered thoroughly; apart from adding complexity, this design might result in channels being under-utilized which reduces performance. This situation occurs because testing usually involves sending lots of messages at a constant rate, so the parallel channels are fully used. But with real-world conditions of a non-constant stream of messages, the workload balancing algorithm causes performance to drop as the message flow is switched from channel to channel.

When a queue manager is a member of multiple clusters, the option exists to use a single channel definition with a cluster namelist, rather than defining a separate CLUSRCVR channel for each cluster. However, this setup can cause administration difficulties later; consider for example the case where SSL is to be applied to one cluster but not a second. It is therefore preferable to create separate definitions, and the naming convention suggested in “Cluster naming conventions” on page 174 supports this.

Workload balancing over multiple channels

This information is intended as an advanced understanding of the subject. For the basic explanation of this subject (which must be understood before using the information here), see "Using clusters for workload management" on page 257, Workload balancing , and The cluster workload management algorithm.

The cluster workload management algorithm provides a large set of tools, but they must not all be used with each other without fully understanding how they work and interact. It might not be immediately obvious how important channels are to the workload balancing process: The workload management round-robin algorithm behaves as though multiple cluster channels to a queue manager that owns a clustered queue, are treated as multiple instances of that queue. This process is explained in more detail in the following example:

1. There are two queue managers hosting a queue in a cluster: QM1 and QM2.
2. There are five cluster receiver channels to QM1.
3. There is only one cluster receiver channel to QM2.
4. When **MQPUT** or **MQOPEN** on QM3 chooses an instance, the algorithm is five times more likely to send the message to QM1 than to QM2.
5. The situation in step 4 occurs because the algorithm sees six options to choose from (5+1) and round-robins across all five channels to QM1 and the single channel to QM2.

Another subtle behavior is that even when putting messages to a clustered queue that happens to have one instance configured on the local queue manager, IBM WebSphere MQ uses the state of the local cluster receiver channel to decide whether messages are to be put to the local instance of the queue or remote instances of the queue. In this scenario:

1. When putting messages the workload management algorithm does not look at individual cluster queues, it looks at the cluster channels which can reach those destinations.
2. To reach local destinations, the local receiver channels are included in this list (although they are not used to send the message).
3. When a local receiver channel is stopped, the workload management algorithm, prefers an alternative instance by default if its CLUSRCVR is not stopped. If there are multiple local CLUSRCVR instances for the destination and at least one is not stopped, the local instance remains eligible.

Clustering: Application isolation using multiple cluster transmission queues

You can isolate the message flows between queue managers in a cluster. You can place messages being transported by different cluster-sender channels onto different cluster transmission queues. You can use the approach in a single cluster or with overlapping clusters. The topic provides examples and some best practices to guide you in choosing an approach to use.

When you deploy an application, you have a choice over which IBM WebSphere MQ resources it shares with other applications and which resources it does not share. There are a number of types of resources that can be shared, the main ones being the server itself, the queue manager, channels, and queues. You can choose to configure applications with fewer shared resources; allocating separate queues, channels, queue managers, or even servers to individual applications. If you do so, the overall system configuration becomes bigger and more complex. Using IBM WebSphere MQ clusters reduces the complexity of managing more servers, queue managers, queues, and channels, but it introduces another shared resource, the cluster transmission queue, SYSTEM.CLUSTER.TRANSMIT.QUEUE.

Figure 55 on page 279 is a slice out of a large IBM WebSphere MQ deployment that illustrates the significance of sharing SYSTEM.CLUSTER.TRANSMIT.QUEUE. In the diagram, the application, Client App, is connected to the queue manager QM2 in cluster CL1. A message from Client App is processed by the application, Server App. The message is retrieved by Server App from the cluster queue Q1 on the queue manager QM3 in CLUSTER2. Because the client and server applications are not in the same cluster, the message is transferred by the gateway queue manager QM1.

The normal way to configure a cluster gateway is to make the gateway queue manager a member of all the clusters. On the gateway queue manager are defined clustered alias queues for cluster

queues in all the clusters. The clustered queue aliases are available in all the clusters. Messages put to the cluster queue aliases are routed via the gateway queue manager to their correct destination. The gateway queue manager puts messages sent to the clustered alias queues onto the common SYSTEM.CLUSTER.TRANSMIT.QUEUE on QM1.

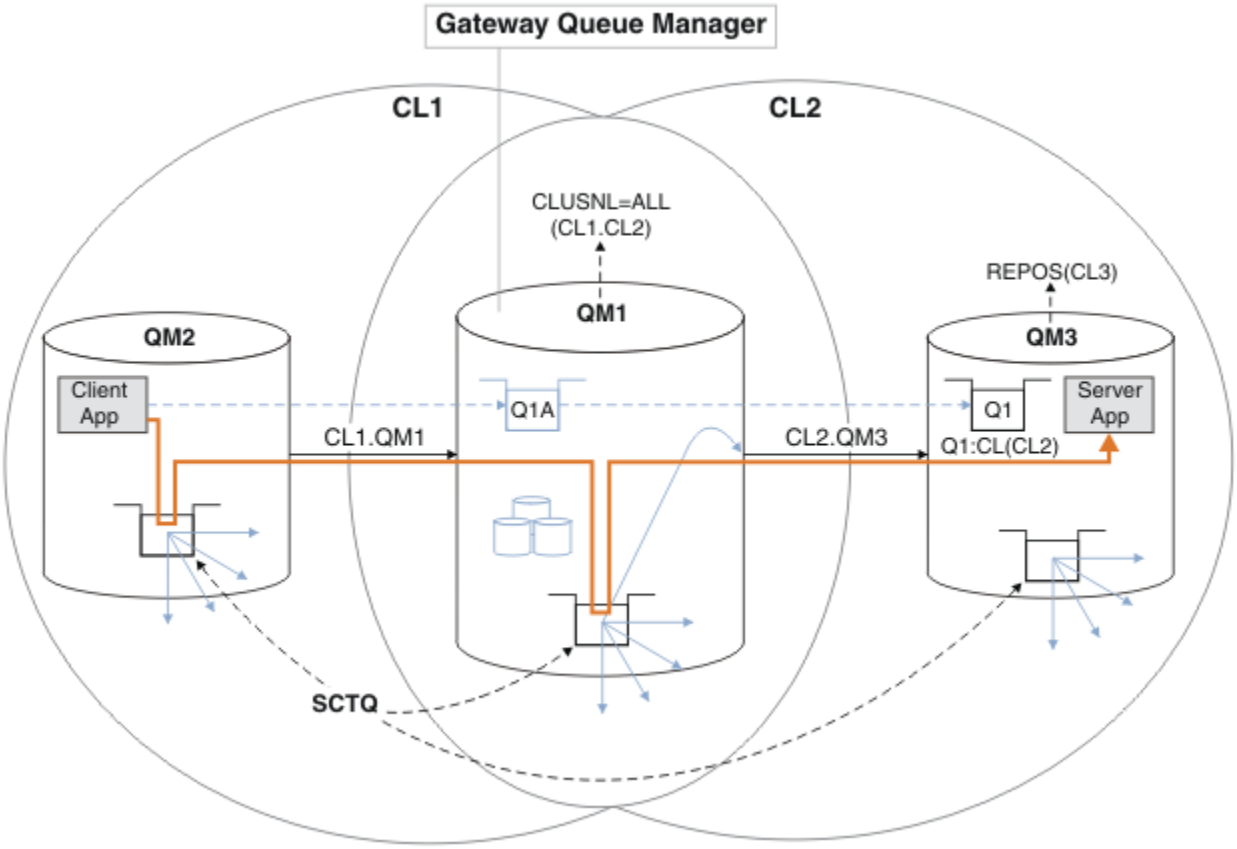
The hub and spoke architecture requires all messages between clusters to pass through the gateway queue manager. The result is that all messages flow through the single cluster transmission queue on QM1, SYSTEM.CLUSTER.TRANSMIT.QUEUE.

From a performance perspective, a single queue is not a problem. A common transmission queue generally does not represent a performance bottleneck. Message throughput on the gateway is largely determined by the performance of the channels that connect to it. Throughput is not generally affected by the number of queues, or the number of messages on the queues that use the channels.

From some other perspectives, using a single transmission queue for multiple applications has drawbacks:

- You cannot isolate the flow of messages to one destination from the flow of messages to another destination. You cannot separate the storage of messages before they are forwarded, even if the destinations are in different clusters on different queue managers.

  If one cluster destination becomes unavailable, messages for that destination build-up in the single transmission queue, and eventually the messages fill it up. Once the transmission queue is full, it stops messages from being placed onto the transmission queue for any cluster destination.
- It is not easy to monitor the transfer of messages to different cluster destinations. All the messages are on the single transmission queue. Displaying the depth of the transmission queue gives you little indication whether messages are being transferred to all destinations.



Note: The arrows in Figure 55 on page 279 and following figures are of different types. Solid arrows represent message flows. The labels on solid arrows are message channel names. The gray solid arrows are potential message flows from the SYSTEM.CLUSTER.TRANSMIT.QUEUE onto cluster-sender channels. Black dashed lines connect labels to their targets. Gray dashed arrows are references; for example from an MQOPEN call by `Client` App to the cluster alias queue definition Q1A.

Figure 55. Client-server application deployed to hub and spoke architecture using IBM WebSphere MQ clusters

In Figure 55 on page 279, clients of `Server` App open the queue Q1A. Messages are put to SYSTEM.CLUSTER.TRANSMIT.QUEUE on QM2, transferred to SYSTEM.CLUSTER.TRANSMIT.QUEUE on QM1, and then transferred to Q1 on QM3, where they are received by the `Server` App application.

The message from `Client` App passes through system cluster transmission queues on QM2 and QM1. In Figure 55 on page 279, the objective is to isolate the message flow on the gateway queue manager from the client application, so that its messages are not stored on SYSTEM.CLUSTER.TRANSMIT.QUEUE. You can isolate flows on any of the other clustered queue managers. You can also isolate flows in the other direction, back to the client. To keep the descriptions of the solutions brief, the descriptions consider only a single flow from the client application.

Solutions to isolating cluster message traffic on a cluster gateway queue manager

One way to solve the problem is to use queue manager aliases, or remote queue definitions, to bridge between clusters. Create a clustered remote queue definition, a transmission queue, and a channel, to separate each message flow on the gateway queue manager; see “Adding a remote queue definition to isolate messages sent from a gateway queue manager” on page 195.

From Version 7.5 onwards, cluster queue managers are not limited to a single cluster transmission queue. You have two choices:

1. Define additional cluster transmission queues manually, and define which cluster-sender channels transfer messages from each transmission queue; see “Adding a cluster transmit queue to isolate cluster message traffic sent from a gateway queue manager” on page 198.
2. Allow the queue manager to create and manage additional cluster transmission queues automatically. It defines a different cluster transmission queue for each cluster-sender channel; see “Changing the default to separate cluster transmission queues to isolate message traffic” on page 218

You can combine manually defined cluster transmission queues for some cluster-sender channels, with the queue manager managing the rest. The combination of transmission queues is the approach taken in “Adding a cluster transmit queue to isolate cluster message traffic sent from a gateway queue manager” on page 198. In that solution, most messages between clusters use the common `SYSTEM.CLUSTER.TRANSMIT.QUEUE`. One application is critical, and all its message flows are isolated from other flows by using one manually defined cluster transmission queue.

The configuration in “Adding a cluster transmit queue to isolate cluster message traffic sent from a gateway queue manager” on page 198is limited. It does not separate the message traffic going to a cluster queue on the same queue manager in the same cluster as another cluster queue. You can separate the message traffic to individual queues by using the remote queue definitions that are part of distributed queueing. With clusters, using multiple cluster transmission queues, you can separate message traffic that goes to different cluster-sender channels. Multiple cluster queues in the same cluster, on the same queue manager, share a cluster-sender channel. Messages for those queues are stored on the same transmission queue, before being forwarded from the gateway queue manager. In the configuration in “Adding a cluster and a cluster transmit queue to isolate cluster message traffic sent from a gateway queue manager” on page 201, the limitation is side-stepped by adding another cluster and making the queue manager and cluster queue a member of the new cluster. The new queue manager might be the only queue manager in the cluster. You could add more queue managers to the cluster, and use the same cluster to isolate cluster queues on those queue managers as well.

Related concepts

“Access control and multiple cluster transmission queues” on page 156
Choose between three modes of checking when an application puts messages to remote cluster queues. The modes are checking remotely against the cluster queue, checking locally against `SYSTEM.CLUSTER.TRANSMIT.QUEUE`, or checking against local profiles for the cluster queue, or cluster queue manager.

“Clustering: Special considerations for overlapping clusters” on page 273
This topic provides guidance for planning and administering IBM WebSphere MQ clusters. This information is a guide based on testing and feedback from customers.

“Cluster transmission queues and cluster-sender channels” on page 166
Messages between clustered queue managers are stored on cluster transmission queues and forwarded by cluster-sender channels.

“Overlapping clusters” on page 175
Overlapping clusters provide additional administrative capabilities. Use namelists to reduce the number of commands needed to administer overlapping clusters.

Related tasks

Authorizing putting messages on remote cluster queues
“Adding a remote queue definition to isolate messages sent from a gateway queue manager” on page 195
Modify the configuration of overlapping clusters that use a gateway queue manager. After the modification messages are transferred to an application from the gateway queue manager without using the same transmission queue or channels as other cluster messages. The solution uses a clustered queue remote definition, and a separate sender channel and transmission queue.

“Adding a cluster transmit queue to isolate cluster message traffic sent from a gateway queue manager” on page 198
Modify the configuration of overlapping clusters that use a gateway queue manager. After the modification messages are transferred to an application from the gateway queue manager without using the same transmission queue or channels as other cluster messages. The solution uses an additional cluster transmission queue to separate message traffic to a single queue manager in a cluster.

"Adding a cluster and a cluster transmit queue to isolate cluster message traffic sent from a gateway queue manager" on page 201
Modify the configuration of overlapping clusters that use a gateway queue manager. After the modification messages are transferred to an application from the gateway queue manager without using the same transmission queue or channels as other cluster messages. The solution uses an additional cluster to isolate the messages to a particular cluster queue.

"Changing the default to separate cluster transmission queues to isolate message traffic" on page 218
You can change the default way a queue manager stores messages for a clustered queue or topic on a transmission queue. Changing the default provides you with a way to isolate cluster messages on a gateway queue manager.

"Creating two-overlapping clusters with a gateway queue manager" on page 212
Follow the instructions in the task to construct overlapping clusters with a gateway queue manager. Use the clusters as a starting point for the following examples of isolating messages to one application from messages to other applications in a cluster.

"Configuring message paths between clusters" on page 252
Connect clusters together using a gateway queue manager. Make queues or queue managers visible to all the clusters by defining cluster queue or cluster queue manager aliases on the gateway queue manager.

"Clustering: Planning how to configure cluster transmission queues" on page 281
You are guided through the choices of cluster transmission queues. You can configure one common default queue, separate default queues, or manually defined queues. Configuring multiple cluster transmission queues applies to platforms other than z/OS.

Related reference

"Security" on page 419
Use the `Security` stanza in the `qm.ini` file to specify options for the Object Authority Manager (OAM).

setmqaut

Clustering: Planning how to configure cluster transmission queues

You are guided through the choices of cluster transmission queues. You can configure one common default queue, separate default queues, or manually defined queues. Configuring multiple cluster transmission queues applies to platforms other than z/OS.

Before you begin

Review "How to choose what type of cluster transmission queue to use" on page 284.

About this task

You have some choices to make when you are planning how to configure a queue manager to select a cluster transmission queue.

1. What is the default cluster transmission queue for cluster message transfers?
   a. A common cluster transmission queue, `SYSTEM.CLUSTER.TRANSMIT.QUEUE`.
   b. Separate cluster transmission queues. The queue manager manages the separate cluster transmission queues. It creates them as permanent-dynamic queues from the model queue, `SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE`. It creates one cluster transmission queue for each cluster-sender channel it uses.
2. For the cluster transmission queues that you do decide to create manually, you have a further two choices:
   a. Define a separate transmission queue for each cluster-sender channel that you decide to configure manually. In this case, set the **CLCHNAME** queue attribute of the transmission queue to the name of a cluster-sender channel. Select the cluster-sender channel that is to transfer messages from this transmission queue.

b. Combine message traffic for a group of cluster-sender channels onto the same cluster transmission queue; see Figure 56 on page 282. In this case, set the **CLCHNAME** queue attribute of each common transmission queue to a generic cluster-sender channel name. A generic cluster-sender channel name is a filter to group cluster-sender channel names. For example, SALES.* groups all cluster-sender channels that have names beginning with SALES.. You can place multiple wildcard characters anywhere in the filter-string. The wildcard character is an asterisk, "*". It represents from zero to any number of characters.

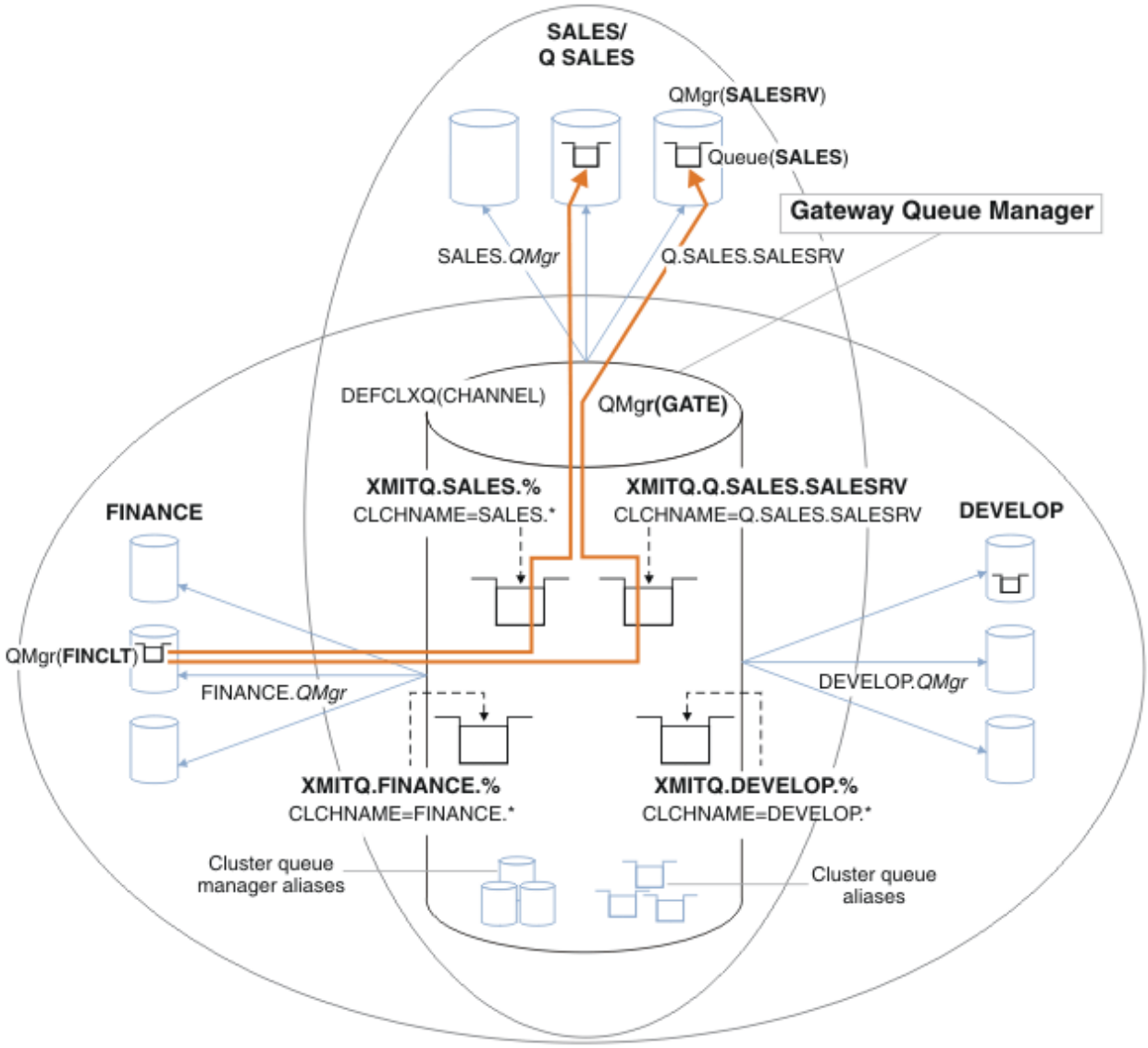


Figure 56. Example of specific transmission queues for different departmental IBM WebSphere MQ clusters

Procedure

1. Select the type of default cluster transmission queue to use.
   - Choose a single cluster transmission queue, or separate queues for each cluster connection.

   Leave the default setting or run the **MQSC** command:

   ```
   ALTER QMGR DEFCLXQ(CHANNEL)
   ```

2. Isolate any message flows that must not share a cluster transmission queue with other flows.
   - See "Clustering: Example configuration of multiple cluster transmission queues" on page 286. In the example the SALES queue, which must be isolated, is a member of the SALES cluster, on SALESRV.

To isolate the SALES queue, create a new cluster `Q.SALES`, make the SALESRV queue manager a member, and modify the SALES queue to belong to `Q.SALES`.

- Queue managers that send messages to SALES must also be members of the new cluster. If you use a clustered queue alias and a gateway queue manager, as in the example, in many cases you can limit the changes to making the gateway queue manager a member of the new cluster.
- However, separating flows from the gateway to the destination does not separate flows to the gateway from the source queue manager. But it sometimes turns out to be sufficient to separate flows from the gateway and not flows to the gateway. If it is not sufficient, then add the source queue manager into the new cluster. If you want messages to travel through the gateway, move the cluster alias to the new cluster and continue to send messages to the cluster alias on the gateway, and not directly to the target queue manager.

Follow these steps to isolate message flows:

a) Configure the destinations of the flows so that each target queue is the only queue in a specific cluster, on that queue manager.

b) Create the cluster-sender and cluster-receiver channels for any new clusters you have created following a systematic naming convention.

- See “Clustering: Special considerations for overlapping clusters” on page 273.

c) Define a cluster transmission queue for each isolated destination on every queue manager that sends messages to the target queue.

- A naming convention for cluster transmission queues is to use the value of the cluster channel name attribute, `CLCHNAME`, prefixed with `XMITQ.`

3. Create cluster transmission queues to meet governance or monitoring requirements.

- Typical governance and monitoring requirements result in a transmission queue per cluster or a transmission queue per queue manager. If you follow the naming convention for cluster channels, *`ClusterName.QueueManagerName`*, it is easy to create generic channel names that select a cluster of queue managers, or all the clusters a queue manager is a member of; see “Clustering: Example configuration of multiple cluster transmission queues” on page 286.
- Extend the naming convention for cluster transmission queues to cater for generic channel names, by replacing the asterisk symbol by a percent sign. For example,

```
DEFINE QLOCAL(XMITQ.SALES.%) USAGE(XMITQ) CLCHNAME(SALES.*)
```

Related concepts

“Cluster transmission queues and cluster-sender channels” on page 166
Messages between clustered queue managers are stored on cluster transmission queues and forwarded by cluster-sender channels.

“Clustering: Application isolation using multiple cluster transmission queues” on page 277
You can isolate the message flows between queue managers in a cluster. You can place messages being transported by different cluster-sender channels onto different cluster transmission queues. You can use the approach in a single cluster or with overlapping clusters. The topic provides examples and some best practices to guide you in choosing an approach to use.

“Access control and multiple cluster transmission queues” on page 156
Choose between three modes of checking when an application puts messages to remote cluster queues. The modes are checking remotely against the cluster queue, checking locally against `SYSTEM.CLUSTER.TRANSMIT.QUEUE`, or checking against local profiles for the cluster queue, or cluster queue manager.

“Clustering: Special considerations for overlapping clusters” on page 273
This topic provides guidance for planning and administering IBM WebSphere MQ clusters. This information is a guide based on testing and feedback from customers.

“Overlapping clusters” on page 175

Overlapping clusters provide additional administrative capabilities. Use namelists to reduce the number of commands needed to administer overlapping clusters.

Related tasks

"Adding a remote queue definition to isolate messages sent from a gateway queue manager" on page 195
Modify the configuration of overlapping clusters that use a gateway queue manager. After the modification messages are transferred to an application from the gateway queue manager without using the same transmission queue or channels as other cluster messages. The solution uses a clustered queue remote definition, and a separate sender channel and transmission queue.

"Adding a cluster transmit queue to isolate cluster message traffic sent from a gateway queue manager" on page 198
Modify the configuration of overlapping clusters that use a gateway queue manager. After the modification messages are transferred to an application from the gateway queue manager without using the same transmission queue or channels as other cluster messages. The solution uses an additional cluster transmission queue to separate message traffic to a single queue manager in a cluster.

"Adding a cluster and a cluster transmit queue to isolate cluster message traffic sent from a gateway queue manager" on page 201
Modify the configuration of overlapping clusters that use a gateway queue manager. After the modification messages are transferred to an application from the gateway queue manager without using the same transmission queue or channels as other cluster messages. The solution uses an additional cluster to isolate the messages to a particular cluster queue.

"Changing the default to separate cluster transmission queues to isolate message traffic" on page 218
You can change the default way a queue manager stores messages for a clustered queue or topic on a transmission queue. Changing the default provides you with a way to isolate cluster messages on a gateway queue manager.

"Creating two-overlapping clusters with a gateway queue manager" on page 212
Follow the instructions in the task to construct overlapping clusters with a gateway queue manager. Use the clusters as a starting point for the following examples of isolating messages to one application from messages to other applications in a cluster.

"Configuring message paths between clusters" on page 252
Connect clusters together using a gateway queue manager. Make queues or queue managers visible to all the clusters by defining cluster queue or cluster queue manager aliases on the gateway queue manager.

How to choose what type of cluster transmission queue to use

How to choose between different cluster transmission queue configuration options.

From Version 7.5 onwards, you can choose which cluster transmission queue is associated with a cluster-sender channel.

1. You can have all cluster-sender channels associated with the single default cluster transmit queue, `SYSTEM.CLUSTER.TRANSMIT.QUEUE`. This option is the default, and is the only choice for queue managers that run version Version 7.1, or earlier.
2. You can set all cluster-sender channels to be automatically associated with a separate cluster transmission queue. The queues are created by the queue manager from the model queue `SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE` and named `SYSTEM.CLUSTER.TRANSMIT.`*`ChannelName`*. Channels will use their uniquely named cluster transmit queue if the queue manager attribute **DEFCLXQ** is set to `CHANNEL`.

   Attention: If you are using dedicated `SYSTEM.CLUSTER.TRANSMIT.QUEUES` with a queue manager that was upgraded from an earlier version of the product, ensure that the `SYSTEM.CLUSTER.TRANSMIT.MODEL.QUEUE` has the SHARE/NOSHARE option set to **SHARE**.
3. You can set specific cluster-sender channels to be served by a single cluster transmission queue. Select this option by creating a transmission queue and setting its **CLCHNAME** attribute to the name of the cluster-sender channel.
4. You can select groups of cluster-sender channels to be served by a single cluster transmission queue. Select this option by creating a transmission queue and setting its **CLCHNAME** attribute to a generic

channel name, such as *ClusterName*.*. If you name cluster channels by following the naming conventions in "Clustering: Special considerations for overlapping clusters" on page 273, this name selects all cluster channels connected to queue managers in the cluster *ClusterName*.

You can combine either of the default cluster transmission queue options for some cluster-sender channels, with any number of specific and generic cluster transmission queue configurations.

Best practices

In most cases, for existing IBM WebSphere MQ installations, the default configuration is the best choice. A cluster queue manager stores cluster messages on a single cluster transmission queue, SYSTEM.CLUSTER.TRANSMIT.QUEUE. You have the choice of changing the default to storing messages for different queue managers and different clusters on separate transmission queues, or of defining your own transmission queues.

In most cases, for new IBM WebSphere MQ installations, the default configuration is also the best choice. The process of switching from the default configuration to the alternative default of having one transmission queue for each cluster-sender channel is automatic. Switching back is also automatic. The choice of one or the other is not critical, you can reverse it.

The reason for choosing a different configuration is more to do with governance, and management, than with functionality or performance. With a couple of exceptions, configuring multiple cluster transmission queues does not benefit the behavior of the queue manager. It results in more queues, and requires you to modify monitoring and management procedures you have already set up that refer to the single transmission queue. That is why, on balance, remaining with the default configuration is the best choice, unless you have strong governance or management reasons for a different choice.

The exceptions are both concerned with what happens if the number of messages stored on SYSTEM.CLUSTER.TRANSMIT.QUEUE increases. If you take every step to separate the messages for one destination from the messages for another destination, then channel and delivery problems with one destination ought not to affect the delivery to another destination. However, the number of messages stored on SYSTEM.CLUSTER.TRANSMIT.QUEUE can increase due to not delivering messages fast enough to one destination. The number of messages on SYSTEM.CLUSTER.TRANSMIT.QUEUE for one destination can affect the delivery of messages to other destinations.

To avoid problems that result from filling up a single transmission queue, aim to build sufficient capacity into your configuration. Then, if a destination fails and a message backlog starts to build up, you have time to fix the problem.

If messages are routed through a hub queue manager, such as a cluster gateway, they share a common transmission queue, SYSTEM.CLUSTER.TRANSMIT.QUEUE. If the number of messages stored on SYSTEM.CLUSTER.TRANSMIT.QUEUE on the gateway queue manager reaches its maximum depth, the queue manager starts to reject new messages for the transmission queue until its depth reduces. The congestion affects messages for all destinations that are routed through the gateway. Messages back up the transmission queues of other queue managers that are sending messages to the gateway. The problem manifests itself in messages written to queue manager error logs, falling message throughput, and longer elapsed times between sending a message and the time that a message arrives at its destination.

The effect of congestion on a single transmission queue can become apparent, even before it is full. If you have a mixed message traffic, with some large non-persistent messages and some small messages, the time to deliver small messages increases as the transmission queue fills. The delay is due to writing large non-persistent messages to disk that would not normally get written to disk. If you have time critical message flows, sharing a cluster transmission queue with other mixed messages flows, it could be worth configuring a special message path to isolate it from other message flows; see "Adding a cluster and a cluster transmit queue to isolate cluster message traffic sent from a gateway queue manager" on page 201

The other reasons for configuring separate cluster transmission queues are to meet governance requirements, or to simplify monitoring messages that are sent to different cluster destinations. For example, you might have to demonstrate that messages for one destination never share a transmission queue with messages for another destination.

Change the queue manager attribute **DEFCLXQ** that controls the default cluster transmission queue, to create different cluster transmission queues for every cluster-sender channel. Multiple destinations can share a cluster-sender channel, so you must plan your clusters to meet this objective fully. Apply the method "Adding a cluster and a cluster transmit queue to isolate cluster message traffic sent from a gateway queue manager" on page 201 systematically to all your cluster queues. The result you are aiming for is for no cluster destination to share a cluster-sender channel with another cluster destination. As a consequence, no message for a cluster destination shares its cluster transmission queue with a message for another destination.

Creating a separate cluster transmission queue for some specific message flow, makes it easy to monitor the flow of messages to that destination. To use a new cluster transmission queue, define the queue, associate it with a cluster-sender channel, and stop and start the channel. The change does not have to be permanent. You could isolate a message flow for a while, to monitor the transmission queue, and then revert to using the default transmission queue again.

Related tasks

Clustering: Example configuration of multiple cluster transmission queues
In this task you apply the steps to plan multiple cluster transmission queues to three overlapping clusters. The requirements are to separate messages flows to one cluster queue, from all other message flows, and to store messages for different clusters on different cluster transmission queues.

Clustering: Switching cluster transmission queues
Plan how the changes to the cluster transmission queues of an existing production queue manager are going to be brought into effect.

Clustering: Example configuration of multiple cluster transmission queues

In this task you apply the steps to plan multiple cluster transmission queues to three overlapping clusters. The requirements are to separate messages flows to one cluster queue, from all other message flows, and to store messages for different clusters on different cluster transmission queues.

About this task

The steps in this task show how to apply the procedure in "Clustering: Planning how to configure cluster transmission queues" on page 281 and arrive at the configuration shown in Figure 57 on page 287. It is an example of three overlapping clusters, with a gateway queue manager, that is configured with separate cluster transmission queues. The MQSC commands to define the clusters are described in "Creating the example clusters" on page 289.

For the example, there are two requirements. One is to separate the message flow from the gateway queue manager to the sales application that logs sales. The second is to query how many messages are waiting to be sent to different departmental areas at any point in time. The SALES, FINANCE, and DEVELOP clusters are already defined. Cluster messages are currently forwarded from SYSTEM.CLUSTER.TRANSMIT.QUEUE.

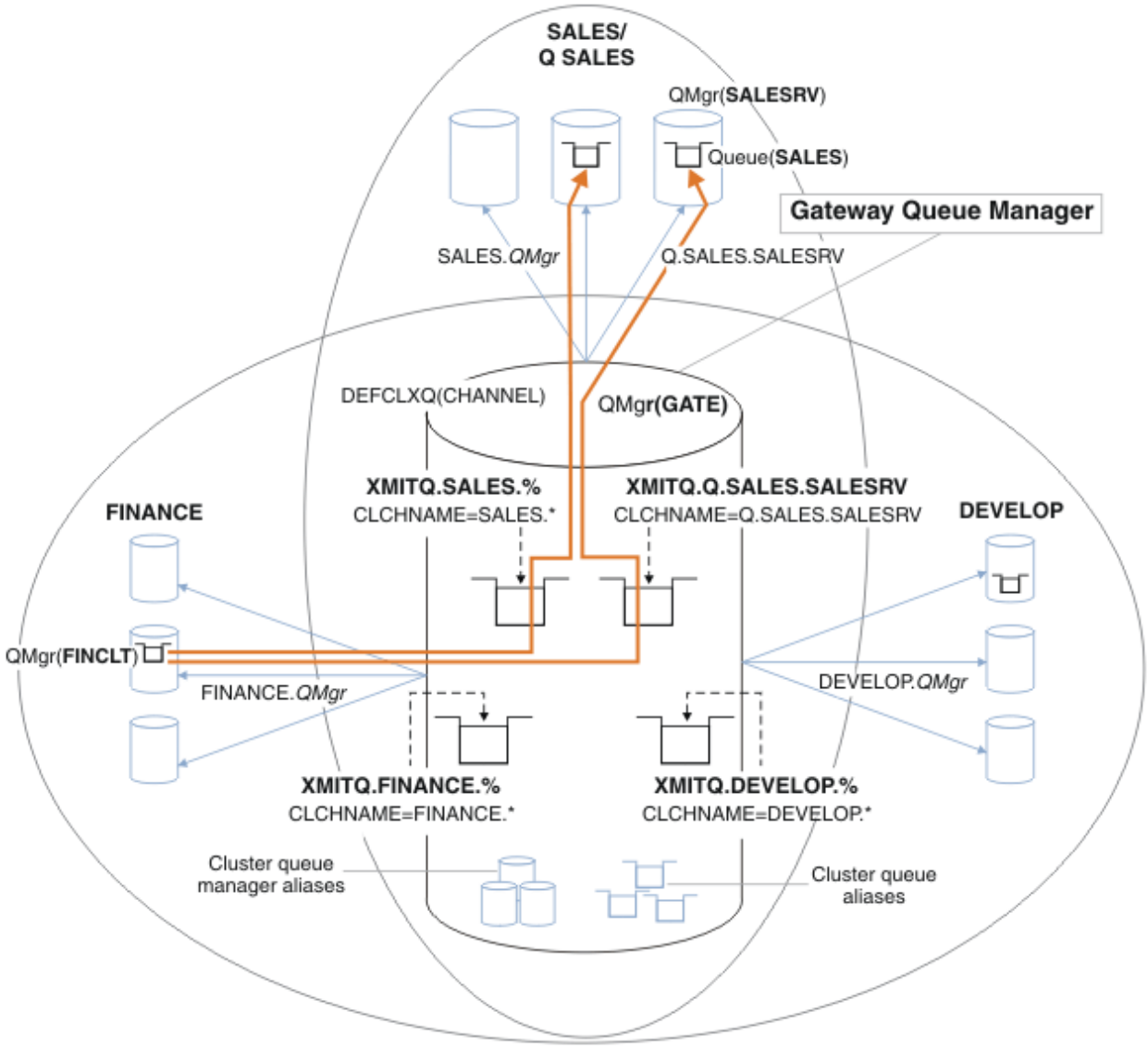


Figure 57. Example of specific transmission queues for different departmental IBM WebSphere MQ clusters

The steps to modify the clusters are as follows; see Changes to isolate the sales queue in a new cluster and separate the gateway cluster transmission queues for the definitions.

Procedure

1. The first configuration step is to " Select the type of default cluster transmission queue to use".

   The decision is to create separate default cluster transmission queues by running the following **MQSC** command on the GATE queue manager.

   ```
   ALTER QMGR DEFCLXQ(CHANNEL)
   ```

   There is no strong reason for choosing this default, as the intention is to manually define cluster transmission queues. The choice does have a weak diagnostic value. If a manual definition is done wrongly, and a message flows down a default cluster transmission queue, it shows up in the creation of a permanent-dynamic cluster transmission queue.
2. The second configuration step is to " Isolate any message flows that must not share a cluster transmission queue with other flows".

In this case the sales application that receives messages from the queue SALES on SALESRV requires isolation. Only isolation of messages from the gateway queue manager is required. The three sub-steps are:

a) "Configure the destinations of the flows so that each target queue is the only queue in a specific cluster, on that queue manager".

The example requires adding the queue manager SALESRV to a new cluster within the sales department. If you have few queues that require isolation, you might decide on creating a specific cluster for the SALES queue. A possible naming convention for the cluster name is to name such clusters, `Q.` *`QueueName`*, for example `Q.SALES` . An alternative approach, which might be more practical if you have a large number of queues to be isolated, is to create clusters of isolated queues where and when needed. The clusters names might be `QUEUES.`*`n`*.

In the example, the new cluster is called `Q.SALES`. To add the new cluster, see the definitions in Changes to isolate the sales queue in a new cluster and separate the gateway cluster transmission queues. The summary of definition changes is as follows:

i) Add `Q.SALES` to the namelist of clusters on the repository queue managers. The namelist is referred to in the queue manager **REPOSNL** parameter.

ii) Add `Q.SALES` to the namelist of clusters on the gateway queue manager. The namelist is referred to in all the cluster queue alias and cluster queue manager alias definitions on the gateway queue manager.

iii) Create a namelist on the queue manager SALESRV , for both the clusters it is a member of, and change the cluster membership of the SALES queue:

```
DEFINE NAMELIST(CLUSTERS) NAMES(SALES, Q.SALES) REPLACE
ALTER QLOCAL(SALES) CLUSTER(' ') CLUSNL(SALESRV.CLUSTERS)
```

The SALES queue is a member of both clusters, just for the transition. Once the new configuration is running, you remove the SALES queue from the SALES cluster; see Figure 58 on page 292.

b) "Create the cluster-sender and cluster-receiver channels for any new clusters you have created following a systematic naming convention".

i) Add the cluster-receiver channel `Q.SALES.`*`RepositoryQMgr`* to each of the repository queue managers

ii) Add the cluster-sender channel `Q.SALES.`*`OtherRepositoryQMgr`* to each of the repository queue managers, to connect to the other repository manager. Start these channels.

iii) Add the cluster receiver channels `Q.SALES.SALESRV` , and `Q.SALES.GATE` to either of the repository queue managers that is running.

iv) Add the cluster-sender channels `Q.SALES.SALESRV` , and `Q.SALES.GATE` to the SALESRV and GATE queue managers. Connect the cluster-sender channel to the repository queue manager that you created the cluster-receiver channels on.

c) "Define a cluster transmission queue for each isolated destination on every queue manager that sends messages to the target queue".

On the gateway queue manager define the cluster transmission queue `XMITQ.Q.SALES.SALESRV` for the `Q.SALES.SALESRV` cluster-sender channel:

```
DEFINE QLOCAL(XMITQ.Q.SALES.SALESRV) USAGE(XMITQ) CLCHNAME(Q.SALES.SALESRV) REPLACE
```

3. The third configuration step is to " Create cluster transmission queues to meet governance or monitoring requirements".

On the gateway queue manager define the cluster transmission queues:

```
DEFINE QLOCAL(XMITQ.SALES)   USAGE(XMITQ) CLCHNAME(SALES.*)   REPLACE
DEFINE QLOCAL(XMITQ.DEVELOP) USAGE(XMITQ) CLCHNAME(DEVELOP.*) REPLACE
DEFINE QLOCAL(XMITQ.FINANCE) USAGE(XMITQ) FINANCE(SALES.*)    REPLACE
```

What to do next

Switch to the new configuration on the gateway queue manager.

The switch is triggered by starting the new channels, and restarting the channels that are now associated with different transmission queues. Alternatively, you can stop and start the gateway queue manager.

1. Stop the following channels on the gateway queue manager:

```
SALES.Qmgr
DEVELOP.Qmgr
FINANCE.Qmgr
```

2. Start the following channels on the gateway queue manager:

```
SALES.Qmgr
DEVELOP.Qmgr
FINANCE.Qmgr
Q.SALES.SAVESRV
```

When the switch is complete, remove the SALES queue from the SALES cluster; see Figure 58 on page 292.

Related concepts

How to choose what type of cluster transmission queue to use
How to choose between different cluster transmission queue configuration options.

Related tasks

Clustering: Switching cluster transmission queues
Plan how the changes to the cluster transmission queues of an existing production queue manager are going to be brought into effect.

Creating the example clusters
The definitions and instructions to create the example cluster, and modify it to isolate the SALES queue and separate messages on the gateway queue manager.

About this task

The full **MQSC** commands to create the FINANCE, SALES, and Q.SALES clusters are provided in Definitions for the basic clusters, Changes to isolate the sales queue in a new cluster and separate the gateway cluster transmission queues, and Remove the sales queue on queue manager SALESRV from the sales cluster. The DEVELOP cluster is omitted from the definitions, to keep the definitions shorter.

Procedure

1. Create the SALES and FINANCE clusters, and the gateway queue manager.
   a) Create the queue managers.

   Run the command: `crtmqm -sax -u SYSTEM.DEAD.LETTER.QUEUE` *QmgrName* for each of the queue manager names in Table 27 on page 289.

Table 27. Queue manager names and port numbers

Description	Queue Manager Name	Port number
Finance repository	FINR1	1414
Finance repository	FINR2	1415
Finance client	FINCLT	1418
Sales repository	SALER1	1416

Table 27. Queue manager names and port numbers (continued)

Description	Queue Manager Name	Port number
Sales repository	SALER2	1417
Sales server	SALESRV	1419
Gateway	GATE	1420

b) Start all the queue managers

Run the command: `strmqm` *QmgrName* for each of the queue manager names in Table 27 on page 289.

c) Create the definitions for each of the queue managers

Run the command: `runmqsc` *QmgrName* < *filename* where the files are listed in Definitions for the basic clusters, and the file name matches the queue manager name.

Definitions for the basic clusters

finr1.txt

```
DEFINE LISTENER(1414) TRPTYPE(TCP) IPADDR(localhost) CONTROL(QMGR) PORT(1414) REPLACE
START LISTENER(1414)
ALTER QMGR REPOS(FINANCE)
DEFINE CHANNEL(FINANCE.FINR2) CHLTYPE(CLUSSDR)  CONNAME('localhost(1415)')
CLUSTER(FINANCE) REPLACE
DEFINE CHANNEL(FINANCE.FINR1) CHLTYPE(CLUSRCVR) CONNAME('localhost(1414)')
CLUSTER(FINANCE) REPLACE
```

finr2.txt

```
DEFINE LISTENER(1415) TRPTYPE(TCP) IPADDR(localhost) CONTROL(QMGR) PORT(1415) REPLACE
START LISTENER(1415)
ALTER QMGR REPOS(FINANCE)
DEFINE CHANNEL(FINANCE.FINR1) CHLTYPE(CLUSSDR)  CONNAME('localhost(1414)')
CLUSTER(FINANCE) REPLACE
DEFINE CHANNEL(FINANCE.FINR2) CHLTYPE(CLUSRCVR) CONNAME('localhost(1415)')
CLUSTER(FINANCE) REPLACE
```

finclt.txt

```
DEFINE LISTENER(1418) TRPTYPE(TCP) IPADDR(localhost) CONTROL(QMGR) PORT(1418) REPLACE
START LISTENER(1418)
DEFINE CHANNEL(FINANCE.FINR1) CHLTYPE(CLUSSDR)   CONNAME('localhost(1414)')
CLUSTER(FINANCE) REPLACE
DEFINE CHANNEL(FINANCE.FINCLT) CHLTYPE(CLUSRCVR) CONNAME('localhost(1418)')
CLUSTER(FINANCE) REPLACE
DEFINE QMODEL(SYSTEM.SAMPLE.REPLY) REPLACE
```

saler1.txt

```
DEFINE LISTENER(1416) TRPTYPE(TCP) IPADDR(localhost) CONTROL(QMGR) PORT(1416) REPLACE
START LISTENER(1416)
ALTER QMGR REPOS(SALES)
DEFINE CHANNEL(SALES.SALER2) CHLTYPE(CLUSSDR)  CONNAME('localhost(1417)')
CLUSTER(SALES) REPLACE
DEFINE CHANNEL(SALES.SALER1) CHLTYPE(CLUSRCVR) CONNAME('localhost(1416)')
CLUSTER(SALES) REPLACE
```

saler2.txt

```
DEFINE LISTENER(1417) TRPTYPE(TCP) IPADDR(localhost) CONTROL(QMGR) PORT(1417) REPLACE
START LISTENER(1417)
ALTER QMGR REPOS(SALES)
DEFINE CHANNEL(SALES.SALER1) CHLTYPE(CLUSSDR)  CONNAME('localhost(1416)')
CLUSTER(SALES) REPLACE
DEFINE CHANNEL(SALES.SALER2) CHLTYPE(CLUSRCVR) CONNAME('localhost(1417)')
CLUSTER(SALES) REPLACE
```

salesrv.txt

```
DEFINE LISTENER(1419) TRPTYPE(TCP) IPADDR(localhost) CONTROL(QMGR) PORT(1419) REPLACE
START LISTENER(1419)
DEFINE CHANNEL(SALES.SALER1)  CHLTYPE(CLUSSDR)  CONNAME('localhost(1416)')
CLUSTER(SALES) REPLACE
DEFINE CHANNEL(SALES.SALESRV) CHLTYPE(CLUSRCVR) CONNAME('localhost(1419)')
CLUSTER(SALES) REPLACE
DEFINE QLOCAL(SALES) CLUSTER(SALES) TRIGGER INITQ(SYSTEM.DEFAULT.INITIATION.QUEUE)
PROCESS(ECHO) REPLACE
DEFINE PROCESS(ECHO) APPLICID(AMQSECH) REPLACE
```

gate.txt

```
DEFINE LISTENER(1420) TRPTYPE(TCP) IPADDR(LOCALHOST) CONTROL(QMGR) PORT(1420) REPLACE
START LISTENER(1420)
DEFINE NAMELIST(ALL) NAMES(SALES, FINANCE)
DEFINE CHANNEL(FINANCE.FINR1) CHLTYPE(CLUSSDR)  CONNAME('LOCALHOST(1414)')
CLUSTER(FINANCE) REPLACE
DEFINE CHANNEL(FINANCE.GATE)  CHLTYPE(CLUSRCVR) CONNAME('LOCALHOST(1420)')
CLUSTER(FINANCE) REPLACE
DEFINE CHANNEL(SALES.SALER1)  CHLTYPE(CLUSSDR)  CONNAME('LOCALHOST(1416)')
CLUSTER(SALES) REPLACE
DEFINE CHANNEL(SALES.GATE)    CHLTYPE(CLUSRCVR) CONNAME('LOCALHOST(1420)')
CLUSTER(SALES) REPLACE
DEFINE QALIAS(A.SALES) CLUSNL(ALL) TARGET(SALES) TARGTYPE(QUEUE) DEFBIND(NOTFIXED)
REPLACE
DEFINE QREMOTE(FINCLT)  RNAME(' ') RQMNAME(FINCLT)  CLUSNL(ALL) REPLACE
DEFINE QREMOTE(SALESRV) RNAME(' ') RQMNAME(SALESRV) CLUSNL(ALL) REPLACE
```

2. Test the configuration by running the sample request program.
   a) Start the trigger monitor program on the SALESRV queue manager

      On Windows, open a command window and run the command `runmqtrm -m SALESRV`

   b) Run the sample request program, and send a request.

      On Windows, open a command window and run the command `amqsreq A.SALES FINCLT`

      The request message is echoed back, and after 15 seconds the sample program finishes.
3. Create the definitions to isolate the SALES queue in the `Q.SALES` cluster and separate cluster messages for the SALES and FINANCE cluster on the gateway queue manager.

   Run the command: `runmqsc` *QmgrName* < *filename* where the files are listed in the following list, and the file name almost matches the queue manager name.

   Changes to isolate the sales queue in a new cluster and separate the gateway cluster transmission queues
   chgsaler1.txt

```
DEFINE NAMELIST(CLUSTERS) NAMES(SALES, Q.SALES)
ALTER  QMGR REPOS(' ') REPOSNL(CLUSTERS)
DEFINE CHANNEL(Q.SALES.SALER2) CHLTYPE(CLUSSDR)  CONNAME('localhost(1417)')
CLUSTER(Q.SALES) REPLACE
DEFINE CHANNEL(Q.SALES.SALER1) CHLTYPE(CLUSRCVR) CONNAME('localhost(1416)')
CLUSTER(Q.SALES) REPLACE
```

   chgsaler2.txt

```
DEFINE NAMELIST(CLUSTERS) NAMES(SALES, Q.SALES)
ALTER  QMGR REPOS(' ') REPOSNL(CLUSTERS)
DEFINE CHANNEL(Q.SALES.SALER1) CHLTYPE(CLUSSDR)  CONNAME('localhost(1416)')
CLUSTER(Q.SALES) REPLACE
DEFINE CHANNEL(Q.SALES.SALER2) CHLTYPE(CLUSRCVR) CONNAME('localhost(1417)')
CLUSTER(Q.SALES) REPLACE
```

   chgsalesrv.txt

```
DEFINE NAMELIST (CLUSTERS) NAMES(SALES, Q.SALES)
DEFINE CHANNEL(Q.SALES.SALER1)  CHLTYPE(CLUSSDR)  CONNAME('localhost(1416)')
CLUSTER(Q.SALES) REPLACE
DEFINE CHANNEL(Q.SALES.SAVESRV) CHLTYPE(CLUSRCVR) CONNAME('localhost(1419)')
```

```
CLUSTER(Q.SALES) REPLACE
ALTER  QLOCAL (SALES) CLUSTER(' ') CLUSNL(CLUSTERS)
```

chggate.txt

```
ALTER  NAMELIST(ALL) NAMES(SALES, FINANCE, Q.SALES)
ALTER  QMGR DEFCLXQ(CHANNEL)
DEFINE CHANNEL(Q.SALES.SALER1)  CHLTYPE(CLUSSDR)   CONNAME('localhost(1416)')
CLUSTER(Q.SALES) REPLACE
DEFINE CHANNEL(Q.SALES.GATE)    CHLTYPE(CLUSRCVR)  CONNAME('localhost(1420)')
CLUSTER(Q.SALES) REPLACE
DEFINE QLOCAL (XMITQ.Q.SALES.SALESRV) USAGE(XMITQ) CLCHNAME(Q.SALES.SALESRV) REPLACE
DEFINE QLOCAL (XMITQ.SALES)         USAGE(XMITQ) CLCHNAME(SALES.*)   REPLACE
DEFINE QLOCAL (XMITQ.FINANCE)       USAGE(XMITQ) CLCHNAME(FINANCE.*) REPLACE
```

4. Remove the SALES queue from the SALES cluster.

   Run the **MQSC** command in Figure 58 on page 292:

   ```
   ALTER QLOCAL(SALES) CLUSTER('Q.SALES') CLUSNL(' ')
   ```

   Figure 58. Remove the sales queue on queue manager SALESRV from the sales cluster

5. Switch the channels to the new transmission queues.

   The requirement is to stop and start all the channels that the GATE queue manager is using. To do this with the least number of commands, stop and start the queue manager

   ```
   endmqm -i GATE
   strmqm GATE
   ```

What to do next

1. Rerun the sample request program to verify the new configuration works; see step "2" on page 291
2. Monitor the messages flowing through all the cluster transmission queues on the GATE queue manager:
   a. Alter the definition of each of the cluster transmission queues to turn queue monitoring on.

      ```
      ALTER QLOCAL(SYSTEM.CLUSTER.TRANSMIT.
      name) STATQ(ON)
      ```

   b. Check queue manager statistics monitoring is OFF , to minimize output, and set the monitoring interval to a lower value to perform multiple tests conveniently.

      ```
      ALTER QMGR STATINT(60) STATCHL(OFF) STATQ(OFF) STATMQI(OFF) STATACLS(OFF)
      ```

   c. Restart the GATE queue manager.
   d. Run the sample request program a few times to verify that an equal number of messages are flowing through SYSTEM.CLUSTER.TRANSMIT.Q.SALES.SALESRV and SYSTEM.CLUSTER.TRANSMIT.QUEUE. Requests flow through SYSTEM.CLUSTER.TRANSMIT.Q.SALES.SALESRV and replies through SYSTEM.CLUSTER.TRANSMIT.QUEUE.

      ```
      amqsmon -m GATE -t statistics
      ```

   e. The results over a couple of intervals are as follows:

      ```
      C:\Documents and Settings\Admin>amqsmon -m GATE -t statistics
      MonitoringType: QueueStatistics
      QueueManager: 'GATE'
      IntervalStartDate: '2012-02-27'
      IntervalStartTime: '14.59.20'
      IntervalEndDate: '2012-02-27'
      ```

```
IntervalEndTime: '15.00.20'
CommandLevel: 700
ObjectCount: 2
QueueStatistics: 0
  QueueName: 'SYSTEM.CLUSTER.TRANSMIT.QUEUE'
  CreateDate: '2012-02-24'
  CreateTime: '15.58.15'
  ...
  Put1Count: [0, 0]
  Put1FailCount: 0
  PutBytes: [435, 0]
  GetCount: [1, 0]
  GetBytes: [435, 0]
  ...
QueueStatistics: 1
  QueueName: 'SYSTEM.CLUSTER.TRANSMIT.Q.SALES.SAVESRV'
  CreateDate: '2012-02-24'
  CreateTime: '16.37.43'
  ...
  PutCount: [1, 0]
  PutFailCount: 0
  Put1Count: [0, 0]
  Put1FailCount: 0
  PutBytes: [435, 0]
  GetCount: [1, 0]
  GetBytes: [435, 0]
  ...
MonitoringType: QueueStatistics
QueueManager: 'GATE'
IntervalStartDate: '2012-02-27'
IntervalStartTime: '15.00.20'
IntervalEndDate: '2012-02-27'
IntervalEndTime: '15.01.20'
CommandLevel: 700
ObjectCount: 2
QueueStatistics: 0
  QueueName: 'SYSTEM.CLUSTER.TRANSMIT.QUEUE'
  CreateDate: '2012-02-24'
  CreateTime: '15.58.15'
  ...
  PutCount: [2, 0]
  PutFailCount: 0
  Put1Count: [0, 0]
  Put1FailCount: 0
  PutBytes: [863, 0]
  GetCount: [2, 0]
  GetBytes: [863, 0]
  ...
QueueStatistics: 1
  QueueName: 'SYSTEM.CLUSTER.TRANSMIT.Q.SALES.SAVESRV'
  CreateDate: '2012-02-24'
  CreateTime: '16.37.43'
  ...
  PutCount: [2, 0]
  PutFailCount: 0
  Put1Count: [0, 0]
  Put1FailCount: 0
  PutBytes: [863, 0]
```

```
  GetCount: [2, 0]
  GetBytes: [863, 0]
  ...
2 Records Processed.
```

One request and reply message were sent in the first interval and two in the second. You can infer that the request messages were placed on `SYSTEM.CLUSTER.TRANSMIT.Q.SALES.SAVESRV`, and the reply messages on `SYSTEM.CLUSTER.TRANSMIT.QUEUE`.

Clustering: Switching cluster transmission queues

Plan how the changes to the cluster transmission queues of an existing production queue manager are going to be brought into effect.

Before you begin

If you reduce the number of messages the switching process has to transfer to the new transmission queue, switching completes more quickly. Read "How the process to switch cluster-sender channel to a different transmission queue works" on page 169 for the reasons for trying to empty the transmission queue before proceeding any further.

About this task

You have a choice of two ways of making the changes to cluster transmission queues take effect.

1. Let the queue manager make the changes automatically. This is the default. The queue manager switches cluster-sender channels with pending transmission queue changes when a cluster-sender channel next starts.
2. Make the changes manually. You can make the changes to a cluster-sender channel when it is stopped. You can switch it from one cluster transmission queue to another before the cluster-sender channel starts.

What factors do you take into account when deciding which of the two options to choose, and how do you manage the switch?

Procedure

- Option 1: Let the queue manager make the changes automatically; see "Switching active cluster-sender channels to another set of cluster-transmission queues" on page 295.

  Choose this option if you want the queue manager to make the switch for you.

  An alternative way to describe this option is to say the queue manager switches a cluster-sender channel without you forcing the channel to stop. You do have the option of forcing the channel to stop, and then starting the channel, to make the switch happen sooner. The switch starts when the channel starts, and runs while the channel is running, which is different to option 2. In option 2, the switch takes place when the channel is stopped.

  If you choose this option by letting the switch happen automatically, the switching process starts when a cluster-sender channel starts. If the channel is not stopped, it starts after it becomes inactive, if there is a message to process. If the channel is stopped, start it with the START CHANNEL command.

  The switch process completes as soon as there are no messages left for the cluster-sender channel on the transmission queue the channel was serving. As soon as that is the case, newly arrived messages for the cluster-sender channel are stored directly on the new transmission queue. Until then, messages are stored on the old transmission queue, and the switching process transfers messages from the old transmission queue to the new transmission queue. The cluster-sender channel forwards messages from the new cluster transmission queue during the whole switching process.

  When the switch process completes depends on the state of the system. If you are making the changes in a maintenance window, assess beforehand whether the switching process will complete

in time. Whether it will complete in time depends on whether the number of messages that are awaiting transfer from the old transmission queue reaches zero.

The advantage of the first method is it is automatic. A disadvantage is that if the time to make the configuration changes is limited to a maintenance window, you must be confident that you can control the system to complete the switch process inside the maintenance window. If you cannot be sure, option 2 might be a better choice.

- Option 2: Make the changes manually; see "Switching a stopped cluster-sender channel to another cluster transmission queue" on page 297.

  Choose this option if you want to control the entire switching process manually, or if you want to switch a stopped or inactive channel. It is a good choice, if you are switching a few cluster-sender channels, and you want to do the switch during a maintenance window.

  An alternative description of this option is to say that you switch the cluster-sender channel, while the cluster-sender channel is stopped.

  If you choose this option you have complete control over when the switch takes place.
  You can be certain about completing the switching process in a fixed amount of time, within a maintenance window. The time the switch takes depends on how many messages have to be transferred from one transmission queue to the other. If messages keep arriving, it might take a time for the process to transfer all the messages.
  You have the option of switching the channel without transferring messages from the old transmission queue. The switch is "instant".
  When you restart the cluster-sender channel, it starts processing messages on the transmission queue you newly assigned to it.

  The advantage of the second method is you have control over the switching process. The disadvantage is that you must identify the cluster-sender channels to be switched, run the necessary commands, and resolve any in-doubt channels that might be preventing the cluster-sender channel stopping.

Related concepts

How to choose what type of cluster transmission queue to use
How to choose between different cluster transmission queue configuration options.

Related tasks

Clustering: Example configuration of multiple cluster transmission queues
In this task you apply the steps to plan multiple cluster transmission queues to three overlapping clusters. The requirements are to separate messages flows to one cluster queue, from all other message flows, and to store messages for different clusters on different cluster transmission queues.

"Switching active cluster-sender channels to another set of cluster-transmission queues" on page 295
This task gives you three options for switching active cluster-sender channels. One option is to let the queue manager make the switch automatically, which does not affect running applications. The other options are to stop and start channels manually, or to restart the queue manager.

"Switching a stopped cluster-sender channel to another cluster transmission queue" on page 297

Related information

"How the process to switch cluster-sender channel to a different transmission queue works" on page 169

Switching active cluster-sender channels to another set of cluster-transmission queues
This task gives you three options for switching active cluster-sender channels. One option is to let the queue manager make the switch automatically, which does not affect running applications. The other options are to stop and start channels manually, or to restart the queue manager.

Before you begin

Change the cluster transmission queue configuration. You can change the **DEFCLXQ** queue manager attribute, or add or modify the **CLCHNAME** attribute of transmission queues.

If you reduce the number of messages the switching process has to transfer to the new transmission queue, switching completes more quickly. Read "How the process to switch cluster-sender channel to a different transmission queue works" on page 169 for the reasons for trying to empty the transmission queue before proceeding any further.

About this task

Use the steps in the task as a basis for working out your own plan for making cluster-transmission queue configuration changes.

Procedure

1. Optional: Record the current channel status

   Make a record of the status of current and saved channels that are serving cluster transmission queues. The following commands display the status associated with system cluster transmission queues. Add your own commands to display the status associated with cluster-transmission queues that you have defined. Use a convention, such as XMITQ.*ChannelName*, to name cluster transmission queues that you define to make it easy to display the channel status for those transmission queues.

   ```
   DISPLAY CHSTATUS(*) WHERE(XMITQ LK 'SYSTEM.CLUSTER.TRANSMIT.*')
   DISPLAY CHSTATUS(*) SAVED WHERE(XMITQ LK 'SYSTEM.CLUSTER.TRANSMIT.*')
   ```

2. Switch transmission queues.

   - Do nothing. The queue manager switches cluster-sender channels when they restart after being stopped or inactive.

     Choose this option if you have no rules or concerns about altering a queue manager configuration. Running applications are not affected by the changes.
   - Restart the queue manager. All cluster-sender channels are stopped and restarted automatically on demand.

     Choose this option to initiate all the changes immediately. Running applications are interrupted by the queue manager as it shuts down and restarts.
   - Stop individual cluster-sender channels and restart them.

     Choose this option to change a few channels immediately. Running applications experience a short delay in message transfer between your stopping and starting the message channel again. The cluster-sender channel remains running, except during the time you stopped it. During the switch process messages are delivered to the old transmission queue, transferred to the new transmission queue by the switching process, and forwarded from the new transmission queue by the cluster-sender channel.

3. Optional: Monitor the channels as they switch

   Display the channel status and the transmission queue depth during the switch. The following example display the status for system cluster transmission queues.

   ```
   DISPLAY CHSTATUS(*) WHERE(XMITQ LK 'SYSTEM.CLUSTER.TRANSMIT.*')
   DISPLAY CHSTATUS(*) SAVED WHERE(XMITQ LK 'SYSTEM.CLUSTER.TRANSMIT.*')
   DISPLAY QUEUE('SYSTEM.CLUSTER.TRANSMIT.*') CURDEPTH
   ```

4. Optional: Monitor the messages "AMQ7341 The `transmission queue for channel` *`ChannelName`* `switched from queue` *`QueueName`* `to` *`QueueName`*" that are written to the queue manager error log.

Before you begin

You might make some configuration changes, and now want to make them effective without starting the cluster-sender channels that are affected. Alternatively, you make the configuration changes you require as one of the steps in the task.

If you reduce the number of messages the switching process has to transfer to the new transmission queue, switching completes more quickly. Read "How the process to switch cluster-sender channel to a different transmission queue works" on page 169 for the reasons for trying to empty the transmission queue before proceeding any further.

About this task

This task switches the transmission queues served by stopped or inactive cluster-sender channels. You might do this task because a cluster-sender channel is stopped, and you want to switch its transmission queue immediately. For example, for some reason a cluster-sender channel is not starting, or has some other configuration problem. To resolve the problem, you decide to create a cluster-sender channel, and associate the transmission queue for the old cluster-sender channel with the new cluster-sender channel you defined.

A more likely scenario is you want to control when reconfiguration of cluster transmission queues is performed. To fully control the reconfiguration, you stop the channels, change the configuration, and then switch the transmission queues.

Procedure

1. Stop the channels that you intend to switch
   a) Stop any running or inactive channels that you intend to switch. Stopping an inactive cluster-sender channel prevents it starting while you are making configuration changes.

   ```
   STOP CHANNEL(ChannelName) MODE(QUIESCSE) STATUS(STOPPED)
   ```

2. Optional: Make the configuration changes.

   For example, see "Clustering: Example configuration of multiple cluster transmission queues" on page 286.
3. Switch the cluster-sender channels to the new cluster transmission queues.

   ```
   runswchl -m QmgrName -c ChannelName
   ```

   The **runswchl** command transfers any messages on the old transmission queue to the new transmission queue. When the number of messages on the old transmission queue for this channel reaches zero, the switch is completed. The command is synchronous. The command writes progress messages to the window during the switching process.

   During the transfer phase existing and new messages destined for the cluster-sender channel are transferred in order to the new transmission queue.

   Because the cluster-sender channel is stopped, the messages build up on the new transmission queue. Contrast the stopped cluster-sender channel, to step "2" on page 296 in "Switching active cluster-sender channels to another set of cluster-transmission queues" on page 295. In that step, the cluster-sender channel is running, so messages do not necessarily build up on the new transmission queue.
4. Optional: Monitor the channels as they switch

   In a different command window, display the transmission queue depth during the switch. The following example display the status for system cluster transmission queues.

   ```
   DISPLAY QUEUE('SYSTEM.CLUSTER.TRANSMIT.*') CURDEPTH
   ```

5. Optional: Monitor the messages "AMQ7341 The `transmission queue for channel` *ChannelName* `switched from queue` *QueueName* `to` *QueueName*" that are written to the queue manager error log.
6. Restart the cluster-sender channels that you stopped.

   The channels do not start automatically, as you stopped them, placing them into STOPPED status.

   ```
   START CHANNEL(ChannelName)
   ```

Related reference
runswchl
RESOLVE CHANNEL
STOP CHANNEL

Clustering: Migration and modification best practices

This topic provides guidance for planning and administering IBM WebSphere MQ clusters. This information is a guide based on testing and feedback from customers.

1. “Moving objects in a cluster” on page 298 (Best practices for moving objects around inside a cluster, without installing any fix packs or new versions of IBM WebSphere MQ).
2. “Upgrades and maintenance installations” on page 299 (Best practices for keeping a working cluster architecture up and running, while applying maintenance or upgrades and testing the new architecture).

Moving objects in a cluster

Applications and their queues

When you must move a queue instance hosted on one queue manager to be hosted on another, you can work with the workload balancing parameters to ensure a smooth transition.

Create an instance of the queue where it is to be newly hosted, but use cluster workload balancing settings to continue sending messages to the original instance until your application is ready to switch. This is achieved with the following steps:

1. Set the **CLWLRANK** property of the existing queue to a high value, for example five.
2. Create the new instance of the queue and set its **CLWLRANK** property to zero.
3. Complete any further configuration of the new system, for example deploy and start consuming applications against the new instance of the queue.
4. Set the **CLWLRANK** property of the new queue instance to be higher than the original instance, for example nine.
5. Allow the original queue instance to process any queued messages in the system and then delete the queue.

Moving entire queue managers

If the queue manager is staying on the same host, but the IP address is changing, then the process is as follows:

- DNS, when used correctly, can help simplify the process. For information about using DNS by setting the Connection name (CONNAME) channel attribute, see ALTER CHANNEL .
- If moving a full repository, ensure that you have at least one other full repository which is running smoothly (no problems with channel status for example) before making changes.
- Suspend the queue manager using the SUSPEND QMGR command to avoid traffic buildup.
- Modify the IP address of the computer. If your CLUSRCVR channel definition uses an IP address in the CONNAME field, modify this IP address entry. The DNS cache might need to be flushed through to ensure that updates are available everywhere.

- When the queue manager reconnects to the full repositories, channel auto-definitions automatically resolve themselves.
- If the queue manager hosted a full repository and the IP address changes, it is important to ensure that partials are switched over as soon as possible to point any manually defined CLUSSDR channels to the new location. Until this switch is carried out, these queue managers might be able to only contact the remaining (unchanged) full repository, and warning messages might be seen regarding the incorrect channel definition.
- Resume the queue manager using the RESUME QMGR command.

If the queue manager must be moved to a new host, it is possible to copy the queue manager data and restore from a backup. This process is not recommended however, unless there are no other options; it might be better to create a queue manager on a new machine and replicate queues and applications as described in the previous section. This situation gives a smooth rollover/rollback mechanism.

If you are determined to move a complete queue manager using backup, follow these best practices:

- Treat the whole process as a queue manager restore from backup, applying any processes you would usually use for system recovery as appropriate for your operating system environment.
- Use the **REFRESH CLUSTER** command after migration to discard all locally held cluster information (including any auto-defined channels that are in doubt), and force it to be rebuilt.

  Note: For large clusters, use of the **REFRESH CLUSTER** command can be disruptive to the cluster while it is in progress, and again at 27 day intervals thereafter when the cluster objects automatically send status updates to all interested queue managers. See Refreshing in a large cluster can affect performance and availability of the cluster.

When creating a queue manager and replicating the setup from an existing queue manager in the cluster (as described previously in this topic), never treat the two different queue managers as actually being the same. In particular, do not give a new queue manager the same queue manager name and IP address. Attempting to 'drop in' a replacement queue manager is a frequent cause of problems in IBM WebSphere MQ clusters. The cache expects to receive updates including the **QMID** attribute, and state can be corrupted.

If two different queue managers are accidentally created with the same name, it is recommended to use the RESET CLUSTER **QMID** command to eject the incorrect entry from the cluster.

Upgrades and maintenance installations

Avoid the "big bang scenario" (for example, stopping all cluster and queue manager activity, applying all upgrades and maintenance to all queue managers, then starting everything at the same time): Clusters are designed to still work with multiple versions of queue manager coexisting, so a well-planned, phased maintenance approach is recommended.

Have a backup plan:

- On z/OS, have you applied backwards migration PTFs?
- Have you taken backups?
- Avoid using new cluster functionality immediately: Wait until you are sure that all the queue managers are upgraded to the new level, and are certain that you are not going to roll any of them back. Using new cluster function in a cluster where some queue managers are still at an earlier level can lead to undefined behavior. For example, in the move to IBM WebSphere MQ Version 7.1 from IBM WebSphere MQ Version 6.0, if a queue manager defines a cluster topic, IBM WebSphere MQ Version 6.0 queue managers will not understand the definition or be able to publish on this topic.

Migrate full repositories first. Although they can forward information that they do not understand, they cannot persist it, so it is not the recommended approach unless absolutely necessary. For more information, see Queue manager cluster migration.

Related concepts

“Clustering: Using REFRESH CLUSTER best practices” on page 300

You use the **REFRESH CLUSTER** command to discard all locally held information about a cluster and rebuild that information from the full repositories in the cluster. You should not need to use this command, except in exceptional circumstances. If you do need to use it, there are special considerations about how you use it. This information is a guide based on testing and feedback from customers.

Clustering: Using REFRESH CLUSTER best practices

You use the **REFRESH CLUSTER** command to discard all locally held information about a cluster and rebuild that information from the full repositories in the cluster. You should not need to use this command, except in exceptional circumstances. If you do need to use it, there are special considerations about how you use it. This information is a guide based on testing and feedback from customers.

Only run REFRESH CLUSTER if you really need to do so

The IBM WebSphere MQ cluster technology ensures that any change to the cluster configuration, such as a change to a clustered queue, automatically becomes known to any member of the cluster that needs to know the information. There is no need for further administrative steps to be taken to achieve this propagation of information.

If such information does not reach the queue managers in the cluster where it is required, for example a clustered queue is not known by another queue manager in the cluster when an application attempts to open it for the first time, it implies a problem in the cluster infrastructure. For example, it is possible that a channel cannot be started between a queue manager and a full repository queue manager. Therefore, any situation where inconsistencies are observed must be investigated. If possible, resolve the situation without using the **REFRESH CLUSTER** command.

In rare circumstances that are documented elsewhere in this product documentation, or when requested by IBM support, you can use the **REFRESH CLUSTER** command to discard all locally held information about a cluster and rebuild that information from the full repositories in the cluster.

Refreshing in a large cluster can affect performance and availability of the cluster

Use of the **REFRESH CLUSTER** command can be disruptive to the cluster while it is in progress, for example by creating a sudden increase in work for the full repositories as they process the repropagation of queue manager cluster resources. If you are refreshing in a large cluster (that is, many hundreds of queue managers) you should avoid use of the command in day-to-day work if possible and use alternative methods to correct specific inconsistencies. For example, if a cluster queue is not being correctly propagated across the cluster, an initial investigation technique of updating the clustered queue definition, such as altering its description, repropagates the queue configuration across the cluster. This process can help to identify the problem and potentially resolve a temporary inconsistency.

If alternative methods cannot be used, and you have to run **REFRESH CLUSTER** in a large cluster, you should do so at off-peak times or during a maintenance window to avoid impact on user workloads. You should also avoid refreshing a large cluster in a single batch, and instead stagger the activity as explained in “Avoid performance and availability issues when cluster objects send automatic updates” on page 300.

Avoid performance and availability issues when cluster objects send automatic updates

After a new cluster object is defined on a queue manager, an update for this object is generated every 27 days from the time of definition, and sent to every full repository in the cluster and onwards to any other interested queue managers. When you issue the **REFRESH CLUSTER** command to a queue manager, you reset the clock for this automatic update on all objects defined locally in the specified cluster.

If you refresh a large cluster (that is, many hundreds of queue managers) in a single batch, or in other circumstances such as recreating a system from configuration backup, after 27 days all of those queue managers will re-advertise all of their object definitions to the full repositories at the same time. This could again cause the system to run significantly slower, or even become unavailable, until all the updates have completed. Therefore, when you have to refresh or recreate multiple queue managers in a large

cluster, you should stagger the activity over several hours, or several days, so that subsequent automatic updates do not regularly impact system performance.

The system cluster history queue

When a **REFRESH CLUSTER** is performed, the queue manager takes a snapshot of the cluster state before the refresh and stores it on the SYSTEM.CLUSTER.HISTORY.QUEUE (SCHQ) if it is defined on the queue manager. This snapshot is for IBM service purposes only, in case of later problems with the system. The SCHQ is defined by default on distributed queue managers on startup. For z/OS migration, the SCHQ must be manually defined. Messages on the SCHQ expire after three months.

Related concepts

Application issues seen when running REFRESH CLUSTER

REFRESH CLUSTER considerations for publish/subscribe clusters

Related reference

MQSC Commands reference: REFRESH CLUSTER

Clustering: Availability, multi instance, and disaster recovery

This topic provides guidance for planning and administering IBM WebSphere MQ clusters. This information is a guide based on testing and feedback from customers.

IBM WebSphere MQ Clustering itself is not a High Availability solution, but in some circumstances it can be used to improve the availability of services using IBM WebSphere MQ, for example by having multiple instances of a queue on different queue managers. This section gives guidance on ensuring that the IBM WebSphere MQ infrastructure is as highly available as possible so that it can be used in such an architecture.

Availability of cluster resources

The reason for the usual recommendation to maintain two full repositories is that the loss of one is not critical to the smooth running of the cluster. Even if both become unavailable, there is a 60 day grace period for existing knowledge held by partial repositories, although new or not previously accessed resources (queues for example) are not available in this event.

Using clusters to improve application availability

A cluster can help in designing highly available applications (for example a request/response type server application), by using multiple instances of the queue and application. If needed, priority attributes can give preference to the 'live' application unless a queue manager or channel for example become unavailable. This is powerful for switching over quickly to continue processing new messages when a problem occurs.

However, messages which were delivered to a particular queue manager in a cluster are held only on that queue instance, and are not available for processing until that queue manager is recovered. For this reason, for true data high availability you might want to consider other technologies such as multi-instance queue managers.

Multi-instance queue managers

Software High Availability (multi instance) is the best built-in offering for keeping your existing messages available. See “Using WebSphere MQ with high availability configurations” on page 310, “Create a multi-instance queue manager” on page 338, and the following section for more information. Any queue manager in a cluster may be made highly available using this technique, as long as all queue managers in the cluster are running at least IBM WebSphere MQ Version 7.0.1. If any queue managers in the cluster are at previous levels, they might lose connectivity with the multi-instance queue managers if they fail over to a secondary IP.

As discussed previously in this topic, as long as two full repositories are configured, they are almost by their nature highly available. If you need to, IBM WebSphere MQ software High Availability / multi-instance queue managers can be used for full repositories. There is no strong reason to use these methods, and in fact for temporary outages these methods might cause additional performance cost during the failover. Using software HA instead of running two full repositories is discouraged

because in the event of a single channel outage, for example, it would not necessarily fail over, but might leave partial repositories unable to query for cluster resources.

Disaster recovery

Disaster recovery, for example recovering from when the disks storing a queue manager's data becomes corrupt, is difficult to do well; IBM WebSphere MQ can help, but it cannot do it automatically. The only 'true' disaster recovery option in IBM WebSphere MQ (excluding any operating system or other underlying replication technologies) is restoration from a backup. There are some cluster specific points to consider in these situations:

- Take care when testing disaster recovery scenarios. For example, if testing the operation of backup queue managers, be careful when bringing them online in the same network as it is possible to accidentally join the live cluster and start 'stealing' messages by hosting the same named queues as in the live cluster queue managers.
- Disaster recovery testing must not interfere with a running live cluster. Techniques to avoid interference include:
  - Complete network separation or separation at the firewall level.
  - Not issuing live SSL certificate to the disaster recovery system until, or unless, an actual disaster recovery scenario occurs.
- When restoring a backup of a queue manager in the cluster it is possible that the backup is out of sync with the rest of the cluster. The **REFRESH CLUSTER** command can resolve updates and synchronize with the cluster but the **REFRESH CLUSTER** command must be used as a last resort. See “Clustering: Using REFRESH CLUSTER best practices” on page 300. Review any in-house process documentation and IBM WebSphere MQ documentation to see whether a simple step was missed before resorting to using the command.
- As for any recovery, applications must deal with replay and loss of data. It must be decided whether to clear the queues down to a known state, or if there is enough information elsewhere to manage replays.

Clustering: Monitoring

This topic provides guidance for planning and administering IBM WebSphere MQ clusters. This information is a guide based on testing and feedback from customers.

Monitoring application messages in the cluster

Typically, all cluster messages that leave the queue manager pass through the `SYSTEM.CLUSTER.TRANSMIT.QUEUE`, irrespective of which cluster sender channel is being used to transmit the message. Each channel is draining messages targeted for that channel in parallel with all other cluster sender channels. A growing build-up of messages on this queue can indicate a problem with one or more channels and must be investigated:

- The depth of the queue must be monitored appropriately for the cluster design.
- The following command returns all channels that have more than one message that is waiting on the transmit queue:

  ```
  DIS CHSTATUS(*) WHERE(XQMSGSA GT 1)
  ```

  With all cluster messages on a single queue, it is not always easy to see which channel has problems when it begins to fill up. Using this command is an easy way to see which channel is responsible.

You can configure a cluster queue manager to have multiple transmission queues. If you change the queue manager attribute `DEFCLXQ` to `CHANNEL`, every cluster-sender channel is associated with a different cluster transmit queue. Alternatively you can configure separate transmission queues manually. To display all the cluster transmit queues that are associated with cluster-sender channels, run the command:

```
DISPLAY CLUSQMGR (qmgrName) XMITQ
```

Define cluster transmission queues so that they follow the pattern of having the fixed stem of the queue name on the left. You can then query the depth of all the cluster transmission queues returned by the **DISPLAY CLUSMGR** command, by using a generic queue name:

```
DISPLAY QUEUE (qname*) CURDEPTH
```

Monitoring control messages in the cluster

The SYSTEM.CLUSTER.COMMAND.QUEUE queue is used for processing all cluster control messages for a queue manager, either generated by the local queue manager or sent to this queue manager from other queue managers in the cluster. When a queue manager is correctly maintaining its cluster state, this queue tends toward zero. There are situations where the depth of messages on this queue can temporarily grow however:

- Having lots of messages on the queue indicates churn in the cluster state.
- When making significant changes, allow the queue to settle in between those changes. For example, when moving repositories, allow the queue to reach zero before moving the second repository.

While a backlog of messages exists on this queue, updates to the cluster state or cluster-related commands are not processed. If messages are not being removed from this queue for a long time, further investigation is required, initially through inspection of the queue manager error logs which might explain the process that is causing this situation.

The SYSTEM.CLUSTER.REPOSITORY.QUEUE holds the cluster repository cache information as a number of messages. It is usual for messages to always exist on this queue, and more for larger clusters. Therefore, the depth of messages on this queue is not an issue for concern.

Monitoring logs

Problems that occur in the cluster might not show external symptoms to applications for many days (and even months) after the problem originally occurs due to the caching of information and the distributed nature of clustering. However, the original problem is often reported in the IBM WebSphere MQ error logs. For this reason, it is vital to actively monitor these logs for any messages written that relate to clustering. These messages must be read and understood, with any action taken where necessary.

For example: A break in communications with a queue manager in a cluster can result in knowledge of certain cluster resources that are being deleted due to the way that clusters regularly revalidate the cluster resources by republishing the information. A warning of such an event potentially occurring is reported by the message AMQ9465. This message indicates that the problem needs to be investigated.

Special considerations for load balancing

When the cluster load balances between two or more instances of a queue, consuming applications must be processing messages on each of the instances. If one or more of those consuming applications terminates or stops processing messages, it is possible that clustering might continue to send messages to those instances of the queue. In this situation, those messages are not processed until the applications are functioning correctly again. For this reason the monitoring of the applications is an important part of the solution and action must be taken to reroute messages in that situation. An example of a mechanism to automate such monitoring can be found in this sample: The Cluster Queue Monitoring sample program (AMQSCLM) .

Availability, recovery and restart

Make your applications highly available by maintaining queue availability if a queue manager fails, and recover messages after server or storage failure.

Improve client application availability by using client reconnection to switch a client automatically between a group of queue managers, or to the new active instance of a multi-instance queue manager after a queue manager failure. Automatic client reconnect is not supported by WebSphere MQ classes for Java.

On Windows, UNIX, Linux and IBM i platforms deploy server applications to a multi-instance queue manager, which is configured to run as a single queue manager on multiple servers; if the server running the active instance fails, execution is automatically switched to a standby instance of the same queue manager on a different server. If you configure server applications to run as queue manager services, they are restarted when a standby instance becomes the actively running queue manager instance.

You can configure WebSphere MQ as part of a platform-specific clustering solution such as Microsoft Cluster Server, or PowerHA for AIX (formerly HACMP on AIX) and other UNIX and Linux clustering solutions.

Another way to increase server application availability is to deploy server applications to multiple computers in a queue manager cluster.

A messaging system ensures that messages entered into the system are delivered to their destination. WebSphere MQ can trace the route of a message as it moves from one queue manager to another using the **dspmqrte** command. If a system fails, messages can be recovered in various ways depending on the type of failure, and the way a system is configured.

WebSphere MQ ensures that messages are not lost by maintaining recovery logs of the activities of the queue managers that handle the receipt, transmission, and delivery of messages. It uses these logs for three types of recovery:

1. *Restart recovery*, when you stop WebSphere MQ in a planned way.
2. *Failure recovery*, when a failure stops WebSphere MQ.
3. *Media recovery*, to restore damaged objects.

In all cases, the recovery restores the queue manager to the state it was in when the queue manager stopped, except that any in-flight transactions are rolled back, removing from the queues any updates that were in-flight at the time the queue manager stopped. Recovery restores all persistent messages; nonpersistent messages might be lost during the process.

Automatic client reconnection

You can make your client applications reconnect automatically, without writing any additional code, by configuring a number of components.

Automatic client reconnection is *inline*. The connection is automatically restored at any point in the client application program, and the handles to open objects are all restored.

In contrast, manual reconnection requires the client application to re-create a connection using `MQCONN` or `MQCONNX`, and to reopen objects. Automatic client reconnection is suitable for many, but not all client applications.

Table 28 on page 305 lists the earliest release of IBM WebSphere MQ client support that must be installed on a client workstation. You must upgrade client workstations to one of these levels for an application to use automatic client reconnection. Table 29 on page 305 lists other requirements to enable automatic client reconnection.

With program access to reconnection options, a client application can set reconnection options. Except for JMS and XMS clients, if a client application has access to reconnection options, it can also create an event handler to handle reconnection events.

An existing client application might be able to benefit from reconnection support, without recompilation and linking:

- For a non-JMS client, set the `mqclient.ini` environment variable `DefRecon` to set reconnection options. Use a CCDT to connect to a queue manager. If the client is to connect to a multi-instance queue manager, provide the network addresses of the active and standby queue manager instances in the CCDT.
- For a JMS client, set the reconnection options in the connection factory configuration. When you use the WebSphere MQ Resource adapter or a JMS Client that is integrated in a Java EE Environment, automatic client reconnection might not be available. There are restrictions in some of the managed environments, for more information see Using automatic client reconnection in Java SE and Java EE environments.

Note: Automatic client reconnection is not supported by WebSphere MQ classes for Java.

Table 28. Supported clients

Client interface	Client	Program access to reconnection options	Reconnection support
Messaging APIs	C, C++, COBOL, Unmanaged Visual Basic, XMS (Unmanaged XMS on Windows)	7.0.1	7.0.1
	JMS (JSE, and Java EE client container and managed containers)	7.0.1.3	7.0.1.3
	WebSphere MQ classes for Java	Not supported	Not supported
	Managed XMS and managed .NET clients: C#, Visual Basic	7.1	7.1
Other APIs	Windows Communication Foundation (Unmanaged[1])	Not supported	7.0.1
	Windows Communication Foundation (Managed[1])	Not supported	Not supported
	Axis 1	Not supported	Not supported
	Axis 2	Not supported	7.0.1.3
	HTTP (web 2.0)	Not supported	7.0.1.3

1. Set managed or unmanaged mode in the WCF binding configuration.

Automatic reconnection has the following configuration requirements:

Table 29. Automatic reconnection configuration requirements

Component	Requirement	Effect of not meeting requirement
WebSphere MQ MQI client installation	See Table 28 on page 305	`MQRC_OPTIONS_ERROR`
WebSphere MQ Server installation	Level 7.0.1	`MQRC_OPTIONS_ERROR`
Channel	`SHARECNV > 0`	`MQRC_ENVIRONMENT_ERROR`
Application environment	Must be threaded	`MQRC_ENVIRONMENT_ERROR`

Table 29. Automatic reconnection configuration requirements (continued)		
Component	**Requirement**	**Effect of not meeting requirement**
MQI	One of: • MQCONNX with MQCNO Options set to MQCNO_RECONNECT or MQCNO_RECONNECT_Q_MGR. • Defrecon=YES | QMGR in mqclient.ini • In JMS set the CLIENTRECONNECTOPTIONS property of the connection factory.	MQCC_FAILED when a connection is broken or queue manager ends or fails.

Figure 59 on page 306 shows the main interactions between components that are involved in client reconnection.

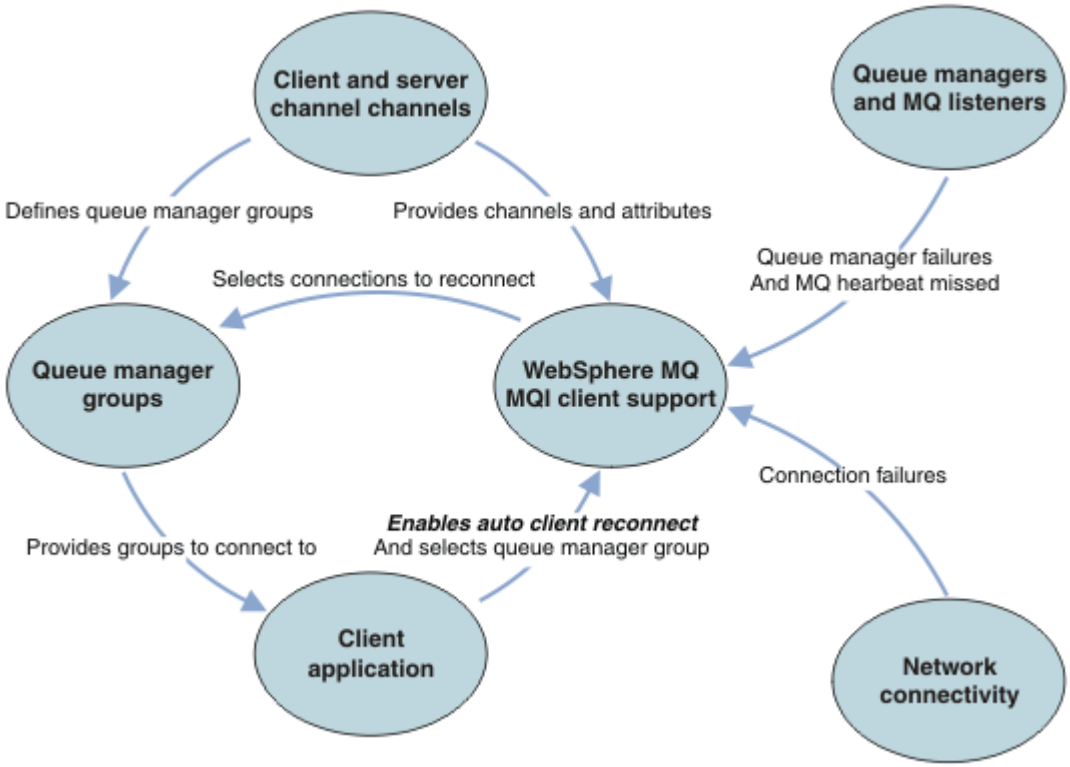


Figure 59. Automatic client reconnection

Client application

The client application is an IBM WebSphere MQ MQI client.

- By default clients are not automatically reconnected. Enable the automatic client reconnection by setting the MQCONNX MQCNO Option MQCNO_RECONNECT or MQCNO_RECONNECT_Q_MGR.
- Many applications are written in such a way that they are able to take advantage of auto-reconnection with no additional coding. Enable automatic reconnection for existing programs, without making any

coding changes, by setting the DefRecon attribute in the channels stanza of the mqclient.ini configuration file.

- Use one of these three options:
  1. Modify the program so that the logic is unaffected by reconnection. For example, you might have to issue MQI calls within the sync point, and resubmit backed-out transactions.
  2. Add an event handler to detect reconnection, and restore the state of the client application when the connection is reestablished.
  3. Do not enable auto-reconnection: instead, disconnect the client and issue a new MQCONN or MQCONNX MQI call to find another queue manager instance that is running in the same queue manager group.

  For further details about these three options, see "Application recovery" on page 382.
- Reconnecting to a queue manager of the same name does not guarantee that you have reconnected to the same instance of a queue manager.

  Use an MQCNO option MQCNO_RECONNECT_Q_MGR , to reconnect to an instance of the same queue manager.
- A client can register an event handler so that it can be informed the state of reconnection. The MQHCONN passed in the event handler cannot be used. The following reason codes are provided:

  MQRC_RECONNECTING
  The connection failed, and the system is attempting to reconnect. You receive multiple MQRC_RECONNECTING events if multiple reconnect attempts are made.

  MQRC_RECONNECTED
  The reconnection made and all handles successfully reestablished.

  MQRC_RECONNECT_FAILED
  The reconnection was not successful.

  MQRC_RECONNECT_QMID_MISMATCH
  A reconnectable connection specified MQCNO_RECONNECT_Q_MGR and the connection attempted to reconnect to a different queue manager.

  MQRC_RECONNECT_Q_MGR_REQD
  An option, such MQMO_MATCH_MSG_TOKEN in an MQGET call, was specified in the client program that requires reconnection to the same queue manager.
- A reconnectable client is able to reconnect automatically only *after* connecting. That is, the MQCONNX call itself is not tried again if it fails. For example, if you receive the return code 2543 - MQRC_STANDBY_Q_MGR from MQCONNX, reissue the call after a short delay.

  MQRC_RECONNECT_INCOMPATIBLE
  This reason code is returned when the application tries to use MQPMO_LOGICAL_ORDER (with MQPUT and MQPUT1) or MQGMO_LOGICAL_ORDER (with MQGET) when reconnect options are set. The reason for returning the reason code is to make sure that applications never use reconnect in such cases.

  MQRC_CALL_INTERRUPTED
  This reason code is returned when the connection breaks during the execution of Commit call and the client reconnects. An MQPUT of a persistent message outside the sync point also results in the same reason code being returned to the application.

Multi-instance queue managers

Simplify restarting WebSphere MQ MQI client applications, after a multi-instance queue manager has activated its standby instance, by using automatic client reconnection.

The standby instance of a multi-instance queue manager is typically at a different network address to the active instance. Include the network addresses of both the instances in the client connection definition table (CCDT). Either provide a list of network addresses for the **CONNAME** parameter, or define multiple rows for the queue manager in the CCDT.

Commonly, WebSphere MQ MQI clients reconnect to any queue manager in a queue manager group. Sometimes you want a WebSphere MQ MQI client to reconnect only to the same queue manager. It might have an affinity to a queue manager. You can prevent a client from reconnecting to a different queue manager. Set the MQCNO option, MQCNO_RECONNECT_Q_MGR. The WebSphere MQ MQI client fails if it reconnects to a different queue manager. If you set the MQCNO option, MQCNO_RECONNECT_Q_MGR, do not include other queue managers in the same queue manager group. The client returns an error if the queue manager it reconnects to is not the same queue manager as the one it connected to.

Queue manager groups

You can select whether the client application always connects and reconnects to a queue manager of the same name, to the same queue manager, or to any of a set of queue managers that are defined with the same QMNAME value in the client connection table.

- The queue manager *name* attribute, QMNAME , in the client channel definition is the name of a queue manager group.
- In your client application, if you set the value of the MQCONN or MQCONNX QmgrName parameter to a queue manager name, the client connects only to queue managers with that name. If you prefix the queue manager name with an asterisk(*), the client connects to any queue manager in the queue manager group with the same QMNAME value. For a full explanation, see Queue manager groups in the CCDT.

Queue sharing groups

Automatic client reconnection to z/OS queue sharing groups, uses the same mechanisms for reconnection as any other environment. The client will reconnect to the same selection of queue managers as is configured for the original connection. For example, when using the client channel definition table the administrator should ensure that all entries in the table, resolve to the same z/OS queue sharing group.

Client and server channel definitions

Client and server channel definitions define the groups of queue managers a client application can reconnect to. The definitions govern the selection and timing of reconnections, and other factors, such as security; see the related topics. The most relevant channel attributes to consider for reconnection are listed in two groups:

Client connection attributes

Connection affinity (AFFINITY)AFFINITY
Connection affinity.

Client channel weight (CLNTWGHT)CLNTWGHT
Client channel weight.

Connection name (CONNAME)CONNAME
Connection information.

Heartbeat interval (HBINT)HBINT
Heartbeat interval. Set the heartbeat interval on the server connection channel.

Keepalive Interval (KAINT)KAINT
Keepalive interval. Set the keepalive interval on the server connection channel.

Note that KAINT applies to z/OS only.

Queue manager name (QMNAME)QMNAME
Queue manager name.

Server connection attributes

Heartbeat interval (HBINT)HBINT
Heartbeat interval. Set the heartbeat interval on the client connection channel.

Keepalive Interval (KAINT)KAINT
Keepalive interval. Set the keepalive interval on the client connection channel.

Note that KAINT applies to z/OS only.

KAINT is a network layer heartbeat, and HBINT is a WebSphere MQ heartbeat between the client and the queue manager. Setting these heartbeats to a shorter time serves two purposes:

1. By simulating activity on the connection, network layer software that is responsible for closing inactive connections is less likely to shut down your connection.
2. If the connection is shut down, the delay before the broken connection is detected, is shortened.

The default TCP/IP keepalive interval is two hours. Consider setting the KAINT and HBINT attributes to a shorter time. Do not assume that the normal behavior of a network suits the needs of automatic reconnection. For example, some firewalls can shut down an inactive TCP/IP connection after as little as 10 minutes.

Network connectivity

Only network failures that are passed to the WebSphere MQ MQI client by the network, are handled by the automatic reconnection capability of the client.

- Reconnections performed automatically by the transport are invisible to IBM WebSphere MQ.
- Setting HBINT helps to deal with network failures that are invisible to WebSphere MQ.

Queue managers and WebSphere MQ listeners

Client reconnection is triggered by server failure, queue manager failure, network connectivity failure, and by an administrator switching over to another queue manager instance.

- If you are using a multi-instance queue manager, an additional cause of client reconnection occurs when you switch control from the active queue manager instance to a standby instance.
- Ending a queue manager using the default **endmqm** command, does not trigger automatic client reconnection. Add the `-r` option on the **endmqm** command to request automatic client reconnection, or the `-s` option to transfer to a standby queue manager instance after shutting down.

WebSphere MQ MQI client automatic reconnection support

If you use the automatic client reconnection support in the WebSphere MQ MQI client, the client application automatically reconnects and continues processing without you issuing an MQCONN or MQCONNX MQI call to reconnect to the queue manager.

- Automatic client reconnection is triggered by one of the following occurrences:
  - queue manager failure
  - ending a queue manager and specifying the `-r`, reconnect, option on the **endmqm** command
- The MQCONNX MQCNO options control whether you have enabled the automatic client reconnection. The options are described in Reconnection options.
- Automatic client reconnection issues MQI calls on behalf of your application to restore the connection handle and the handles to other open objects, so that your program can resume normal processing after it has processed any MQI errors that resulted from the broken connection. See “Recovery of an automatically reconnected client” on page 384 .
- If you have written a channel exit program for the connection, the exit receives these additional MQI calls.
- You can register a reconnection event handler, which is triggered when reconnection begins and when it finishes.

Although reconnection takes no more than a minute, reconnection can take longer because a queue manager might have numerous resources to manage. During this time, a client application might be

holding locks that do not belong to WebSphere MQ resources. There is a timeout value you can configure to limit the time a client waits for reconnection. The value (in seconds) is set in the `mqclient.ini` file.

```
Channels:
    MQReconnectTimeout = 1800
```

No reconnection attempts are made after the timeout has expired. When the system detects that the timeout has expired it returns a `MQRC_RECONNECT_FAILED` error.

Console message monitoring

There are a number of information messages issued by the queue manager or channel initiator that should be considered particularly significant. These messages do not in themselves indicate a problem, but may be useful in tracking because they do indicate a potential issue which potentially needs addressing.

The presence of this message can also indicate that a user application is putting a large number of messages to the page set, which may be a symptom of a larger problem:

- A problem with the user application which PUT's messages, such as an uncontrolled loop.
- A user application which GET's the messages from the queue is no longer functioning.

Using WebSphere MQ with high availability configurations

If you want to operate your WebSphere MQ queue managers in a high availability (HA) configuration, you can set up your queue managers to work either with a high availability manager, such as PowerHA for AIX (formerly HACMP) or the Microsoft Cluster Service (MSCS), or with WebSphere MQ multi-instance queue managers.

You need to be aware of the following configuration definitions:

Queue manager clusters
Groups of two or more queue managers on one or more computers, providing automatic interconnection, and allowing queues to be shared among them for load balancing and redundancy.

HA clusters
HA clusters are groups of two or more computers and resources such as disks and networks, connected together and configured in such a way that, if one fails, a high availability manager, such as HACMP (UNIX) or MSCS (Windows) performs a *failover*. The failover transfers the state data of applications from the failing computer to another computer in the cluster and re-initiates their operation there. This provides high availability of services running within the HA cluster. The relationship between IBM WebSphere MQ clusters and HA clusters is described in “Relationship of HA clusters to queue manager clusters” on page 311.

Multi-instance queue managers
Instances of the same queue manager configured on two computers. By starting multiple instances, one instance becomes the active instance and the other instance becomes the standby. If the active instance fails, the standby instance running on a different computer automatically takes over. You can use multi-instance queue managers to configure your own highly available messaging systems based on WebSphere MQ, without requiring a cluster technology such as HACMP or MSCS. HA clusters and multi-instance queue managers are alternative ways of making queue managers highly available. Do not combine them by putting a multi-instance queue manager in an HA cluster.

Differences between multi-instance queue managers and HA clusters

Multi-instance queue managers and HA clusters are alternative ways to achieve high availability for your queue managers. Here are some points that highlight the differences between the two approaches.

Multi-instance queue managers include the following features:

- Basic failover support integrated into WebSphere MQ
- Faster failover than HA cluster

- Simple configuration and operation
- Integration with WebSphere MQ Explorer

Limitations of multi-instance queue managers include:

- Highly available, high performance networked storage required
- More complex network configuration because queue manager changes IP address when it fails over

HA clusters include the following features:

- The ability to coordinate multiple resources, such as an application server or database
- More flexible configuration options including clusters comprising more than two nodes
- Can failover multiple times without operator intervention
- Takeover of queue manager's IP address as part of the failover

Limitations of HA clusters include:

- Additional product purchase and skills are required
- Disks which can be switched between the nodes of the cluster are required
- Configuration of HA clusters is relatively complex
- Failover is rather slow historically, but recent HA cluster products are improving this
- Unnecessary failovers can occur if there are shortcomings in the scripts that are used to monitor resources such as queue managers

Relationship of HA clusters to queue manager clusters

Queue manager clusters reduce administration and provide load balancing of messages across instances of queue manager cluster queues. They also offer higher availability than a single queue manager because, following a failure of a queue manager, messaging applications can still access surviving instances of a queue manager cluster queue. However, queue manager clusters alone do not provide automatic detection of queue manager failure and automatic triggering of queue manager restart or failover. HA clusters provide these features. The two types of cluster can be used together to good effect.

Using WebSphere MQ with a high availability cluster on UNIX and Linux

You can use WebSphere MQ with a high availability (HA) cluster on UNIX and Linux platforms: for example, PowerHA for AIX (formerly HACMP), Veritas Cluster Server, HP Serviceguard, or a Red Hat Enterprise Linux cluster with Red Hat Cluster Suite.

Before WebSphere MQ Version 7.0.1, SupportPac MC91 was provided to assist in configuring HA clusters. WebSphere MQ Version 7.0.1 provided a greater degree of control than previous versions over where queue managers store their data. This makes it easier to configure queue managers in an HA cluster. Most of the scripts provided with SupportPac MC91 are no longer required, and the SupportPac is withdrawn.

This section introduces “HA cluster configurations” on page 311, the relationship of HA clusters to queue manager clusters, “WebSphere MQ clients” on page 312, and “WebSphere MQ operating in an HA cluster” on page 312, and guides you through the steps and provides example scripts that you can adapt to configure queue managers with an HA cluster.

Refer to the HA cluster documentation particular to your environment for assistance with the configuration steps described in this section.

HA cluster configurations

In this section the term *node* is used to refer to the entity that is running an operating system and the HA software; "computer", "system" or "machine" or "partition" or "blade" might be considered synonyms in this usage. You can use WebSphere MQ to help set up either standby or takeover configurations, including mutual takeover where all cluster nodes are running WebSphere MQ workload.

A *standby* configuration is the most basic HA cluster configuration in which one node performs work while the other node acts only as standby. The standby node does not perform work and is referred to as idle; this configuration is sometimes called *cold standby*. Such a configuration requires a high degree of hardware redundancy. To economize on hardware, it is possible to extend this configuration to have multiple worker nodes with a single standby node. The point of this is that the standby node can take over the work of any other worker node. This configuration is still referred to as a standby configuration and sometimes as an "N+1" configuration.

A *takeover* configuration is a more advanced configuration in which all nodes perform some work and critical work can be taken over in the event of a node failure.

A *one-sided takeover* configuration is one in which a standby node performs some additional, noncritical and unmovable work. This configuration is similar to a standby configuration but with (noncritical) work being performed by the standby node.

A *mutual takeover* configuration is one in which all nodes are performing highly available (movable) work. This type of HA cluster configuration is also sometimes referred to as "Active/Active" to indicate that all nodes are actively processing critical workload.

With the extended standby configuration or either of the takeover configurations it is important to consider the peak load that might be placed on a node that can take over the work of other nodes. Such a node must possess sufficient capacity to maintain an acceptable level of performance.

Relationship of HA clusters to queue manager clusters

Queue manager clusters reduce administration and provide load balancing of messages across instances of queue manager cluster queues. They also offer higher availability than a single queue manager because, following a failure of a queue manager, messaging applications can still access surviving instances of a queue manager cluster queue. However, queue manager clusters alone do not provide automatic detection of queue manager failure and automatic triggering of queue manager restart or failover. HA clusters provide these features. The two types of cluster can be used together to good effect.

WebSphere MQ clients

WebSphere MQ clients that are communicating with a queue manager that might be subject to a restart or takeover must be written to tolerate a broken connection and must repeatedly attempt to reconnect. WebSphere MQ Version 7 introduced features in the processing of the Client Channel Definition Table (CCDT) that assist with connection availability and workload balancing; however these are not directly relevant when working with a failover system.

The Extended Transactional Client (ETC), which allows a WebSphere MQ MQI client to participate in two-phase transactions, must always connect to the same queue manager. The ETC cannot use techniques such as an IP load balancer to select from a list of queue managers. When you use an HA product, a queue manager maintains its identity (name and address) whichever node it is running on, so the ETC can be used with queue managers that are under HA control.

WebSphere MQ operating in an HA cluster

All HA clusters have the concept of a unit of failover. This is a set of definitions that contains all the resources that make up the highly available service. The unit of failover includes the service itself and all other resources upon which it depends.

HA solutions use different terms for a unit of failover:

- On PowerHA for AIX the unit of failover is called a *resource group*.
- On Veritas Cluster Server it is known as a *service group*.
- On Serviceguard it is called a *package*.

This topic uses the term *resource group* to mean a unit of failover.

The smallest unit of failover for WebSphere MQ is a queue manager. Typically, the resource group containing the queue manager also contains shared disks in a volume group or disk group that is reserved exclusively for use by the resource group, and the IP address that is used to connect to the queue manager. It is also possible to include other WebSphere MQ resources, such as a listener or a trigger monitor in the same resource group, either as separate resources, or under the control of the queue manager itself.

A queue manager that is to be used in an HA cluster must have its data and logs on disks that are shared between the nodes in the cluster. The HA cluster ensures that only one node in the cluster at a time can write to the disks. The HA cluster can use a monitor script to monitor the state of the queue manager.

It is possible to use a single shared disk for both the data and logs that are related to the queue manager. However, it is normal practice to use separate shared file systems so that they can be independently sized and tuned.

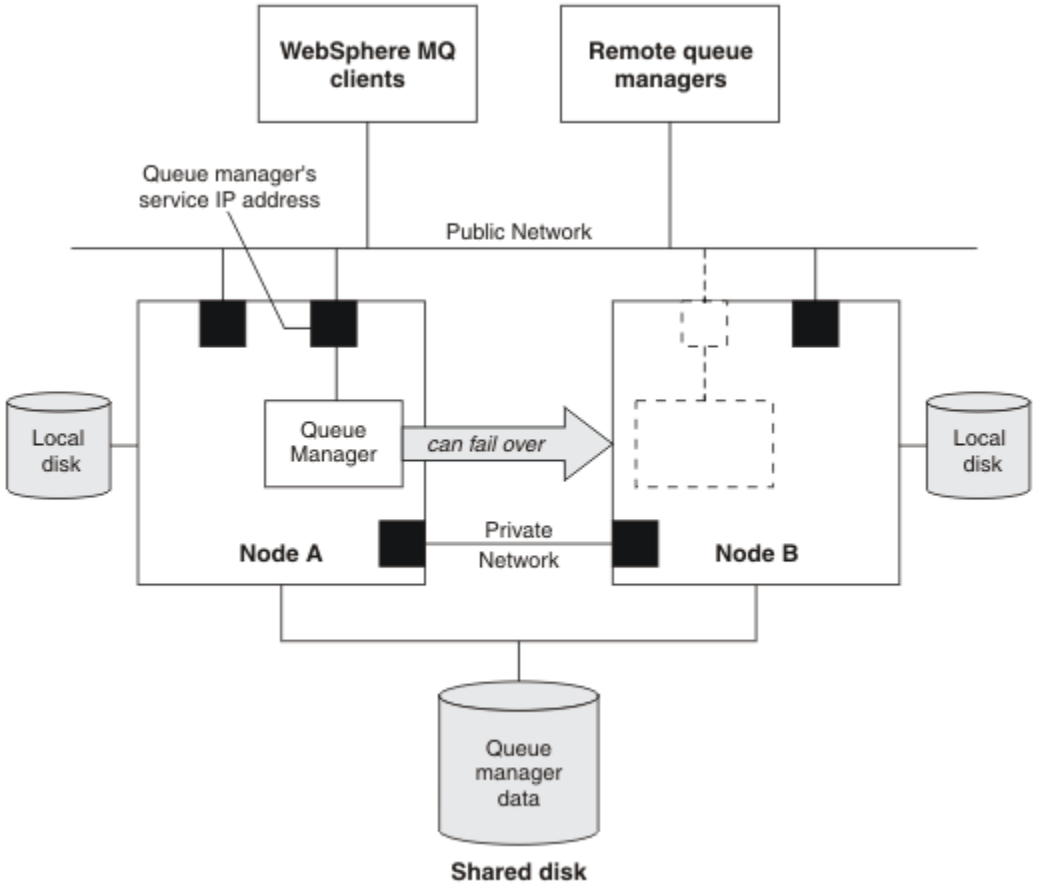


Figure 60. HA cluster

Figure 1 illustrates a HA cluster with two nodes. The HA cluster is managing the availability of a queue manager which has been defined in a resource group. This is an active/passive or cold standby configuration, because only one node, node A, is currently running a queue manager. The queue manager was created with its data and log files on a shared disk. The queue manager has a service IP address which is also managed by the HA cluster. The queue manager depends on the shared disk and its service IP address. When the HA cluster fails the queue manager over from node A to node B, it first moves the queue manager's dependent resources onto node B and then starts the queue manager.

If the HA cluster contains more than one queue manager, your HA cluster configuration might result in two or more queue managers running on the same node after a failover. Each queue manager in the HA cluster must be assigned its own port number, which it uses on whichever cluster node it happens to be active at any particular time.

Generally, the HA cluster runs as the root user. WebSphere MQ runs as the mqm user. Administration of WebSphere MQ is granted to members of the mqm group. Ensure that the mqm user and group both exist on all HA cluster nodes. The user ID and group ID must be consistent across the cluster. Administration of WebSphere MQ by the root user is not allowed; scripts that start, stop, or monitor scripts must switch to the mqm user.

Note: WebSphere MQ must be installed correctly on all nodes; you cannot share the product executable files.

Configuring the shared disks

A WebSphere MQ queue manager in an HA cluster requires data files and log files to be in common named remote file systems on a shared disk.

To configure the shared disks, complete the following steps:

1. Decide the names of the mount points for the queue manager's file systems. For example, `/MQHA/qmgrname/data` for the queue manager's data files and `/MQHA/qmgrname/log` for its log files.
2. Create a volume group (or disk group) to contain the queue manager's data and log files. This volume group is managed by the high availability (HA) cluster in the same resource group as the queue manager.
3. Create the file systems for the queue manager's data and log files in the volume group.
4. For each node in turn, create the mount points for the file systems and make sure that the file systems can be mounted. The mqm user must own the mount points.

Figure 1 shows a possible layout for a queue manager in an HA cluster. The queue manager's data and log directories are both on the shared disk which is mounted at /MQHA/QM1. This disk is switched between the nodes of the HA cluster when failover occurs so that the data is available wherever the queue manager is restarted. The `mqs.ini` file has a stanza for the QM1 queue manager. The Log stanza in the `qm.ini` file has a value for LogPath.

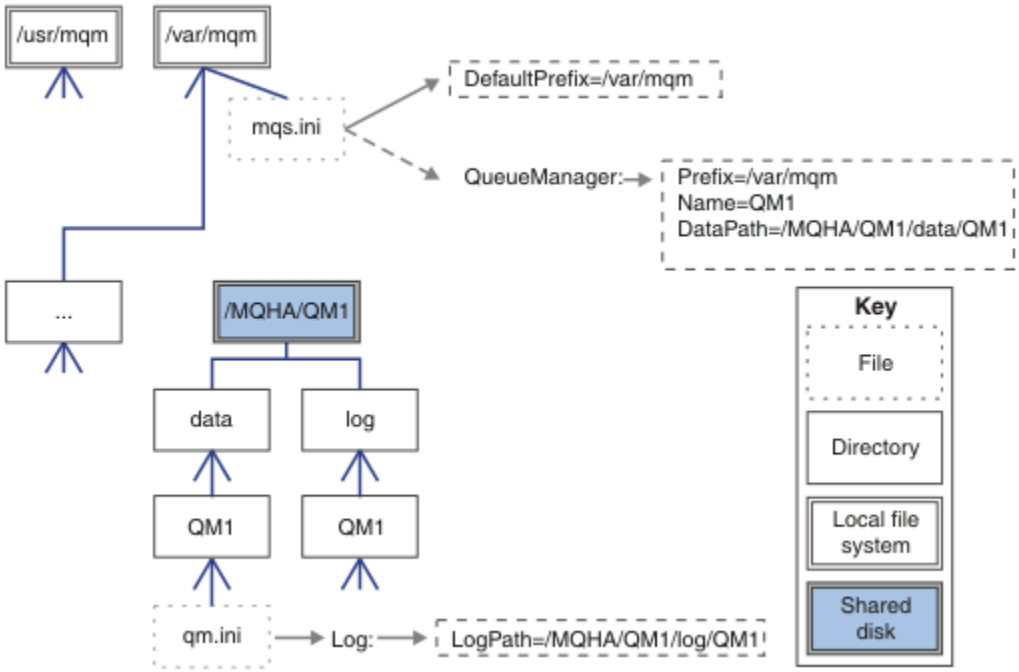


Figure 61. Shared named `data` and `log` directories

Creating a queue manager for use in a high availability (HA) cluster

The first step towards using a queue manager in a high availability cluster is to create the queue manager on one of the nodes.

To create a queue manager for use in an HA cluster, select one of the nodes in the cluster on which to create the queue manager. On this node complete the following steps:

1. Mount the queue manager's file systems on the node.
2. Create the queue manager by using the **crtmqm** command. For example:

   ```
   crtmqm -md /MQHA/qmgrname/data -ld /MQHA/qmgrname/log qmgrname
   ```
3. Start the queue manager manually by using the **strmqm** command.
4. Complete any initial configuration of the queue manager, such as creating queues and channels, and setting the queue manager to start a listener automatically when the queue manager starts.
5. Stop the queue manager by using the **endmqm** command.
6. Use the **dspmqinf** command to display the **addmqinf** command that you can use in a later task, which is documented in "Adding queue manager configuration information to other nodes in a high availability (HA) cluster" on page 315:

   ```
   dspmqinf -o command qmgrname
   ```

   where qmgrname is the name of the queue manager.
7. The **addmqinf** command that is displayed will be similar to the following example:

   ```
   addmqinf -sQueueManager -vName=qmgrname -vDirectory=qmgrname \
            -vPrefix=/var/mqm -vDataPath=/MQHA/qmgrname/data/qmgrname
   ```

   Make a careful note of the displayed command.
8. Unmount the queue manager's file systems.

You are now ready to complete the steps described in "Adding queue manager configuration information to other nodes in a high availability (HA) cluster" on page 315.

Adding queue manager configuration information to other nodes in a high availability (HA) cluster

You must add the queue manager configuration to the other nodes in the HA cluster.

Before you complete this task, you must have completed the steps in "Creating a queue manager for use in a high availability (HA) cluster" on page 315.

To add the configuration information for the queue manager to each of other nodes in the HA cluster, complete the following steps on each of the other nodes:

1. Mount the queue manager file systems.
2. Add the queue manager configuration information to the node, either by editing /var/mqm/mqs.ini directly, or by issuing the **addmqinf** command that was displayed by the **dspmqinf** command in steps 6 and 7 in "Creating a queue manager for use in a high availability (HA) cluster" on page 315.
3. Start and stop the queue manager to verify the configuration.

   The commands used to start and stop the queue manager must be issued from the same IBM WebSphere MQ installation as the **addmqinf** command. To start and stop the queue manager from a different installation, you must first set the installation associated with the queue manager using the **setmqm** command. For more information, see setmqm.
4. Unmount the queue manager file systems.

Starting a queue manager under control of a high availability (HA) cluster

The queue manager is represented in the HA cluster as a resource. The HA cluster must be able to start and stop the queue manager. In most cases you can use a shell script to start the queue manager. You

must make these scripts available at the same location on all nodes in the cluster, either using a network filesystem or by copying them to each of the local disks.

Note: Before you restart a failed queue manager, you must disconnect your applications from that instance of the queue manager. If you do not, the queue manager might not restart correctly.

Examples of suitable shell scripts are given here. You can tailor these to your needs and use them to start the queue manager under the control of your HA cluster.

The following shell script is an example of how to switch from the HA cluster user to the mqm user so that the queue manager can be successfully started:

```
#!/bin/ksh

# A simple wrapper script to switch to the mqm user.

su mqm -c name_of_your_script $*
```

The following shell script is an example of how to start a queue manager without making any assumptions about the current state of the queue manager. Note that it uses an extremely abrupt method of ending any processes that belong to the queue manager:

```
#!/bin/ksh
#
# This script robustly starts the queue manager.
#
# The script must be run by the mqm user.

# The only argument is the queue manager name. Save it as QM variable
QM=$1

if [ -z "$QM" ]
then
  echo "ERROR! No queue manager name supplied"
  exit 1
fi

# End any queue manager processes which might be running.

srchstr="( |-m)$QM *.*$"
for process in amqzmuc0 amqzxma0 amqfcxba amqfqpub amqpcsea amqzlaa0 \
               amqzlsa0 runmqchi runmqlsr amqcrsta amqrrmfa amqrmppa \
               amqzfuma amqzdmaa amqzmuf0 amqzmur0 amqzmgr0
 do
  ps -ef | tr "\t" " " | grep $process | grep -v grep | \
    egrep "$srchstr" | awk '{print $2}'| \
       xargs kill -9 > /dev/null 2>&1
done

# It is now safe to start the queue manager.
# The strmqm command does not use the -x flag.
strmqm ${QM}
```

You can modify the script to start other related programs.

Stopping a queue manager under the control of a high availability (HA) cluster

In most cases, you can use a shell script to stop a queue manager. Examples of suitable shell scripts are given here. You can tailor these to your needs and use them to stop the queue manager under control of your HA cluster.

The following script is an example of how to immediately stop without making assumptions about the current state of the queue manager. The script must be run by the mqm user; it might therefore be necessary to wrap this script in a shell script to switch the user from the HA cluster user to mqm (an example shell script is provided in "Starting a queue manager under control of a high availability (HA) cluster" on page 315):

```
#!/bin/ksh
#
# The script ends the QM by using two phases, initially trying an immediate
# end with a time-out and escalating to a forced stop of remaining
# processes.
```

```
#
# The script must be run by the mqm user.
#
# There are two arguments: the queue manager name and a timeout value.
QM=$1
TIMEOUT=$2

if [ -z "$QM" ]
then
  echo "ERROR! No queue manager name supplied"
  exit 1
fi

if [ -z "$TIMEOUT" ]
then
  echo "ERROR! No timeout specified"
  exit 1
fi

for severity in immediate brutal
do
  # End the queue manager in the background to avoid
  # it blocking indefinitely. Run the TIMEOUT timer
  # at the same time to interrupt the attempt, and try a
  # more forceful version. If the brutal version fails,
  # nothing more can be done here.

  echo "Attempting ${severity} end of queue manager '${QM}'"
  case $severity in

  immediate)
    # Minimum severity of endmqm is immediate which severs connections.
    # HA cluster should not be delayed by clients
    endmqm -i ${QM} &
    ;;

  brutal)
    # This is a forced means of stopping queue manager processes.

    srchstr="( |-m)$QM *.*$"
    for process in amqzmuc0 amqzxma0 amqfcxba amqfqpub amqpcsea amqzlaa0 \
                amqzlsa0 runmqchi runmqlsr amqcrsta amqrrmfa amqrmppa \
                amqzfuma amqzdmaa amqzmuf0 amqzmur0 amqzmgr0
    do
      ps -ef | tr "\t" " " | grep $process | grep -v grep | \
         egrep "$srchstr" | awk '{print $2}'| \
            xargs kill -9 > /dev/null 2>&1
    done

  esac

  TIMED_OUT=yes
  SECONDS=0
  while (( $SECONDS < ${TIMEOUT} ))
  do
   TIMED_OUT=yes
   i=0
   while [ $i -lt 5 ]
   do
     # Check for execution controller termination
     srchstr="( |-m)$QM *.*$"
     cnt=`ps -ef | tr "\t" " " | grep amqzxma0 | grep -v grep | \
       egrep "$srchstr" | awk '{print $2}' | wc -l `
     i=`expr $i + 1`
     sleep 1
     if [ $cnt -eq 0 ]
     then
       TIMED_OUT=no
       break
     fi
   done

   if [ ${TIMED_OUT} = "no" ]
   then
     break
   fi

   echo "Waiting for ${severity} end of queue manager '${QM}'"
   sleep 1
  done # timeout loop

  if [ ${TIMED_OUT} = "yes" ]
```

```
  then
    continue        # to next level of urgency
  else
    break           # queue manager is ended, job is done
  fi

done # next phase
```

Monitoring a queue manager

It is usual to provide a way for the high availability (HA) cluster to monitor the state of the queue manager periodically. In most cases, you can use a shell script for this. Examples of suitable shell scripts are given here. You can tailor these scripts to your needs and use them to make additional monitoring checks specific to your environment.

From WebSphere MQ version 7.1, it is possible to have multiple installations of WebSphere MQ coexisting on a system. For more information about multiple installations, see Multiple installations. If you intend to use the monitoring script across multiple installations, including installations at version 7.1, or higher, you might need to perform some additional steps. If you have a primary installation, or you are using the script with versions earlier than version 7.1, you do not need to specify the*MQ_INSTALLATION_PATH* to use the script. Otherwise, the following steps ensure that the *MQ_INSTALLATION_PATH* is identified correctly:

1. Use the **crtmqenv** command from a version 7.1 installation to identify the correct *MQ_INSTALLATION_PATH* for a queue manager:

   ```
   crtmqenv -m qmname
   ```

   This command returns the correct *MQ_INSTALLATION_PATH* value for the queue manager specified by *qmname*.
2. Run the monitoring script with the appropriate *qmname* and *MQ_INSTALLATION_PATH* parameters.

Note: PowerHA for AIX does not provide a way of supplying a parameter to the monitoring program for the queue manager. You must create a separate monitoring program for each queue manager, that encapsulates the queue manager name. Here is an example of a script used on AIX to encapsulate the queue manager name:

```
#!/bin/ksh
su mqm -c name_of_monitoring_script qmname MQ_INSTALLATION_PATH
```

where *MQ_INSTALLATION_PATH* is an optional parameter that specifies the path to the installation of IBM WebSphere MQ that the queue manager *qmname* is associated with.

The following script is not robust to the possibility that **runmqsc** hangs. Typically, HA clusters treat a hanging monitoring script as a failure and are themselves robust to this possibility.

The script does, however, tolerate the queue manager being in the starting state. This is because it is common for the HA cluster to start monitoring the queue manager as soon as it has started it. Some HA clusters distinguish between a starting phase and a running phase for resources, but it is necessary to configure the duration of the starting phase. Because the time taken to start a queue manager depends on the amount of work that it has to do, it is hard to choose a maximum time that starting a queue manager takes. If you choose a value that is too low, the HA cluster incorrectly assumes that the queue manager failed when it has not completed starting. This could result in an endless sequence of failovers.

This script must be run by the mqm user; it might therefore be necessary to wrap this script in a shell script to switch the user from the HA cluster user to mqm (an example shell script is provided in “Starting a queue manager under control of a high availability (HA) cluster” on page 315):

```
#!/bin/ksh
#
# This script tests the operation of the queue manager.
#
# An exit code is generated by the runmqsc command:
# 0  => Either the queue manager is starting or the queue manager is running and responds.
#       Either is OK.
# >0 => The queue manager is not responding and not starting.
#
# This script must be run by the mqm user.
```

```
QM=$1
MQ_INSTALLATION_PATH=$2

if [ -z "$QM" ]
then
  echo "ERROR! No queue manager name supplied"
  exit 1
fi

if [ -z "$MQ_INSTALLATION_PATH" ]
then
  # No path specified, assume system primary install or MQ level < 7.1.0.0
  echo "INFO: Using shell default value for MQ_INSTALLATION_PATH"
else
  echo "INFO: Prefixing shell PATH variable with $MQ_INSTALLATION_PATH/bin"
  PATH=$MQ_INSTALLATION_PATH/bin:$PATH
fi

# Test the operation of the queue manager. Result is 0 on success, non-zero on error.
echo "ping qmgr" | runmqsc ${QM} > /dev/null 2>&1
pingresult=$?

if [ $pingresult -eq 0 ]
then # ping succeeded

  echo "Queue manager '${QM}' is responsive"
  result=0

else # ping failed

  # Don't condemn the queue manager immediately, it might be starting.
  srchstr="( |-m)$QM *.*$"
  cnt=`ps -ef | tr "\t" " " | grep strmqm | grep "$srchstr" | grep -v grep \
               | awk '{print $2}' | wc -l`
  if [ $cnt -gt 0 ]
  then
    # It appears that the queue manager is still starting up, tolerate
    echo "Queue manager '${QM}' is starting"
    result=0
  else
    # There is no sign of the queue manager starting
    echo "Queue manager '${QM}' is not responsive"
    result=$pingresult
  fi

fi

exit $result
```

Putting the queue manager under control of the high availability (HA) cluster

You must configure the queue manager, under control of the HA cluster, with the queue manager's IP address and shared disks.

To define a resource group to contain the queue manager and all of its associated resources, complete the following steps:

1. Create the resource group containing the queue manager, the queue manager's volume or disk group, and the queue manager's IP address. The IP address is a virtual IP address, not the IP address of the computer.
2. Verify that the HA cluster correctly switches the resources between the cluster nodes and is ready to control the queue manager.

Deleting a queue manager from a high availability (HA) cluster node

You might want to remove a queue manager from a node that is no longer required to run the queue manager.

To remove the queue manager from a node in an HA cluster, complete the following steps:

1. Remove the node from the HA cluster so that the HA cluster will no longer attempt to activate the queue manager on this node.
2. Use the following **rmvmqinf** command to remove the queue manager's configuration information:

   `rmvmqinf` *qmgrname*

To completely delete the queue manager, use the **dltmqm** command. However, be aware that this completely deletes the queue manager's data and log files. When you have deleted the queue manager, you can use the **rmvmqinf** command to remove remaining configuration information from the other nodes.

Supporting the Microsoft Cluster Service (MSCS)

Introducing and setting up MSCS to support failover of virtual servers.

This information applies to WebSphere MQ for Windows only.

The Microsoft Cluster Service (MSCS) enables you to connect servers into a *cluster*, giving higher availability of data and applications, and making it easier to manage the system. MSCS can automatically detect and recover from server or application failures.

MSCS supports *failover* of *virtual servers*, which correspond to applications, Web sites, print queues, or file shares (including, for example, their disk spindles, files, and IP addresses).

Failover is the process by which MSCS detects a failure in an application on one computer in the cluster, and shuts down the disrupted application in an orderly manner, transfers its state data to the other computer, and reinitiates the application there.

This section introduces MSCS clusters and describes setting up MSCS support in the following sections:

- "Introducing MSCS clusters" on page 320
- "Setting up IBM WebSphere MQ for MSCS clustering" on page 321

Then tells you how to configure WebSphere MQ for MSCS clustering, in the following sections:

- "Creating a queue manager for use with MSCS" on page 323
- "Moving a queue manager to MSCS storage" on page 324
- "Putting a queue manager under MSCS control" on page 325
- "Removing a queue manager from MSCS control" on page 331

And then gives some useful hints on using MSCS with WebSphere MQ, and details the WebSphere MQ MSCS support utility programs, in the following sections:

- "Hints and tips on using MSCS" on page 332
- "IBM WebSphere MQ MSCS support utility programs" on page 335

Introducing MSCS clusters

MSCS clusters are groups of two or more computers, connected together and configured in such a way that, if one fails, MSCS performs a *failover*, transferring the state data of applications from the failing computer to another computer in the cluster and re-initiating their operation there.

"Using WebSphere MQ with high availability configurations" on page 310 contains a comparison between MSCS clusters, multi-instance queue managers, and WebSphere MQ clusters.

In this section and its subordinate topics, the term *cluster*, when used by itself, **always** means an MSCS cluster. This is distinct from a WebSphere MQ cluster described elsewhere in this guide.

A two-machine cluster comprises two computers (for example, A and B) that are jointly connected to a network for client access using a *virtual IP address*. They might also be connected to each other by one or more private networks. A and B share at least one disk for the server applications on each to use. There is also another shared disk, which must be a redundant array of independent disks (*RAID*) Level 1, for the exclusive use of MSCS; this is known as the *quorum* disk. MSCS monitors both computers to check that the hardware and software are running correctly.

In a simple setup such as this, both computers have all the applications installed on them, but only computer A runs with live applications; computer B is just running and waiting. If computer A encounters any one of a range of problems, MSCS shuts down the disrupted application in an orderly manner, transfers its state data to the other computer, and re-initiates the application there. This is known as

a *failover*. Applications can be made *cluster-aware* so that they interact fully with MSCS and failover gracefully.

A typical setup for a two-computer cluster is as shown in Figure 62 on page 321.

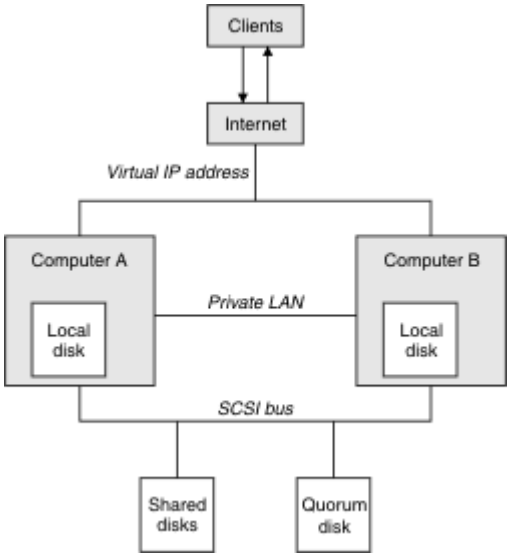


Figure 62. Two-computer MSCS cluster

Each computer can access the shared disk, but only one at a time, under the control of MSCS. In the event of a failover, MSCS switches the access to the other computer. The shared disk itself is usually a RAID, but need not be.

Each computer is connected to the external network for client access, and each has an IP address. However an external client, communicating with this cluster, is aware of only one *virtual IP address*, and MSCS routes the IP traffic within the cluster appropriately.

MSCS also performs its own communications between the two computers, either over one or more private connections or over the public network, for example to monitor their states using the heartbeat, and to synchronize their databases.

Setting up IBM WebSphere MQ for MSCS clustering

You configure IBM WebSphere MQ for clustering by making the queue manager the unit of failover to MSCS. You define a queue manager as a resource to MSCS, which can then monitor it, and transfer it to another computer in the cluster if there is a problem.

To set your system up for this, you start by installing IBM WebSphere MQ on each computer in the cluster.

As the queue manager is associated with the IBM WebSphere MQ installation name, the IBM WebSphere MQ installation name on all the computers in the cluster should be the same. See Installing and uninstalling.

The queue managers themselves need to exist only on the computer on which you create them. In the event of a failover, the MSCS initiates the queue managers on the other computer. The queue managers, however, must have their log and data files on a cluster shared disk, and not on a local drive. If you have a queue manager already installed on a local drive, you can migrate it using a tool provided with IBM WebSphere MQ; see “Moving a queue manager to MSCS storage” on page 324. If you want to create new queue managers for use with MSCS, see “Creating a queue manager for use with MSCS” on page 323.

After installation and migration, use the MSCS Cluster Administrator to make MSCS aware of your queue managers; see “Putting a queue manager under MSCS control” on page 325.

If you decide to remove a queue manager from MSCS control, use the procedure described in “Removing a queue manager from MSCS control” on page 331.

Setup symmetry
When an application switches from one node to the other it must behave in the same way, regardless of node. The best way of ensuring this is to make the environments identical.

If you can, set up a cluster with identical hardware, operating system software, product software, and configuration on each computer. In particular, ensure that all the required software installed on the two computers is identical in terms of version, maintenance level, SupportPacs, paths and exits, and that there is a common namespace (security environment) as described in "MSCS security" on page 322.

MSCS security
For successful MSCS security, follow these guidelines.

The guidelines are as follows:

- Make sure you that you have identical software installations on each computer in the cluster.
- Create a common namespace (security environment) across the cluster.
- Make the nodes of the MSCS cluster members of a domain, within which the user account that is the *cluster owner* is a domain account.
- Make the other user accounts on the cluster also domain accounts, so that they are available on both nodes. This is automatically the case if you already have a domain, and the accounts relevant to WebSphere MQ are domain accounts. If you do not currently have a domain, consider setting up a *mini-domain* to cater for the cluster nodes and relevant accounts. Your aim is to make your cluster of two computers look like a single computing resource.

  Remember that an account that is local to one computer does not exist on the other one. Even if you create an account with the same name on the other computer, its security identifier (SID) is different, so, when your application is moved to the other node, the permissions do not exist on that node.

During a failover or move, WebSphere MQ MSCS support ensures that all files that contain queue manager objects have equivalent permissions on the destination node. Explicitly, the code checks that the Administrators and mqm groups, and the SYSTEM account, have full control, and that if `Everyone` had read access on the old node, that permission is added on the destination node.

You can use a domain account to run your WebSphere MQ Service. Make sure that it exists in the local mqm group on each computer in the cluster.

Using multiple queue managers with MSCS
If you are running more than one queue manager on a computer, you can choose one of these setups.

The setups are as follows:

- All the queue managers in a single group. In this configuration, if a problem occurs with any queue manager, all the queue managers in the group failover to the other computer as a group.
- A single queue manager in each group. In this configuration, if a problem occurs with the queue manager, it alone fails over to the other computer without affecting the other queue managers.
- A mixture of the first two setups.

Cluster modes
There are two modes in which you might run a cluster system with WebSphere MQ: Active/Passive or Active/Active.

Note: If you are using MSCS together with the Microsoft Transaction Server (COM+), you cannot use Active/Active mode.

Active/Passive mode

In Active/Passive mode, computer A has the running application on it, and computer B is backup, only being used when MSCS detects a problem.

You can use this mode with only one shared disk, but, if any application causes a failover, **all** the applications must be transferred as a group (because only one computer can access the shared disk at a time).

You can configure MSCS with A as the *preferred* computer. Then, when computer A has been repaired or replaced and is working properly again, MSCS detects this and automatically switches the application back to computer A.

If you run more than one queue manager, consider having a separate shared disk for each. Then put each queue manager in a separate group in MSCS. In this way, any queue manager can failover to the other computer without affecting the other queue managers.

Active/Active mode

In Active/Active mode, computers A and B both have running applications, and the groups on each computer are set to use the other computer as backup. If a failure is detected on computer A, MSCS transfers the state data to computer B, and reinitiates the application there. computer B then runs its own application and A's.

For this setup you need at least two shared disks. You can configure MSCS with A as the preferred computer for A's applications, and B as the preferred computer for B's applications. After failover and repair, each application automatically ends up back on its own computer.

For WebSphere MQ this means that you could, for example, run two queue managers, one on each of A and B, with each exploiting the full power of its own computer. After a failure on computer A, both queue managers will run on computer B. This will mean sharing the power of the one computer, with a reduced ability to process large quantities of data at speed. However, your critical applications will still be available while you find and repair the fault on A.

Creating a queue manager for use with MSCS

This procedure ensures that a new queue manager is created in such a way that it is suitable for preparing and placing under MSCS control.

You start by creating the queue manager with all its resources on a local drive, and then migrate the log files and data files to a shared disk. (You can reverse this operation.) Do **not** attempt to create a queue manager with its resources on a shared drive.

You can create a queue manager for use with MSCS in two ways, either from a command prompt, or in the WebSphere MQ Explorer. The advantage of using a command prompt is that the queue manager is created *stopped* and set to *manual startup*, which is ready for MSCS. (The IBM WebSphere MQ Explorer automatically starts a new queue manager and sets it to automatic startup after creation. You have to change this.)

Creating a queue manager from a command prompt

Follow these steps to create a queue manager from a command prompt, for use with MSCS:

1. Ensure that you have the environment variable `MQSPREFIX` set to refer to a local drive, for example `C:\WebSphere MQ`. If you change this, reboot the machine so that the System account picks up the change. If you do not set the variable, the queue manager is created in the WebSphere MQ default directory for queue managers.
2. Create the queue manager using the **crtmqm** command. For example, to create a queue manager called `mscs_test` in the default directory, use:

   ```
   crtmqm mscs_test
   ```

3. Proceed to “Moving a queue manager to MSCS storage” on page 324.

Creating a queue manager using the WebSphere MQ Explorer

Follow these steps to create a queue manager using the IBM WebSphere MQ Explorer, for use with MSCS:

1. Start the IBM WebSphere MQ Explorer from the Start menu.
2. In the Navigator View, expand the tree nodes to find the `Queue Managers` tree node.
3. Right-click the `Queue Managers` tree node, and select `New->Queue Manager`. The Create Queue Manager panel is displayed.
4. Complete the dialog (Step 1), then click `Next>`.
5. Complete the dialog (Step 2), then click `Next>`.
6. Complete the dialog (Step 3), ensuring that `Start Queue Manager` and `Create Server Connection Channel` are not selected, then click `Next>`.
7. Complete the dialog (Step 4), then click `Finish`.
8. Proceed to "Moving a queue manager to MSCS storage" on page 324.

Moving a queue manager to MSCS storage

This procedure configures an existing queue manager to make it suitable for putting under MSCS control.

To achieve this, you move the log files and data files to shared disks to make them available to the other computer in the event of a failure. For example, the existing queue manager might have paths such as `C:\WebSphere MQ\log\<QMname>` and `C:\WebSphere MQ\qmgrs\<QMname>`. Do ***not*** try to move the files by hand; use the utility program supplied as part of WebSphere MQ MSCS Support as described in this topic.

If the queue manager being moved uses SSL connections and the SSL key repository is in the queue manager data directory on the local machine, then the key repository will be moved with the rest of the queue manager to the shared disk. By default, the queue manager attribute that specifies the SSL key repository location, SSLKEYR, is set to *MQ_INSTALLATION_PATH*`\qmgrs\QMGRNAME\ssl\key`, which is under the queue manager data directory. *MQ_INSTALLATION_PATH* represents the high-level directory in which WebSphere MQ is installed. The `hamvmqm` command does not modify this queue manager attribute. In this situation you must modify the queue manager attribute, SSLKEYR, using the IBM WebSphere MQ Explorer or the MQSC command `ALTER QMGR`, to point to the new SSL key repository file.

The procedure is as follows:

1. Shut down the queue manager, and check that there are no errors.
2. If the queue manager's log files or queue files are already stored on a shared disk, skip the rest of this procedure and proceed directly to "Putting a queue manager under MSCS control" on page 325.
3. Make a full media backup of the queue files and log files and store the backup in a safe place (see "Queue manager log files" on page 333 for why this is important).
4. If you already have a suitable shared disk resource proceed to step 6. Otherwise, using the MSCS Cluster Administrator to create a resource of type *shared disk* with sufficient capacity to store the queue manager log files and data (queue) files.
5. Test the shared disk by using the MSCS Cluster Administrator to move it from one cluster node to the other and back again.
6. Make sure that the shared disk is online on the cluster node where the queue manager log and data files are stored locally.
7. Run the utility program to move the queue manager as follows:

   ```
   hamvmqm /m qmname /dd "e:\
   WebSphere MQ" /ld "e:\
   WebSphere MQ\log"
   ```

   substituting your queue manager name for *qmname*, your shared disk drive letter for *e*, and your chosen directory for *WebSphere MQ*. The directories are created if they do not already exist.
8. Test the queue manager to ensure that it works, using the IBM WebSphere MQ Explorer. For example:

a. Right-click the queue manager tree node, then select `Start`. The queue manager starts.
b. Right-click the `Queues` tree node, then select `New->Local Queue...`, and give the queue a name.
c. Click `Finish`.
d. Right-click the queue, then select `Put Test Message...`. The Put Test Message panel is displayed.
e. Type some message text, then click `Put Test Message`, and close the panel.
f. Right-click the queue, then select `Browse Messages...`. The Message Browser panel is displayed.
g. Ensure your message is on the queue, then click `Close` . The Message Browser panel closes.
h. Right-click the queue, then select `Clear Messages...`. The messages on the queue are cleared.
i. Right-click the queue, then select `Delete...`. A confirmation panel is displayed, click OK. The queue is deleted.
j. Right-click the queue manager tree node, then select `Stop...`. The End Queue Manager panel is displayed.
k. Click OK. The queue manager stops.

9. As WebSphere MQ Administrator ensure that the startup attribute of the queue manager is set to manual. In the IBM WebSphere MQ Explorer, set the Startup field to `manual` in the queue manager properties panel.
10. Proceed to "Putting a queue manager under MSCS control" on page 325.

Putting a queue manager under MSCS control

The tasks involved in placing a queue manager under MSCS control, including prerequisite tasks.

Before you put a queue manager under MSCS control

Before you put a queue manager under MSCS control, perform the following tasks:

1. Ensure that IBM WebSphere MQ and its MSCS Support are installed on both machines in the cluster and that the software on each computer is identical, as described in "Setting up IBM WebSphere MQ for MSCS clustering" on page 321.
2. Use the **haregtyp** utility program to register WebSphere MQ as an MSCS resource type on all the cluster nodes. See "IBM WebSphere MQ MSCS support utility programs" on page 335 for additional information.
3. If you have not yet created the queue manager, see "Creating a queue manager for use with MSCS" on page 323.
4. If you have created the queue manager, or it already exists, ensure that you have carried out the procedure in "Moving a queue manager to MSCS storage" on page 324.
5. Stop the queue manager, if it is running, using either a command prompt or the IBM WebSphere MQ Explorer.
6. Test MSCS operation of the shared drives before going on to either of the following Windows procedures in this topic.

Windows Server 2012

Attention: MSCS support is delivered in WebSphere MQ 7.5 using a 32-bit DLL. Due to a restriction in Windows 2012, the IBM WebSphere MQ queue manager does not fail over following a restart.

Microsoft have deprecated the use of 32-bit DLLs with Windows 2012 and, therefore, no operating system fix is currently available for this issue. IBM does not provide a 64-bit library for IBM WebSphere MQ 7.5.

From IBM MQ 8.0 a 64-bit library is available, so you must use this version of the product for full MSCS functionality with Windows 2012, and later.

To place a queue manager under MSCS control on Windows Server 2012, use the following procedure:

1. Log in to the cluster node computer hosting the queue manager, or log in to a remote workstation as a user with cluster administration permissions, and connect to the cluster node hosting the queue manager.
2. Start the Failover Cluster Management tool.
3. Right-click **Failover Cluster Management > Connect Cluster ...** to open a connection to the cluster.
4. In contrast to the group scheme used in the MSCS Cluster Administrator on previous versions of Windows, the Failover Cluster Management tool uses the concept of services and applications. A configured service or application contains all the resources necessary for one application to be clustered. You can configure a queue manager under MSCS as follows:
   a. Right-click on the cluster and select **Configure Role** to start the configuration wizard.
   b. Select **Other Server** on the "Select Service or Application" panel.
   c. Select an appropriate IP address as a client access point.

      This address should be an unused IP address to be used by clients and other queue managers to connect to the *virtual* queue manager. This IP address is not the normal (static) address of either node; it is an additional address that *floats* between them. Although MSCS handles the routing of this address, it does **not** verify that the address can be reached.
   d. Assign a storage device for exclusive use by the queue manager. This device needs to be created as a resource instance before it can be assigned.

      You can use one drive to store both the logs and queue files, or you can split them up across drives. In either case, if each queue manager has its own shared disk, ensure that all drives used by this queue manager are exclusive to this queue manager, that is, that nothing else relies on the drives. Also ensure that you create a resource instance for every drive that the queue manager uses.

      The resource type for a drive depends on the SCSI support you are using; refer to your SCSI adapter instructions. There might already be groups and resources for each of the shared drives. If so, you do not need to create the resource instance for each drive. Move it from its current group to the one created for the queue manager.

      For each drive resource, set possible owners to both nodes. Set dependent resources to none.
   e. Select the **IBM MQSeries MSCS** resource on the "Select Resource Type" panel.
   f. Complete the remaining steps in the wizard.
5. Before bringing the resource online, the IBM MQSeries® MSCS resource needs additional configuration:
   a. Select the newly defined service which contains a resource called 'New IBM MQSeries MSCS'.
   b. Right-click **Properties** on the MQ resource.
   c. Configure the resource:
      - `Name`; choose a name that makes it easy to identify which queue manager it is for.
      - `Run in a separate Resource Monitor`; for better isolation
      - `Possible owners`; set both nodes
      - `Dependencies`; add the drive and IP address for this queue manager.

        Warning: Failure to add these dependencies means that IBM WebSphere MQ attempts to write the queue manager status to the wrong cluster disk during failovers. Because many processes might be attempting to write to this disk simultaneously, some IBM WebSphere MQ processes could be blocked from running.
      - `Parameters`; as follows:
        – `QueueManagerName` (required); the name of the queue manager that this resource is to control. This queue manager must exist on the local computer.

- PostOnlineCommand (optional); you can specify a program to run whenever the queue manager resource changes its state from offline to online. For more details see "PostOnlineCommand and PreOfflineCommand" on page 334.
- PreOfflineCommand (optional); you can specify a program to run whenever the queue manager resource changes its state from online to offline. For more details see "PostOnlineCommand and PreOfflineCommand" on page 334.

  Note: The *looksAlive* poll interval is set to default value of 5000 ms. The *isAlive* poll interval is set to default value of 60000 ms. These defaults can only be modified after the resource definition has been completed. For further details see "Summary of looksAlive and isAlive polling" on page 331.

d. Optionally, set a preferred node (but note the comments in "Using preferred nodes" on page 335)

e. The *Failover Policy* is set by default to sensible values, but you can tune the thresholds and periods that control *Resource Failover* and *Group Failover* to match the loads placed on the queue manager.

6. Test the queue manager by bringing it online in the MSCS Cluster Administrator and subjecting it to a test workload. If you are experimenting with a test queue manager, use the IBM WebSphere MQ Explorer. For example:

   a. Right-click the Queues tree node, then select New->Local Queue..., and give the queue a name.

   b. Click Finish. The queue is created, and displayed in the content view.

   c. Right-click the queue, then select Put Test Message.... The Put Test Message panel is displayed.

   d. Type some message text, then click Put Test Message, and close the panel.

   e. Right-click the queue, then select Browse Messages.... The Message Browser panel is displayed.

   f. Ensure that your message is on the queue, then click Close . The Message Browser panel closes.

   g. Right-click the queue, then select Clear Messages.... The messages on the queue are cleared.

   h. Right-click the queue, then select Delete.... A confirmation panel is displayed, click OK. The queue is deleted.

7. Test that the queue manager can be taken offline and back online using the MSCS Cluster Administrator.

8. Simulate a failover.

   In the MSCS Cluster Administrator, right-click the group containing the queue manager and select Move Group. This can take some minutes to do. (If at other times you want to move a queue manager to another node quickly, follow the procedure in "Moving a queue manager to MSCS storage" on page 324.) You can also right-click and select Initiate Failure; the action (local restart or failover) depends on the current state and the configuration settings.

Windows Server 2008

To place a queue manager under MSCS control on Windows Server 2008, use the following procedure:

1. Log in to the cluster node computer hosting the queue manager, or log in to a remote workstation as a user with cluster administration permissions, and connect to the cluster node hosting the queue manager.
2. Start the Failover Cluster Management tool.
3. Right-click **Failover Cluster Management > Manage a Cluster ...** to open a connection to the cluster.
4. In contrast to the group scheme used in the MSCS Cluster Administrator on previous versions of Windows, the Failover Cluster Management tool uses the concept of services and applications. A configured service or application contains all the resources necessary for one application to be clustered. You can configure a queue manager under MSCS as follows:

   a. Right-click **Services and Applications > Configure a Service or Application ...** to start the configuration wizard.

b. Select **Other Server** on the "Select Service or Application" panel.

c. Select an appropriate IP address as a client access point.

   This address should be an unused IP address to be used by clients and other queue managers to connect to the *virtual* queue manager. This IP address is not the normal (static) address of either node; it is an additional address that *floats* between them. Although MSCS handles the routing of this address, it does **not** verify that the address can be reached.

d. Assign a storage device for exclusive use by the queue manager. This device needs to be created as a resource instance before it can be assigned.

   You can use one drive to store both the logs and queue files, or you can split them up across drives. In either case, if each queue manager has its own shared disk, ensure that all drives used by this queue manager are exclusive to this queue manager, that is, that nothing else relies on the drives. Also ensure that you create a resource instance for every drive that the queue manager uses.

   The resource type for a drive depends on the SCSI support you are using; refer to your SCSI adapter instructions. There might already be groups and resources for each of the shared drives. If so, you do not need to create the resource instance for each drive. Move it from its current group to the one created for the queue manager.

   For each drive resource, set possible owners to both nodes. Set dependent resources to none.

e. Select the **IBM MQSeries MSCS** resource on the "Select Resource Type" panel.

f. Complete the remaining steps in the wizard.

5. Before bringing the resource online, the IBM MQSeries MSCS resource needs additional configuration:

   a. Select the newly defined service which contains a resource called 'New IBM MQSeries MSCS'.

   b. Right-click **Properties** on the MQ resource.

   c. Configure the resource:

      - `Name`; choose a name that makes it easy to identify which queue manager it is for.
      - `Run in a separate Resource Monitor`; for better isolation
      - `Possible owners`; set both nodes
      - `Dependencies`; add the drive and IP address for this queue manager.

        Warning: Failure to add these dependencies means that WebSphere MQ attempts to write the queue manager status to the wrong cluster disk during failovers. Because many processes might be attempting to write to this disk simultaneously, some IBM WebSphere MQ processes could be blocked from running.

      - `Parameters`; as follows:

        – `QueueManagerName` (required); the name of the queue manager that this resource is to control. This queue manager must exist on the local computer.

        – `PostOnlineCommand` (optional); you can specify a program to run whenever the queue manager resource changes its state from offline to online. For more details see “PostOnlineCommand and PreOfflineCommand” on page 334.

        – `PreOfflineCommand` (optional); you can specify a program to run whenever the queue manager resource changes its state from online to offline. For more details see “PostOnlineCommand and PreOfflineCommand” on page 334.

          Note: The *looksAlive* poll interval is set to default value of 5000 ms. The *isAlive* poll interval is set to default value of 60000 ms. These defaults can only be modified after the resource definition has been completed. For further details see “Summary of looksAlive and isAlive polling” on page 331.

   d. Optionally, set a preferred node (but note the comments in “Using preferred nodes” on page 335)

   e. The *Failover Policy* is set by default to sensible values, but you can tune the thresholds and periods that control *Resource Failover* and *Group Failover* to match the loads placed on the queue manager.

6. Test the queue manager by bringing it online in the MSCS Cluster Administrator and subjecting it to a test workload. If you are experimenting with a test queue manager, use the IBM WebSphere MQ Explorer. For example:
   a. Right-click the `Queues` tree node, then select `New->Local Queue...`, and give the queue a name.
   b. Click `Finish`. The queue is created, and displayed in the content view.
   c. Right-click the queue, then select `Put Test Message...`. The Put Test Message panel is displayed.
   d. Type some message text, then click `Put Test Message`, and close the panel.
   e. Right-click the queue, then select `Browse Messages...`. The Message Browser panel is displayed.
   f. Ensure that your message is on the queue, then click `Close` . The Message Browser panel closes.
   g. Right-click the queue, then select `Clear Messages...`. The messages on the queue are cleared.
   h. Right-click the queue, then select `Delete...`. A confirmation panel is displayed, click `OK`. The queue is deleted.
7. Test that the queue manager can be taken offline and back online using the MSCS Cluster Administrator.
8. Simulate a failover.

   In the MSCS Cluster Administrator, right-click the group containing the queue manager and select `Move Group`. This can take some minutes to do. (If at other times you want to move a queue manager to another node quickly, follow the procedure in "Moving a queue manager to MSCS storage" on page 324.) You can also right-click and select `Initiate Failure`; the action (local restart or failover) depends on the current state and the configuration settings.

Windows 2003

To place a queue manager under MSCS control on Windows 2003, use the following procedure:

1. Log in to the cluster node computer hosting the queue manager, or log in to a remote workstation as a user with cluster administration permissions, and connect to the cluster node hosting the queue manager.
2. Start the MSCS Cluster Administrator.
3. Open a connection to the cluster.
4. Create an MSCS group to be used to contain the resources for the queue manager. Name the group in such a way that it is obvious which queue manager it relates to. Each group can contain multiple queue managers, as described in "Using multiple queue managers with MSCS" on page 322.

   Use the group for all the remaining steps.
5. Create a resource instance for each of the SCSI logical drives that the queue manager uses.

   You can use one drive to store both the logs and queue files, or you can split them up across drives. In either case, if each queue manager has its own shared disk, ensure that all drives used by this queue manager are exclusive to this queue manager, that is, that nothing else relies on the drives. Also ensure that you create a resource instance for every drive that the queue manager uses.

   The resource type for a drive depends on the SCSI support you are using; refer to your SCSI adapter instructions. There might already be groups and resources for each of the shared drives. If so, you do not need to create the resource instance for each drive. Move it from its current group to the one created for the queue manager.

   For each drive resource, set possible owners to both nodes. Set dependent resources to none.
6. Create a resource instance for the IP address.

   Create an IP address resource (resource type *IP address*). This address should be an unused IP address to be used by clients and other queue managers to connect to the *virtual* queue manager. This IP address is not the normal (static) address of either node; it is an additional address that *floats*

between them. Although MSCS handles the routing of this address, it does **not** verify that the address can be reached.

7. Create a resource instance for the queue manager.

   Create a resource of type *IBM WebSphere MQ MSCS*. The wizard prompts you for various items, including the following:

   - `Name`; choose a name that makes it easy to identify which queue manager it is for.
   - `Add to group`; use the group that you created
   - `Run in a separate Resource Monitor`; for better isolation
   - `Possible owners`; set both nodes
   - `Dependencies`; add the drive and IP address for this queue manager.

     Warning: Failure to add these dependencies means that WebSphere MQ attempts to write the queue manager status to the wrong cluster disk during failovers. Because many processes might be attempting to write to this disk simultaneously, some IBM WebSphere MQ processes could be blocked from running.
   - `Parameters`; as follows:
     - `QueueManagerName` (required); the name of the queue manager that this resource is to control. This queue manager must exist on the local computer.
     - `PostOnlineCommand` (optional); you can specify a program to run whenever the queue manager resource changes its state from offline to online. For more details see "PostOnlineCommand and PreOfflineCommand" on page 334.
     - `PreOfflineCommand` (optional); you can specify a program to run whenever the queue manager resource changes its state from online to offline. For more details see "PostOnlineCommand and PreOfflineCommand" on page 334.

       Note: The *looksAlive* poll interval is set to default value of 5000 ms. The *isAlive* poll interval is set to default value of 30000 ms. These defaults can only be modified after the resource definition has been completed. For further details see "Summary of looksAlive and isAlive polling" on page 331.
8. Optionally, set a preferred node (but note the comments in "Using preferred nodes" on page 335)
9. The *Failover Policy* (as defined in the properties for the group) is set by default to sensible values, but you can tune the thresholds and periods that control *Resource Failover* and *Group Failover* to match the loads placed on the queue manager.
10. Test the queue manager by bringing it online in the MSCS Cluster Administrator and subjecting it to a test workload. If you are experimenting with a test queue manager, use the IBM WebSphere MQ Explorer. For example:
    a. Right-click the `Queues` tree node, then select `New->Local Queue...`, and give the queue a name.
    b. Click `Finish`. The queue is created, and displayed in the content view.
    c. Right-click the queue, then select `Put Test Message...`. The Put Test Message panel is displayed.
    d. Type some message text, then click `Put Test Message`, and close the panel.
    e. Right-click the queue, then select `Browse Messages...`. The Message Browser panel is displayed.
    f. Ensure that your message is on the queue, then click `Close` . The Message Browser panel closes.
    g. Right-click the queue, then select `Clear Messages...`. The messages on the queue are cleared.
    h. Right-click the queue, then select `Delete...`. A confirmation panel is displayed, click `OK`. The queue is deleted.
11. Test that the queue manager can be taken offline and back online using the MSCS Cluster Administrator.

12. Simulate a failover.

    In the MSCS Cluster Administrator, right-click the group containing the queue manager and select `Move Group`. This can take some minutes to do. (If at other times you want to move a queue manager to another node quickly, follow the procedure in "Moving a queue manager to MSCS storage" on page 324.) You can also right-click and select `Initiate Failure`; the action (local restart or failover) depends on the current state and the configuration settings.

Summary of looksAlive and isAlive polling

looksAlive and *isAlive* are intervals at which MSCS calls back into the resource types supplied library code and requests that the resource performs checks to determine the working status of itself. This ultimately determines if MSCS attempts to fail over the resource.

On every occasion that the *looksAlive* interval elapses (default 5000 ms), the queue manager resource is called to perform its own check to determine if its status is satisfactory.

On every occasion that the *isAlive* interval elapses (default 30000 ms), another call is made to the queue manager resource for it to perform another check to determine if the resource is functioning correctly. This enables two levels of resource type checking.

1. A *looksAlive* status check to establish if the resource appears to be functioning.
2. A more significant *isAlive* check that determines if the queue manager resource is active.

If the queue manager resource is determined not to be active, MSCS, based on other advanced MSCS options, triggers a fail over for the resource and associated dependant resources to another node in the cluster. For further information, see MSCS documentation.

Removing a queue manager from MSCS control

You can remove queue managers from MSCS control, and return them to manual administration.

You do not need to remove queue managers from MSCS control for maintenance operations. You can do that by taking a queue manager offline temporarily, using the MSCS Cluster Administrator. Removing a queue manager from MSCS control is a more permanent change; only do it if you decide that you no longer want MSCS to have any further control of the queue manager.

If the queue manager being removed uses SSL connections you must modify the queue manager attribute, SSLKEYR, using the WebSphere MQ Explorer or the MQSC command `ALTER QMGR`, to point to the SSL key repository file on the local directory.

The procedure is:

1. Take the queue manager resource offline using the MSCS Cluster Administrator, as described in "Taking a queue manager offline from MSCS" on page 331
2. Destroy the resource instance. This does not destroy the queue manager.
3. Optionally, migrate the queue manager files back from shared drives to local drives. To do this, see "Returning a queue manager from MSCS storage" on page 332.
4. Test the queue manager.

Taking a queue manager offline from MSCS

To take a queue manager offline from MSCS, perform the following steps:

1. Start the MSCS Cluster Administrator.
2. Open a connection to the cluster.
3. Select `Groups`, and open the group containing the queue manager to be moved.
4. Select the queue manager resource.
5. Right-click it and select `Offline`.
6. Wait for completion.

Returning a queue manager from MSCS storage

This procedure configures the queue manager to be back on its computer's local drive, that is, it becomes a *normal* WebSphere MQ queue manager. To achieve this, you move the log files and data files from the shared disks. For example, the existing queue manager might have paths such as `E:\WebSphere MQ\log\<QMname>` and `E:\WebSphere MQ\qmgrs\<QMname>`. Do not try to move the files by hand; use the **hamvmqm** utility program supplied as part of WebSphere MQ MSCS Support:

1. Shut down the queue manager, and check that there are no errors.
2. Make a full media backup of the queue files and log files and store the backup in a safe place (see "Queue manager log files" on page 333 for why this is important).
3. Decide which local drive to use and ensure that it has sufficient capacity to store the queue manager log files and data (queue) files.
4. Make sure that the shared disk on which the files currently reside is online on the cluster node to which to move the queue manager log and data files.
5. Run the utility program to move the queue manager as follows:

   ```
   hamvmqm /m qmname /dd "c:\
   WebSphere MQ" /ld "c:\
   WebSphere MQ\log"
   ```

   substituting your queue manager name for *qmname*, your local disk drive letter for *c*, and your chosen directory for *WebSphere MQ* (the directories are created if they do not already exist).
6. Test the queue manager to ensure that it works (as described in "Moving a queue manager to MSCS storage" on page 324).

Hints and tips on using MSCS

This section contains some general information to help you use WebSphere MQ support for MSCS effectively.

This section contains some general information to help you use WebSphere MQ support for MSCS effectively.

How long does it take to fail a queue manager over from one machine to the other? This depends heavily on the amount of workload on the queue manager and on the mix of traffic, for example, how much of it is persistent, within sync point, and how much committed before the failure. IBM tests have given failover and failback times of about a minute. This was on a very lightly loaded queue manager and actual times will vary considerably depending on load.

Verifying that MSCS is working

Follow these steps to ensure that you have a running MSCS cluster.

The task descriptions starting with "Creating a queue manager for use with MSCS" on page 323 assume that you have a running MSCS cluster within which you can create, migrate, and destroy resources. If you want to make sure that you have such a cluster:

1. Using the MSCS Cluster Administrator, create a group.
2. Within that group, create an instance of a generic application resource, specifying the system clock (path name `C:\winnt\system32\clock.exe` and working directory of `C:\`).
3. Make sure that you can bring the resource online, that you can move the group that contains it to the other node, and that you can take the resource offline.

Manual startup
For a queue manager managed by MSCS, you ***must*** set the startup attribute to manual. This ensures that the WebSphere MQ MSCS support can restart the IBM MQSeries Service without immediately starting the queue manager.

The WebSphere MQ MSCS support needs to be able to restart the service so that it can perform monitoring and control, but must itself remain in control of which queue managers are running, and on which machines. See “Moving a queue manager to MSCS storage” on page 324 for more information.

MSCS and queue managers
Considerations concerning queue managers when using MSCS.

Creating a matching queue manager on the other node

For clustering to work with WebSphere MQ, you need an identical queue manager on node B for each one on node A. However, you do not need to explicitly create the second one. You can create or prepare a queue manager on one node, move it to the other node as described in “Moving a queue manager to MSCS storage” on page 324, and it is fully duplicated on that node.

Default queue managers

Do not use a default queue manager under MSCS control. A queue manager does not have a property that makes it the default; WebSphere MQ keeps its own separate record. If you move a queue manager set to be the default to the other computer on failover, it does not become the default there. Make all your applications refer to specific queue managers by name.

Deleting a queue manager

Once a queue manager has moved node, its details exist in the registry on both computers. When you want to delete it, do so as normal on one computer, and then run the utility described in “IBM WebSphere MQ MSCS support utility programs” on page 335 to clean up the registry on the other computer.

Support for existing queue managers

You can put an existing queue manager under MSCS control, provided that you can put your queue manager log files and queue files on a disk that is on the shared SCSI bus between the two machines (see Figure 62 on page 321). You need to take the queue manager offline briefly while the MSCS Resource is created.

If you want to create a new queue manager, create it independently of MSCS, test it, then put it under MSCS control. See:

- “Creating a queue manager for use with MSCS” on page 323
- “Moving a queue manager to MSCS storage” on page 324
- “Putting a queue manager under MSCS control” on page 325

Telling MSCS which queue managers to manage

You choose which queue managers are placed under MSCS control by using the MSCS Cluster Administrator to create a resource instance for each such queue manager. This process presents you with a list of resources from which to select the queue manager that you want that instance to manage.

Queue manager log files

When you move a queue manager to MSCS storage, you move its log and data files to a shared disk (for an example see “Moving a queue manager to MSCS storage” on page 324).

It is advisable before you move, to shut the queue manager cleanly and take a full backup of the data files and log files.

Multiple queue managers

WebSphere MQ MSCS support allows you to run multiple queue managers on each machine and to place individual queue managers under MSCS control.

Always use MSCS to manage clusters
Do not try to perform start and stop operations directly on any queue manager under the control of MSCS, using either the control commands or the IBM WebSphere MQ Explorer. Instead, use MSCS Cluster Administrator to bring the queue manager online or take it offline.

Using the MSCS Cluster Administrator is partly to prevent possible confusion caused by MSCS reporting that the queue manager is offline, when in fact you have started it outside the control of MSCS. More seriously, stopping a queue manager without using MSCS is detected by MSCS as a failure, initiating failover to the other node.

Working in Active/Active mode
Both computers in the MSCS cluster can run queue managers in Active/Active mode. You do not need to have a completely idle machine acting as standby (but you can, if you want, in Active/Passive Mode).

If you plan to use both machines to run workload, provide each with sufficient capacity (processor, memory, secondary storage) to run the entire cluster workload at a satisfactory level of performance.

Note: If you are using MSCS together with Microsoft Transaction Server (COM+), you **cannot** use Active/Active mode. This is because, to use WebSphere MQ with MSCS and COM+:

- Application components that use WebSphere MQ's COM+ support must run on the same computer as the Distributed Transaction Coordinator (DTC), a part of COM+.
- The queue manager must also run on the same computer.
- The DTC must be configured as an MSCS resource, and can therefore run on only one of the computers in the cluster at any time.

PostOnlineCommand and PreOfflineCommand
Use these commands to integrate WebSphere MQ MSCS support with other systems. You can use them to issue WebSphere MQ commands, wih some restrictions.

Specify these commands in the Parameters to a resource of type `IBM WebSphere MQ MSCS`. You can use them to integrate WebSphere MQ MSCS support with other systems or procedures. For example, you could specify the name of a program that sends a mail message, activates a pager, or generates some other form of alert to be captured by another monitoring system.

PostOnlineCommand is invoked when the resource changes from offline to online; PreOfflineCommand is invoked for a change from online to offline. When invoked these commands are run, by default, from the Windows system directory. Because WebSphere MQ uses a 32-bit resource monitor process, on Windows 64-bit systems, this is the `\Windows\SysWOW64` directory rather than `\Windows\system32`. For more information, see the Microsoft documentation about file redirection in a Windows x64 environment.
Both commands run under the user account used to run the MSCS Cluster Service; and are invoked asynchronously; WebSphere MQ MSCS support does not wait for them to complete before continuing. This eliminates any risk that they might block or delay further cluster operations.

You can also use these commands to issue WebSphere MQ commands, for example to restart Requester channels. However, the commands are run at the point in time when the queue manager's state changes so they are not intended to perform long-running functions and must not make assumptions about the current state of the queue manager; it is quite possible that, immediately after the queue manager was brought online, an administrator issued an offline command.

If you want to run programs that depend on the state of the queue manager, consider creating instances of the `MSCS Generic Application` resource type, placing them in the same MSCS group as the queue manager resource, and making them dependent on the queue manager resource.

Using preferred nodes
It can be useful when using Active/Active mode to configure a *preferred node* for each queue manager. However, in general it is better not to set a preferred node but to rely on a manual failback.

Unlike some other relatively stateless resources, a queue manager can take a while to fail over (or back) from one node to the other. To avoid unnecessary outages, test the recovered node before failing a queue manager back to it. This precludes use of the `immediate` failback setting. You can configure failback to occur between certain times of day.

Probably the safest route is to move the queue manager back manually to the required node, when you are certain that the node is fully recovered. This precludes use of the `preferred node` option.

If COM+ errors occur in the Application Event log
When you install WebSphere MQ on a newly-installed MSCS cluster, you might find an error with Source COM+ and Event ID 4691 reported in the Application Event log.

This means that you are trying to run WebSphere MQ on a Microsoft Cluster Server (MSCS) environment when the Microsoft Distributed Transaction Coordinator (MSDTC) has not been configured to run in such an environment. For information on configuring MSDTC in a clustered environment, refer to Microsoft documentation.

IBM WebSphere MQ MSCS support utility programs
A list of the IBM WebSphere MQ support for MSCS utility programs that you can run at a command prompt.

IBM WebSphere MQ support for MSCS includes the following utility programs:

Register/unregister the resource type
`haregtyp.exe`

After you *unregister* the IBM WebSphere MQ MSCS resource type you can no longer create any resources of that type. MSCS does not let you unregister a resource type if you still have instances of that type within the cluster:

1. Using the MSCS Cluster Administrator, stop any queue managers that are running under MSCS control, by taking them offline as described in "Taking a queue manager offline from MSCS" on page 331.
2. Using the MSCS Cluster Administrator, delete the resource instances.
3. At a command prompt, unregister the resource type by entering the following command:

```
haregtyp /u
```

If you want to *register* the type (or re-register it at a later time), enter the following command at a command prompt:

```
haregtyp /r
```

After successfully registering the MSCS libraries, you must reboot the system if you have not done so since installing IBM WebSphere MQ.

Move a queue manager to MSCS storage
`hamvmqm.exe`

See "Moving a queue manager to MSCS storage" on page 324.

Delete a queue manager from a node
`hadltmqm.exe`

Consider the case where you have had a queue manager in your cluster, it has been moved from one node to another, and now you want to destroy it. Use the IBM WebSphere MQ Explorer to delete it on

the node where it currently is. The registry entries for it still exist on the other computer. To delete these, enter the following command at a prompt on that computer:

```
hadltmqm /m qmname
```

where qmname is the name of the queue manager to remove.

Check and save setup details

amqmsysn.exe

This utility presents a dialog showing full details of your IBM WebSphere MQ MSCS Support setup, such as might be requested if you call IBM support. There is an option to save the details to a file.

Multi-instance queue managers

Multi-instance queue managers are instances of the same queue manager configured on different servers. One instance of the queue manager is defined as the active instance and another instance is defined as the standby instance. If the active instance fails, the multi-instance queue manager restarts automatically on the standby server.

Figure 63 on page 336 shows a multi-instance configuration for QM1. IBM WebSphere MQ is installed on two servers, one of which is a spare. One queue manager, QM1, has been created. One instance of QM1 is active, and is running on one server. The other instance of QM1 is running in standby on the other server, doing no active processing, but ready to take over from the active instance of QM1, if the active instance fails.

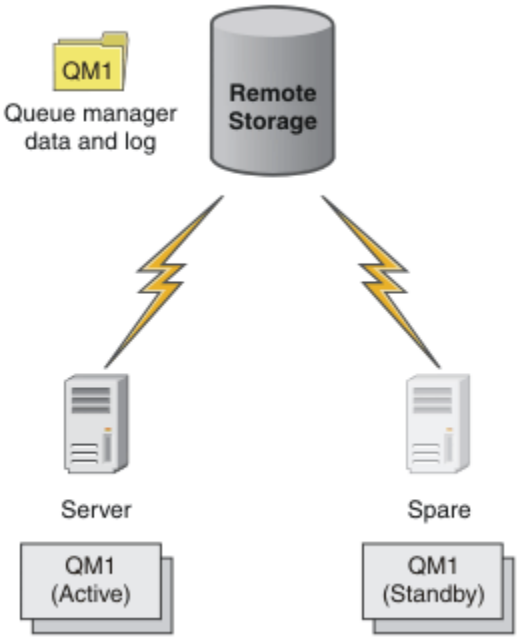


Figure 63. Multi-instance queue manager

When you intend to use a queue manager as a multi-instance queue manager, create a single queue manager on one of the servers using the **crtmqm** command, placing its queue manager data and logs in shared network storage. On the other server, rather than create the queue manager again, use the **addmqinf** command to create a reference to the queue manager data and logs on the network storage.

You can now run the queue manager from either of the servers. Each of the servers references the same queue manager data and logs; there is only one queue manager, and it is active on only one server at a time.

The queue manager can run either as a single instance queue manager, or as a multi-instance queue manager. In both cases only one instance of the queue manager is running, processing requests. The

difference is that when running as a multi-instance queue manager, the server that is not running the active instance of the queue manager runs as a standby instance, ready to take over from the active instance automatically if the active server fails.

The only control you have over which instance becomes active first is the order in which you start the queue manager on the two servers. The first instance to acquire read/write locks to the queue manager data becomes the active instance.

You can swap the active instance to the other server, once it has started, by stopping the active instance using the switchover option to transfer control to the standby.

The active instance of QM1 has exclusive access to the shared queue manager data and logs folders when it is running. The standby instance of QM1 detects when the active instance has failed, and becomes the active instance. It takes over the QM1 data and logs in the state they were left by the active instance, and accepts reconnections from clients and channels.

The active instance might fail for various reasons that result in the standby taking over:

- Failure of the server hosting the active queue manager instance.
- Failure of connectivity between the server hosting the active queue manager instance and the file system.
- Unresponsiveness of queue manager processes, detected by WebSphere MQ, which then shuts down the queue manager.

You can add the queue manager configuration information to multiple servers, and choose any two servers to run as the active/standby pair. There is a limit of a total of two instances. You cannot have two standby instances and one active instance.

A multi-instance queue manager is one part of a high availability solution. You need some additional components to build a useful high availability solution.

- Client and channel reconnection to transfer WebSphere MQ connections to the computer that takes over running the active queue manager instance.
- A high performance shared network file system (NFS) that manages locks correctly and provides protection against media and file server failure.

  Important: You must stop all multi-instance queue manager instances that are running in your environment before you can perform maintenance on the NFS drive. Make sure that you have queue manager configuration backups to recover, in the event of an NFS failure.
- Resilient networks and power supplies to eliminate single points of failure in the basic infrastructure.
- Applications that tolerate failover. In particular you need to pay close attention to the behavior of transactional applications, and to applications that browse WebSphere MQ queues.
- Monitoring and management of the active and standby instances to ensure that they are running, and to restart active instances that have failed. Although multi-instance queue managers restart automatically, you need to be sure that your standby instances are running, ready to take over, and that failed instances are brought back online as new standby instances.

WebSphere MQ MQI clients and channels reconnect automatically to the standby queue manager when it becomes active. More information about reconnection, and the other components in a high availability solution can be found in related topics. Automatic client reconnect is not supported by IBM WebSphere MQ classes for Java.

Supported platforms

You can create a multi-instance queue manager on any of the non-z/OS platforms from version 7.0.1.

Automatic client reconnection is supported for MQI clients from version 7.0.1 onwards.

Create a multi-instance queue manager

Create a multi-instance queue manager, creating the queue manager on one server, and configuring IBM WebSphere MQ on another server. Multi-instance queue managers shared queue manager data and logs.

Most of the effort involved in creating a multi-instance queue manager is the task of setting up the shared queue manager data and log files. You must create shared directories on network storage, and make the directories available to other servers using network shares. These tasks need to be performed by someone with administrative authority, such as *root* on UNIX and Linux systems. The steps are as follows:

1. Create the shares for the data and log files.
2. Create the queue manager on one server.
3. Run the command **dspmqinf** on the first server to collect the queue manager configuration data and copy it into the clipboard.
4. Run the command **addmqinf** with the copied data to create the queue manager configuration on the second server.

You do not run **crtmqm** to create the queue manager again on the second server.

File access control

You need to take care that the user and group `mqm` on all other servers have permission to access the shares.

On UNIX and Linux, you need to make the `uid` and `gid` of `mqm` the same on all the systems. You might need to edit `/etc/passwd` on each system to set a common `uid` and `gid` for `mqm`, and then reboot your system.

On Microsoft Windows, the user ID that is running the queue manager processes must have full control permission to the directories containing the queue manager data and log files. You can configure the permission in two ways:

1. Create a queue manager with a global group as the alternative security principal. Authorize the global group to have full control access to the directories containing queue manager data and log files; see “Secure shared queue manager data and log directories and files on Windows” on page 363. Make the user ID that is running the queue manager a member of the global group. You cannot make a local user a member of a global group, so the queue manager processes must run under a domain user ID. The domain user ID must be a member of the local group `mqm`. The task, “Create a multi-instance queue manager on domain workstations or servers” on page 340, demonstrates how to set up a multi-instance queue manager using files secured in this way.
2. Create a queue manager on the domain controller, so that the local `mqm` group has domain scope, "domain local". Secure the file share with the domain local `mqm`, and run queue manager processes on all instances of a queue manager under the same domain local `mqm` group. The task, “Create a multi-instance queue manager on domain controllers” on page 354, demonstrates how to set up a multi-instance queue manager using files secured in this way.

Configuration information

Configure as many queue manager instances as you need by modifying the IBM WebSphere MQ queue manager configuration information about each server. Each server must have the same version of IBM WebSphere MQ installed at a compatible fix level. The commands, **dspmqinf** and **addmqinf** assist you to configure the additional queue manager instances. Alternatively, you can edit the `mqs.ini` and `qm.ini` files directly. The topics, “Create a multi-instance queue manager on Linux” on page 374, “Create a multi-instance queue manager on domain workstations or servers” on page 340, and “Create a multi-instance queue manager on domain controllers” on page 354 are examples showing how to configure a multi-instance queue manager.

On Windows, UNIX and Linux systems, you can share a single `mqs.ini` file by placing it on the network share and setting the **AMQ_MQS_INI_LOCATION** environment variable to point to it.

Restrictions

1. Configure multiple instances of the same queue manager only on servers having the same operating system, architecture and endianness. For example, both machines must be either 32-bit or 64-bit.
2. All IBM WebSphere MQ installations must be at release level 7.0.1 or higher.
3. Typically, active and standby installations are maintained at the same maintenance level. Consult the maintenance instructions for each upgrade to check whether you must upgrade all installations together.

   Note that the maintenance levels for the active and passive queue managers must be identical.
4. Share queue manager data and logs only between queue managers that are configured with the same IBM WebSphere MQ user, group, and access control mechanism.
5. On UNIX and Linux systems, configure the shared file system on networked storage with a `hard`, interruptible, mount rather than a `soft` mount. A hard interruptible mount forces the queue manager to hang until it is interrupted by a system call. Soft mounts do not guarantee data consistency after a server failure.
6. The shared log and data directories cannot be stored on a FAT, or an NFSv3 file system. For multi-instance queue managers on Windows, the networked storage must be accessed by the Common Internet File System (CIFS) protocol used by Windows networks.

Windows domains and multi-instance queue managers
A multi-instance queue manager on Windows requires its data and logs to be shared. The share must be accessible to all instances of the queue manager running on different servers or workstations. Configure the queue managers and share as part of a Windows domain. The queue manager can run on a domain workstation or server, or on the domain controller.

Before configuring a multi-instance queue manager, read "Secure unshared queue manager data and log directories and files on Windows" on page 366 and "Secure shared queue manager data and log directories and files on Windows" on page 363 to review how to control access to queue manager data and log files. The topics are educational; if you want to go directly to setting up shared directories for a multi-instance queue manager in a Windows domain; see "Create a multi-instance queue manager on domain workstations or servers" on page 340.

Run a multi-instance queue manager on domain workstations or servers

From Version 7.1, multi-instance queue managers run on a workstation or server that is a member of a domain. Before Version 7.1, multi-instance queue managers ran only on domain controllers; see "Run a multi-instance queue manager on domain controllers" on page 340. To run a multi-instance queue manager on Windows, you require a domain controller, a file server, and two workstations or servers running the same queue manager connected to the same domain.

The change that makes it possible to run a multi-instance queue manager on any server or workstation in a domain, is that you can now create a queue manager with an additional security group. The additional security group is passed in the **crtmqm** command, in the `-a` parameter. You secure the directories that contain the queue manager data and logs with the group. The user ID that runs queue manager processes must be a member of this group. When the queue manager accesses the directories, Windows checks the permissions the user ID has to access the directories. By giving both the group and the user ID domain scope, the user ID running the queue manager processes has credentials from the global group. When the queue manager is running on a different server, the user ID running the queue manager processes can have the same credentials. The user ID does not have to be the same. It has to be a member of the alternative security group, as well as a member of the local `mqm` group.

The task of creating a multi-instance queue manager is the same as in Version 7.0.1 with one change. You must add the additional security group name to the parameters of the **crtmqm** command. The task is described in "Create a multi-instance queue manager on domain workstations or servers" on page 340.

Multiple steps are required to configure the domain, and the domain servers and workstations. You must understand how Windows authorizes access by a queue manager to its data and log directories. If you are not sure how queue manager processes are authorized to access their log and data files read the topic

"Secure unshared queue manager data and log directories and files on Windows" on page 366. The topic includes two tasks to help you understand the steps the required. The tasks are "Reading and writing data and log files authorized by the local mqm group" on page 368 and "Reading and writing data and log files authorized by an alternative local security group" on page 371. Another topic, "Secure shared queue manager data and log directories and files on Windows" on page 363, explains how to secure shared directories containing queue manager data and log files with the alternative security group. The topic includes four tasks, to set up a Windows domain, create a file share, install IBM WebSphere MQ for Windows, and configure a queue manager to use the share. The tasks are as follows:

1. "Creating an Active Directory and DNS domain for IBM WebSphere MQ" on page 343.
2. "Installing IBM WebSphere MQ on a server or workstation in a Windows domain" on page 346.
3. "Creating a shared directory for queue manager data and log files" on page 349.
4. "Reading and writing shared data and log files authorized by an alternative global security group" on page 351.

You can then do the task, "Create a multi-instance queue manager on domain workstations or servers" on page 340, using the domain. Do these tasks to explore setting up a multi-instance queue manager before transferring your knowledge to a production domain.

Run a multi-instance queue manager on domain controllers

In Version 7.0.1, multi-instance queue managers ran only on domain controllers. Queue manager data could be secured with the domain `mqm` group. As the topic "Secure shared queue manager data and log directories and files on Windows" on page 363 explains, you cannot share directories secured with the local `mqm` group on workstations or servers. However on domain controllers all group and principals have domain scope. If you install IBM WebSphere MQ for Windows on a domain controller, the queue manager data and log files are secured with the domain `mqm` group, which can be shared. Follow the steps in the task, "Create a multi-instance queue manager on domain controllers" on page 354 to configure a multi-instance queue manager on domain controllers.

Related information

Windows 2000, Windows Server 2003, and Windows Server 2008 cluster nodes as domain controllers

Create a multi-instance queue manager on domain workstations or servers

An example shows how to set up a multi-instance queue manager on Windows on a workstation or a server that is part of a Windows domain. The server does not have to be a domain controller. The setup demonstrates the concepts involved, rather than being production scale. The example is based on Windows Server 2008. The steps might differ on other versions of Windows Server.

In a production scale configuration, you might have to tailor the configuration to an existing domain. For example, you might define different domain groups to authorize different shares, and to group the user IDs that run queue managers.

The example configuration consists of three servers:

sun
: A Windows Server 2008 domain controller. It owns the *wmq.example.com* domain that contains *Sun*, *mars*, and *venus*. For the purposes of illustration, it is also used as the file server.

mars
: A Windows Server 2008 used as the first IBM WebSphere MQ server. It contains one instance of the multi-instance queue manager called *QMGR*.

venus
: A Windows Server 2008 used as the second IBM WebSphere MQ server. It contains the second instance of the multi-instance queue manager called *QMGR*.

Replace the italicized names in the example, with names of your choosing.

Before you begin

On Windows, you do not need to verify the file system that you plan to store queue manager data and log files on. The checking procedure, Verifying shared file system behavior , is applicable to UNIX and Linux. On Windows, the checks are always successful.

Do the steps in the following tasks. The tasks create the domain controller and domain, install IBM WebSphere MQ for Windows on one server, and create the file share for data and log files. If you are configuring an existing domain controller, you might find it useful to try out the steps on a new Windows Server 2008. You can adapt the steps to your domain.

1. "Creating an Active Directory and DNS domain for IBM WebSphere MQ" on page 343.
2. "Installing IBM WebSphere MQ on a server or workstation in a Windows domain" on page 346.
3. "Creating a shared directory for queue manager data and log files" on page 349.
4. "Reading and writing shared data and log files authorized by an alternative global security group" on page 351.

About this task

This task is one of a sequence of tasks to configure a domain controller and two servers in the domain to run instances of a queue manager. In this task you configure a second server, *venus*, to run another instance of the queue manager *QMGR*. Follow the steps in this task to create the second instance of the queue manager, *QMGR*, and test that it works.

This task is separate from the four tasks in the preceding section. It contains the steps that convert a single instance queue manager into a multi-instance queue manager. All the other steps are common to single or multi-instance queue managers.

Procedure

1. Configure a second server to run IBM WebSphere MQ for Windows.
   a) Do the steps in the task "Installing IBM WebSphere MQ on a server or workstation in a Windows domain" on page 346 to create a second domain server. In this sequence of tasks the second server is called *venus*.

      Tip: Create the second installation using the same installation defaults for IBM WebSphere MQ on each of the two servers. If the defaults differ, you might have to tailor the `Prefix` and the `InstallationName` variables in the *QMGR* **QueueManager** stanza in the IBM WebSphere MQ configuration file `mqs.ini`. The variables refer to paths that can differ for each installation and queue manager on each server. If the paths remain the same on every server, it is simpler to configure a multi-instance queue manager.
2. Create a second instance of *QMGR* on *venus*.
   a) If *QMGR* on *mars* does not exist, do the task "Reading and writing shared data and log files authorized by an alternative global security group" on page 351, to create it
   b) Check the values of the `Prefix` and `InstallationName` parameters are correct for *venus*.

      On *mars*, run the **dspmqinf** command:

      ```
      dspmqinf QMGR
      ```

      The system response:

      ```
      QueueManager:
          Name=QMGR
          Directory=QMGR
          Prefix=C:\Program Files\IBM\WebSphere MQ
          DataPath=\\sun\wmq\data\QMGR
          InstallationName=Installation1
      ```

c) Copy the machine-readable form of the **QueueManager** stanza to the clipboard.

On *mars* run the **dspmqinf** command again, with the `-o command` parameter.

```
dspmqinf -o command QMGR
```

The system response:

```
addmqinf -s QueueManager -v Name=QMGR
-v Directory=QMGR -v Prefix="C:\Program Files\IBM\WebSphere MQ"
-v DataPath=\\sun\wmq\data\QMGR
```

d) On *venus* run the **addmqinf** command from the clipboard to create an instance of the queue manager on *venus*.

Adjust the command, if necessary, to accommodate differences in the `Prefix` or `InstallationName` parameters.

```
addmqinf -s QueueManager -v Name=QMGR
-v Directory=QMGR -v Prefix="C:\Program Files\IBM\WebSphere MQ"
-v DataPath=\\sun\wmq\data\QMGR
```

```
WebSphere MQ configuration information added.
```

3. Start the queue manager *QMGR* on *venus*, permitting standby instances.

a) Check *QMGR* on *mars* is stopped.

On *mars*, run the **dspmq** command:

```
dspmq -m QMGR
```

The system response depends on how the queue manager was stopped; for example:

```
C:\Users\Administrator>dspmq -m QMGR
QMNAME(QMGR) STATUS(Ended immediately)
```

b) On *venus* run the **strmqm** command to start *QMGR* permitting standbys:

```
strmqm -x QMGR
```

The system response:

```
WebSphere MQ queue manager 'QMGR' starting.
The queue manager is associated with installation 'Installation1'.
5 log records accessed on queue manager 'QMGR' during the log
replay phase.
Log replay for queue manager 'QMGR' complete.
Transaction manager state recovered for queue manager 'QMGR'.
WebSphere MQ queue manager 'QMGR' started using V7.1.0.0.
```

Results

To test the multi-instance queue manager switches over, do the following steps:

1. On *mars*, run the **strmqm** command to start *QMGR* permitting standbys:

```
strmqm -x QMGR
```

The system response:

```
WebSphere MQ queue manager 'QMGR' starting.
The queue manager is associated with installation 'Installation1'.
A standby instance of queue manager 'QMGR' has been started.
The active instance is running elsewhere.
```

2. On *venus* run the **endmqm** command:

```
endmqm -r -s -i QMGR
```

The system response on *venus*:

```
WebSphere MQ queue manager 'QMGR' ending.
WebSphere MQ queue manager 'QMGR' ending.
WebSphere MQ queue manager 'QMGR' ending.
WebSphere MQ queue manager 'QMGR' ending.
WebSphere MQ queue manager 'QMGR' ending.
WebSphere MQ queue manager 'QMGR' ending.
WebSphere MQ queue manager 'QMGR' ended, permitting switchover to
a standby instance.
```

And on *mars*:

```
dspmq
QMNAME(QMGR) STATUS(Running as standby)
C:\Users\wmquser2>dspmq
QMNAME(QMGR) STATUS(Running as standby)
C:\Users\wmquser2>dspmq
QMNAME(QMGR)  STATUS(Running)
```

What to do next

To verify a multi-instance queue manager using sample programs; see "Verify the multi-instance queue manager on Windows" on page 361.

Creating an Active Directory and DNS domain for IBM WebSphere MQ

This task creates the domain *wmq.example.com* on a Windows 2008 domain controller called *sun*. It configures the `Domain mqm` global group in the domain, with the correct rights, and with one user.

In a production scale configuration, you might have to tailor the configuration to an existing domain. For example, you might define different domain groups to authorize different shares, and to group the user IDs that run queue managers.

The example configuration consists of three servers:

sun
: A Windows Server 2008 domain controller. It owns the *wmq.example.com* domain that contains *Sun*, *mars*, and *venus*. For the purposes of illustration, it is also used as the file server.

mars
: A Windows Server 2008 used as the first IBM WebSphere MQ server. It contains one instance of the multi-instance queue manager called *QMGR*.

venus
: A Windows Server 2008 used as the second IBM WebSphere MQ server. It contains the second instance of the multi-instance queue manager called *QMGR*.

Replace the italicized names in the example, with names of your choosing.

Before you begin

1. The task steps are consistent with a Windows Server 2008 that is installed but not configured with any roles. If you are configuring an existing domain controller, you might find it useful to try out the steps on a new Windows Server 2008. You can adapt the steps to your domain.

About this task

In this task, you create an Active Directory and DNS domain on a new domain controller. You then configure it ready to install IBM WebSphere MQ on other servers and workstations that join the

domain. Follow the task if you are unfamiliar with installing and configuring Active Directory to create a Windows domain. You must create a Windows domain in order to create a multi-instance queue manager configuration. The task is not intended to guide you in the best way to configure a Windows domain. To deploy multi-instance queue managers in a production environment, you must consult Windows documentation.

During the task you do the following steps:

1. Install Active Directory.
2. Add a domain.
3. Add the domain to DNS.
4. Create the global group `Domain mqm` and give it the correct rights.
5. Add a user and make it a member of the global group `Domain mqm`.

This task is one of a set of related tasks that illustrate accessing queue manager data and log files. The tasks show how to create a queue manager authorized to read and write data and log files that are stored in a directory of your choosing. They accompany the task, "Windows domains and multi-instance queue managers" on page 339.

For the purposes of the task the domain controller hostname is *sun*, and the two IBM WebSphere MQ servers are called *mars* and *venus*. The domain is called *wmq.example.com*. You can replace all the italicized names in the task with names of your own choosing.

Procedure

1. Log on to the domain controller, *sun*, as the local or `Workgroup` administrator.

   If the server is already configured as a domain controller, you must log on as a domain administrator.
2. Run the Active Directory Domain Services wizard.

   a) Click **Start** > **Run...** Type `dcpromo` and click **OK**.

   If the Active Directory binary files are not already installed, Windows installs the files automatically.
3. In the first window of the wizard, leave the **Use advanced mode installation** check box clear. Click **Next** > **Next** and click **Create a new domain in a new forest** > **Next**.
4. Type *wmq.example.com* into the **FQDN of the forest root domain** field. Click **Next**.
5. In the Set Forest Functional Level window, select **Windows Server 2003**, or later, from the list of **Forest functional levels** > **Next**.

   The oldest level of Windows Server that is supported by IBM WebSphere MQ is Windows Server 2003.
6. Optional: In the Set Domain Functional Level window, select **Windows Server 2003**, or later, from the list of **Domain functional levels** > **Next**.

   This step is only required if you set the Forest Functional Level to **Windows Server 2003**.
7. The Additional Domain Controller Options window opens, with **DNS server** selected as an additional option. Click **Next** and **Yes** to clear the warning window.

   Tip: If a DNS server is already installed this option is not presented to you. If you want to follow this task precisely, remove all the roles from this domain controller and start again.
8. Leave the `Database`, `Log Files`, and SYSVOL directories unchanged; click **Next**.
9. Type a password into the **Password** and **Confirm password** fields in the Directory Services Restore Mode Administrator Password window. Click **Next** > **Next**. Select **Reboot on completion** in the final wizard window.
10. When the domain controller reboots, log on as *wmq*\`Adminstrator`.

    The server manager starts automatically.
11. Open the *wmq.example.com*\`Users` folder

a) Open **Server Manager** > **Roles** > **Active Directory Domain Services** > ***wmq.example.com*** > **Users**.

12. Right-click **Users** > **New** > **Group**.
    a) Type a group name into the **Group name** field.

       Note: The preferred group name is `Domain mqm`. Type it exactly as shown.

       - Calling the group `Domain mqm` modifies the behavior of the "Prepare IBM WebSphere MQ" wizard on a domain workstation or server. It causes the "Prepare IBM WebSphere MQ" wizard automatically to add the group `Domain mqm` to the local `mqm` group on each new installation of IBM WebSphere MQ in the domain.
       - You can install workstations or servers in a domain with no `Domain mqm` global group. If you do so, you must define a group with the same properties as `Domain mqm` group. You must make that group, or the users that are members of it, members of the local `mqm` group wherever IBM WebSphere MQ is installed in a domain. You can place domain users into multiple groups. Create multiple domain groups, each group corresponding to a set of installations that you want to manage separately. Split domain users, according to the installations they manage, into different domain groups. Add each domain group or groups to the local `mqm` group of different IBM WebSphere MQ installations. Only domain users in the domain groups that are members of a specific local `mqm` group can create, administer, and run queue managers for that installation.
       - The domain user that you nominate when installing IBM WebSphere MQ on a workstation or server in a domain must be a member of the `Domain mqm` group, or of an alternative group you defined with same properties as the `Domain mqm` group.

    b) Leave **Global** clicked as the **Group scope**, or change it to **Universal**. Leave **Security** clicked as the **Group type**. Click **OK**.

13. Add the rights, **Allow Read group membership** and **Allow Read groupMembershipSAM** to the rights of the `Domain mqm` global group.
    a) In the Server Manager action bar, click **View** > **Advanced features**
    b) In the Server Manager navigation tree, click **Users**
    c) In the Users window, right-click **Domain mqm** > **Properties**
    d) Click **Security** > **Advanced** > **Add...**. Type `Domain mqm` and click **Check names** > **OK**.

       The **Name** field is prefilled with the string, `Domain mqm (`*`domain name`*`\Domain mqm)`.
    e) Click **Properties**. In the **Apply to** list, select **Descendant User Objects** from the bottom of the list.
    f) From the **Permissions** list, select the **Read group membership** and **Read groupMembershipSAM Allow** check boxes; click **OK** > **Apply** > **OK** > **OK**.

14. Add two or more users to the `Domain mqm` global group.

    One user, *wmquser1* in the example, runs the IBM IBM WebSphere MQ service, and the other user, *wmquser2*, is used interactively.
    A domain user is required to create a queue manager that uses the alternative security group in a domain configuration. It is not sufficient for the user ID to be an administrator, although an administrator has authority to run the **crtmqm** command. The domain user, who could be an administrator, must be a member of the local `mqm` group as well as of the alternative security group.
    In the example, you make *wmquser1* and *wmquser2* members of the `Domain mqm` global group. The "Prepare IBM WebSphere MQ" wizard automatically configures `Domain mqm` as a member of the local `mqm` group where ever the wizard is run.
    You must provide a different user to run the IBM IBM WebSphere MQ service for each installation of IBM WebSphere MQ on a single computer. You can reuse the same users on different computers.

    a) In the Server Manager navigation tree, click **Users** > **New** > **User**
    b) In the New Object - User window, type *wmquser1* into the **User logon name** field. Type *WebSphere* into the **First name** field, and *MQ1* into the **Last name** field. Click **Next**.

c) Type a password into the **Password** and **Confirm password** fields, and clear the **User must change password at next logon** check box. Click **Next** > **Finish**.

d) In the Users window, right-click ***WebSphere MQ*** > **Add to a group...**. Type `Domain mqm` and click **Check Names** > **OK** > **OK**.

e) Repeat steps a to d to add *`WebSphere MQ2`* as *`wmquser2`*.

15. Running IBM WebSphere MQ as a service.

    If you need to run IBM WebSphere MQ as a service, and then give the domain user (that you obtained from your domain administrator) the right to run as a service, carry out the following procedure:

    a) Click **Start > Run...**.

    Type the command `secpol.msc` and click **OK**.

    b) Open **Security Settings > Local Policies > User Rights Assignments**.

    In the list of policies, right-click **Log on as a service > Properties**.

    c) Click **Add User or Group...**

    Type the name of the user you obtained from your domain administrator, and click **Check Names**

    d) If prompted by a Windows Security window, type the user name and password of an account user or administrator with sufficient authority, and click **OK > Apply > OK**.

    Close the Local Security Policy window.

    Note: On Windows Vista and Windows Server 2008 the User Account Control (UAC) is enabled by default.

    The UAC feature restricts the actions users can perform on certain operating system facilities, even if they are members of the Administrators group. You must take appropriate steps to overcome this restriction.

What to do next

Proceed to the next task, "Installing IBM WebSphere MQ on a server or workstation in a Windows domain" on page 346.

Installing IBM WebSphere MQ on a server or workstation in a Windows domain
In this task, you install and configure IBM WebSphere MQ on a server or workstation in the *`wmq.example.com`* Windows domain.

In a production scale configuration, you might have to tailor the configuration to an existing domain. For example, you might define different domain groups to authorize different shares, and to group the user IDs that run queue managers.

The example configuration consists of three servers:

`sun`
A Windows Server 2008 domain controller. It owns the *`wmq.example.com`* domain that contains *`Sun`*, *`mars`*, and *`venus`*. For the purposes of illustration, it is also used as the file server.

`mars`
A Windows Server 2008 used as the first IBM WebSphere MQ server. It contains one instance of the multi-instance queue manager called *`QMGR`*.

`venus`
A Windows Server 2008 used as the second IBM WebSphere MQ server. It contains the second instance of the multi-instance queue manager called *`QMGR`*.

Replace the italicized names in the example, with names of your choosing.

Before you begin

1. Do the steps in "Creating an Active Directory and DNS domain for IBM WebSphere MQ" on page 343 to create a domain controller, *`sun`*, for the domain *`wmq.example.com`*. Change the italicized names to suit your configuration.

2. See Hardware and software requirements on Windows systems for other Windows versions you can run IBM WebSphere MQ on.

About this task

In this task you configure a Windows Server 2008, called *mars*, as a member of the *wmq.example.com* domain. You install IBM WebSphere MQ, and configure the installation to run as a member of the *wmq.example.com* domain.

This task is one of a set of related tasks that illustrate accessing queue manager data and log files. The tasks show how to create a queue manager authorized to read and write data and log files that are stored in a directory of your choosing. They accompany the task, "Windows domains and multi-instance queue managers" on page 339.

For the purposes of the task the domain controller hostname is *sun*, and the two IBM WebSphere MQ servers are called *mars* and *venus*. The domain is called *wmq.example.com*. You can replace all the italicized names in the task with names of your own choosing.

Procedure

1. Add the domain controller, *sun.wmq.example.com* to *mars* as a DNS server.
   a) On *mars*, log on as *mars*\Administrator and click **Start**.
   b) Right-click **Network** > **Properties** > **Manage network connections**.
   c) Right-click the network adapter, click **Properties**.

      The system responds with the Local Area Connection Properties window listing items the connection uses.
   d) Select the **Internet Protocol Version 4** or **Internet Protocol Version 6** from the list of items in the Local Area Connection Properties window. Click **Properties** > **Advanced...** and click the **DNS** tab.
   e) Under the DNS server addresses, click **Add...**.
   f) Type the IP address of the domain controller, which is also the DNS server, and click **Add**.
   g) Click **Append these DNS suffixes** > **Add...**.
   h) Type *wmq.example.com* and click **Add**.
   i) Type *wmq.example.com* in the **DNS suffix for this connection** field.
   j) Select **Register this connection's address in DNS** and **Use this connection's suffix in DNS registration**. Click **OK** > **OK** > **Close**
   k) Open a command window, and type the command **ipconfig /all** to review the TCP/IP settings.
2. On *mars*, add the computer to the *wmq.example.com* domain.
   a) Click **Start**
   b) Right-click **Computer** > **Properties**. In the Computer name, domain and workgroup settings division, click **Change settings**.
   c) In the System Properties windows, click **Change...**.
   d) Click Domain, type *wmq.example.com*, and click **OK**.
   e) Type the **User name** and **Password** of the domain controller administrator, who has the authority to permit the computer to join the domain, and click **OK**.
   f) Click **OK** > **OK** > **Close** > **Restart Now** in response to the "Welcome to the *wmq.example.com* domain" message.
3. Check that the computer is a member of the *wmq.example.com* domain
   a) On *sun*, log on to the domain controller as *wmq*\Administrator.
   b) Open **Server Manager** > **Active Directory Domain Services** > ***wmq.example.com*** > **Computers** and check *mars* is listed correctly in the Computers window.
4. Install IBM WebSphere MQ for Windows on *mars*.

For further information about running the IBM WebSphere MQ for Windows installation wizard; see Installing IBM WebSphere MQ server on Windows .

a) On *mars*, log on as the local administrator, *mars*\Administrator.

b) Run the **Setup** command on the IBM WebSphere MQ for Windows installation media.

The IBM WebSphere MQ Launchpad application starts.

c) Click **Software Requirements** to check that the prerequisite software is installed.

d) Click **Network Configuration** > **Yes** to configure a domain user ID.

The task, "Creating an Active Directory and DNS domain for IBM WebSphere MQ" on page 343, configures a domain user ID for this set of tasks.

e) Click **WebSphere MQ Installation**, select an installation language and click Launch IBM IBM WebSphere MQ Installer.

f) Confirm the license agreement and click **Next** > **Next** > **Install** to accept the default configuration. Wait for the installation to complete, and click **Finish**.

You might want to change the name of the installation, install different components, configure a different directory for queue manager data and logs, or install into a different directory. If so, click **Custom** rather than **Typical**.

IBM WebSphere MQ is installed, and the installer starts the "Prepare IBM WebSphere MQ" wizard.

Important: Do not run the wizard yet.

5. Configure the user that is going to run the IBM IBM WebSphere MQ service with the **Run as a service** right.

Choose whether to configure the local mqm group, the Domain mqm group, or the user that is going to run the IBM IBM WebSphere MQ service with the right. In the example, you give the user the right.

a) Click **Start** > **Run...**, type the command **secpol.msc** and click **OK**.

b) Open **Security Settings** > **Local Policies** > **User Rights Assignments**. In the list of policies, right-click **Log on as a service** > **Properties**.

c) Click **Add User or Group...** and type *wmquser1* and click **Check Names**

d) Type the user name and password of a domain administrator, *wmq*\Administrator, and click **OK** > **Apply** > **OK**. Close the Local Security Policy window.

6. Run the "Prepare IBM WebSphere MQ" wizard.

For further information about running the "Prepare IBM WebSphere MQ" wizard; see Configuring WebSphere MQ with the Prepare WebSphere MQ wizard .

a) The IBM IBM WebSphere MQ Installer runs the "Prepare IBM WebSphere MQ" automatically.

To start the wizard manually, find the shortcut to the "Prepare IBM WebSphere MQ" in the **Start** > **All programs** > **IBM WebSphere MQ** folder. Select the shortcut that corresponds to the installation of IBM WebSphere MQ in a multi-installation configuration.

b) Click **Next** and leave **Yes** clicked in response to the question "Identify if there is a Windows 2000 or later domain controller in the network".

c) Click **Yes** > **Next** in the first Configuring IBM WebSphere MQ for Windows for Windows domain users window.

d) In the second Configuring IBM WebSphere MQ for Windows for Windows domain users window, type *wmq* in the **Domain** field. Type *wmquser1* in the **User name** field, and the password, if you set one, in the **Password** field. Click **Next**.

The wizard configures and starts the IBM IBM WebSphere MQ with *wmquser1*.

e) In the final page of the wizard, select or clear the check boxes as you require and click **Finish**.

What to do next

1. Do the task, "Reading and writing data and log files authorized by the local mqm group" on page 368, to verify that the installation and configuration are working correctly.
2. Do the task, "Creating a shared directory for queue manager data and log files" on page 349, to configure a file share to store the data and log files of a multi-instance queue manager.

Related concepts

User rights required for a WebSphere MQ Windows Service

Creating a shared directory for queue manager data and log files

This task is one of a set of related tasks that illustrate accessing queue manager data and log files. The tasks show how to create a queue manager authorized to read and write data and log files that are stored in a directory of your choosing.

In a production scale configuration, you might have to tailor the configuration to an existing domain. For example, you might define different domain groups to authorize different shares, and to group the user IDs that run queue managers.

The example configuration consists of three servers:

sun
A Windows Server 2008 domain controller. It owns the *wmq.example.com* domain that contains *Sun*, *mars*, and *venus*. For the purposes of illustration, it is also used as the file server.

mars
A Windows Server 2008 used as the first IBM WebSphere MQ server. It contains one instance of the multi-instance queue manager called *QMGR*.

venus
A Windows Server 2008 used as the second IBM WebSphere MQ server. It contains the second instance of the multi-instance queue manager called *QMGR*.

Replace the italicized names in the example, with names of your choosing.

Before you begin

1. To do this task exactly as documented, do the steps in the task, "Creating an Active Directory and DNS domain for IBM WebSphere MQ" on page 343, to create the domain *sun.wmq.example.com* on the domain controller *sun*. Change the italicized names to suit your configuration.

About this task

This task is one of a set of related tasks that illustrate accessing queue manager data and log files. The tasks show how to create a queue manager authorized to read and write data and log files that are stored in a directory of your choosing. They accompany the task, "Windows domains and multi-instance queue managers" on page 339.

In the task, you create a share containing a data and log directory, and a global group to authorize access to the share. You pass the name of the global group that authorizes the share to the **crtmqm** command in its `-a` parameter. The global group gives you the flexibility of separating the users of this share from users of other shares. If you do not need this flexibility, authorize the share with the `Domain mqm` group rather than create a new global group.

The global group used for sharing in this task is called *wmqha*, and the share is called *wmq*. They are defined on the domain controller *sun* in the Windows domain *wmq.example.com*. The share has full control permissions for the global group *wmqha*. Replace the italicized names in the task with names of your choosing.

For the purposes of this task the domain controller is the same server as the file server. In practical applications, split the directory and file services between different servers for performance and availability.

You must configure the user ID that the queue manager is running under to be a member of two groups. It must be a member of the local `mqm` group on an IBM WebSphere MQ server, and of the *wmqha* global group.

In this set of tasks, when the queue manager is running as a service, it runs under the user ID *wmquser1*, so *wmquser1* must be a member of *wmqha*. When the queue manager is running interactively, it runs under the user ID *wmquser2*, so *wmquser2* must be a member of *wmqha*. Both *wmquser1* and *wmquser2* are members of the global group `Domain mqm`. `Domain mqm` is a member of the local `mqm` group on the *mars* and *venus* IBM WebSphere MQ servers. Hence, *wmquser1* and *wmquser2* are members of the local `mqm` group on bothIBM WebSphere MQ servers.

Procedure

1. Log on to the domain controller, *sun.wmq.example.com* as the domain administrator.
2. Create the global group *wmqha*.
   a) Open **Server Manager** > **Roles** > **Active Directory Domain Services** > ***wmq.example.com*** > **Users**.
   b) Open the *wmq.example.com*`\Users` folder
   c) Right-click **Users** > **New** > **Group**.
   d) Type *wmqha* into the **Group name** field.
   e) Leave **Global** clicked as the **Group scope** and **Security** as the **Group type**. Click **OK**.
3. Add the domain users *wmquser1* and *wmquser2* to the global group, *wmqha*.
   a) In the Server Manager navigation tree, click **Users** and right-click ***wmqha*** > **Properties** in the list of users.
   b) Click the Members tab in the *wmqha* Properties window.
   c) Click **Add...**; type *wmquser1*`;` *wmquser2* and click **Check Names** > **OK** > **Apply** > **OK**.
4. Create the directory tree to contain queue manager data and log files.
   a) Open a command prompt.
   b) Type the command:

```
md c:\wmq\data , c:\wmq\logs
```

5. Authorize the global group *wmqha* to have full control permission to the *c:\wmq* directories and share.
   a) In Windows Explorer, right-click ***c:\wmq*** > **Properties**.
   b) Click the **Security** tab and click **Advanced** > **Edit...**.
   c) Clear the check box for **Include inheritable permissions from this object's owner**. Click **Copy** in the Windows Security window.
   d) Select the lines for Users in the list of **Permission entries** and click **Remove**. Leave the lines for SYSTEM, Administrators, and CREATOR OWNER in the list of **Permission entries**.
   e) Click **Add...**, and type the name of the global group *wmqha*. Click **Check Names** > **OK**.
   f) In the Permission Entry for `wmq` window, select **Full Control** in the list of **Permissions**.
   g) Click **OK** > **Apply** > **OK** > **OK** > **OK**
   h) In Windows Explorer, right-click ***c:\wmq*** > **Share...**.
   i) Click **Advanced Sharing...** and select the **Share this folder** check box. Leave the share name as *wmq*.
   j) Click **Permissions** > **Add...**, and type the name of the global group *wmqha*. Click **Check Names** > **OK**.
   k) Select *wmqha* in the list of **Group or user names**. Select the **Full Control** check box in the list of **Permissions for *wmqha***; click **Apply**.
   l) Select *Administrators* in the list of **Group or user names**. Select the **Full Control** check box in the list of **Permissions for *Administrators***; click **Apply** > **OK** > **OK** > **Close**.

What to do next

Check that you can read and write files to the shared directories from each of the IBM WebSphere MQ servers. Check the IBM IBM WebSphere MQ service user ID, *wmquser1* and the interactive user ID, *wmquser2*.

1. If you are using remote desktop, you must add *wmq**wmquser1* and *wmquser2* to the local group `Remote Desktop Users` on *mars*.
   a. Log on to *mars* as *wmq*\`Administrator`
   b. Run the **lusrmgr.msc** command to open the Local Users and Groups window.
   c. Click **Groups**. Right-click **Remote Desktop Users** > **Properties** > **Add...**. Type *wmquser1*; *wmquser2* and click **Check Names**.
   d. Type in the user name and password of the domain administrator, *wmq*\`Administrator`, and click **OK** > **Apply** > **OK**.
   e. Close the Local Users and Groups window.
2. Log on to *mars* as *wmq**wmquser1*.
   a. Open a Windows Explorer window, and type in *sun**wmq*.

      The system responds by opening the *wmq* share on *sun.wmq.example.com*, and lists the data and logs directories.
   b. Check the permissions of *wmquser1* by creating a file in data subdirectory, adding some content, reading it, and then deleting it.
3. Log on to *mars* as *wmq**wmquser2*, and repeat the checks.
4. Do the next task, to create a queue manager to use the shared data and log directories; see "Reading and writing shared data and log files authorized by an alternative global security group" on page 351.

Reading and writing shared data and log files authorized by an alternative global security group
This task shows how to use the `-a` flag on the **crtmqm** command. The `-a` flag gives the queue manager access to its log and data files on a remote file share using the alternative security group.

In a production scale configuration, you might have to tailor the configuration to an existing domain. For example, you might define different domain groups to authorize different shares, and to group the user IDs that run queue managers.

The example configuration consists of three servers:

sun
A Windows Server 2008 domain controller. It owns the *wmq.example.com* domain that contains *Sun*, *mars*, and *venus*. For the purposes of illustration, it is also used as the file server.

mars
A Windows Server 2008 used as the first IBM WebSphere MQ server. It contains one instance of the multi-instance queue manager called *QMGR*.

venus
A Windows Server 2008 used as the second IBM WebSphere MQ server. It contains the second instance of the multi-instance queue manager called *QMGR*.

Replace the italicized names in the example, with names of your choosing.

Before you begin

Do the steps in the following tasks. The tasks create the domain controller and domain, install IBM WebSphere MQ for Windows on one server, and create the file share for data and log files. If you are configuring an existing domain controller, you might find it useful to try out the steps on a new Windows Server 2008. You can adapt the steps to your domain.

1. "Creating an Active Directory and DNS domain for IBM WebSphere MQ" on page 343.
2. "Installing IBM WebSphere MQ on a server or workstation in a Windows domain" on page 346.

3. “Creating a shared directory for queue manager data and log files” on page 349.

About this task

This task is one of a set of related tasks that illustrate accessing queue manager data and log files. The tasks show how to create a queue manager authorized to read and write data and log files that are stored in a directory of your choosing. They accompany the task, “Windows domains and multi-instance queue managers” on page 339.

In this task, you create a queue manager that stores its data and logs in a remote directory on a file server. For the purposes of this example, the file server is the same server as the domain controller. The directory containing the data and log folders is shared with full control permission given to the global group `wmqha`.

Procedure

1. Log on to the domain server, *`mars`*, as the local administrator, *`mars`*`\Administrator`.
2. Open a command window.
3. Restart the IBM IBM WebSphere MQ service.

   You must restart the service so that the user ID it runs under acquires the additional security credentials you configured for it.

   Type the commands:

   ```
   endmqsvc
   strmqsvc
   ```

   The system responses:

   ```
   5724-H72 (C) Copyright IBM Corp. 1994, 2025. ALL RIGHTS RESERVED.
   The MQ service for installation 'Installation1' ended successfully.
   And:
   5724-H72 (C) Copyright IBM Corp. 1994, 2025. ALL RIGHTS RESERVED.
   The MQ service for installation 'Installation1' started successfully.
   ```
4. Create the queue manager.

   ```
   crtmqm -a wmq\wmqha -sax -u SYSTEM.DEAD.LETTER.QUEUE -md \\sun\wmq\data -ld \\sun\wmq\logs
   QMGR
   ```

   You must specify the domain, *`wmq`*, of the alternative security group *`wmqha`* by specifying full domain name of the global group *`"wmq\wmqha"`*.

   You must spell out the Universal Naming Convention (UNC) name of the share *`\\sun\wmq`*, and not use a mapped drive reference.

   The system response:

   ```
   WebSphere MQ queue manager created.
   Directory '\\sun\wmq\data\QMGR' created.
   The queue manager is associated with installation '1'
   Creating or replacing default objects for queue manager 'QMGR'
   Default objects statistics : 74 created. 0 replaced.
   Completing setup.
   Setup completed.
   ```

What to do next

Test the queue manager by putting and getting a message to a queue.

1. Start the queue manager.

   ```
   strmqm QMGR
   ```

The system response:

```
WebSphere MQ queue manager 'QMGR' starting.
The queue manager is associated with installation '1'.
5 log records accessed on queue manager 'QMGR' during the log
replay phase.
Log replay for queue manager 'QMGR' complete.
Transaction manager state recovered for queue manager 'QMGR'.
WebSphere MQ queue manager 'QMGR' started using V7.1.0.0.
```

2. Create a test queue.

```
echo define qlocal(QTEST) | runmqsc QMGR
```

The system response:

```
5724-H72 (C) Copyright IBM Corp. 1994, 2025. ALL RIGHTS RESERVED.
Starting MQSC for queue manager QMGR.


     1 : define qlocal(QTEST)
AMQ8006: WebSphere MQ queue created.
One MQSC command read.
No commands have a syntax error.
All valid MQSC commands were processed.
```

3. Put a test message using the sample program **amqsput**.

```
echo 'A test message' | amqsput QTEST QMGR
```

The system response:

```
Sample AMQSPUT0 start
target queue is QTEST
Sample AMQSPUT0 end
```

4. Get the test message using the sample program **amqsget**.

```
amqsget QTEST QMGR
```

The system response:

```
Sample AMQSGET0 start
message <A test message>
Wait 15 seconds ...
no more messages
Sample AMQSGET0 end
```

5. Stop the queue manager.

```
endmqm -i QMGR
```

The system response:

```
WebSphere MQ queue manager 'QMGR' ending.
WebSphere MQ queue manager 'QMGR' ended.
```

6. Delete the queue manager.

```
dltmqm QMGR
```

The system response:

```
WebSphere MQ queue manager 'QMGR' deleted.
```

7. Delete the directories you created.

Tip: Add the /Q option to the commands to prevent the command prompting to delete each file or directory.

```
del /F /S C:\wmq\*.*
rmdir /S C:\wmq
```

Create a multi-instance queue manager on domain controllers
An example shows how to set up a multi-instance queue manager on Windows on domain controllers. The setup demonstrates the concepts involved, rather than being production scale. The example is based on Windows Server 2008. The steps might differ on other versions of Windows Server.

The configuration uses the concept of a mini-domain, or "domainlet"; see Windows 2000, Windows Server 2003, and Windows Server 2008 cluster nodes as domain controllers. To add multi-instance queue managers to an existing domain, see “Create a multi-instance queue manager on domain workstations or servers” on page 340.

The example configuration consists of three servers:

sun
A Windows Server 2008 server used as the first domain controller. It defines the *wmq.example.com* domain that contains *sun*, *earth*, and *mars*. It contains one instance of the multi-instance queue manager called *QMGR*.

earth
A Windows Server 2008 used as the second domain controller IBM WebSphere MQ server. It contains the second instance of the multi-instance queue manager called *QMGR*.

mars
A Windows Server 2008 used as the file server.

Replace the italicized names in the example, with names of your choosing.

Before you begin

1. On Windows, you do not need to verify the file system that you plan to store queue manager data and log files on. The checking procedure, Verifying shared file system behavior , is applicable to UNIX and Linux. On Windows, the checks are always successful.
2. Do the steps in “Creating an Active Directory and DNS domain for IBM WebSphere MQ” on page 343 to create the first domain controller.
3. Do the steps in “Adding a second domain controller to the wmq.example.com domain” on page 357 to add a second domain controller, install IBM WebSphere MQ for Windows on both domain controllers, and verify the installations.
4. Do the steps in “Installing IBM WebSphere MQ on domain controllers in the wmq.example.com domain” on page 359 to install IBM WebSphere MQ on the two domain controllers.

About this task

On a file server in the same domain create a share for the queue manager log and data directories. Next, create the first instance of a multi-instance queue manager that uses the file share on one of the domain controllers. Create the other instance on the other domain controller and finally verify the configuration. You can create the file share on a domain controller.

In the sample, *sun* is the first domain controller, *earth* the second, and *mars* is the file server.

Procedure

1. Create the directories that are to contain the queue manager data and log files.
   a) On *mars*, type the command:

```
md c:\wmq\data , c:\wmq\logs
```

2. Share the directories that are to contain the queue manager data and log files.

   You must permit full control access to the domain local group `mqm`, and the user ID you use to create the queue manager. In the example, user IDs that are members of `Domain Administrators` have the authority to create queue managers.

   The file share must be on a server that is in the same domain as the domain controllers. In the example, the server *`mars`* is in the same domain as the domain controllers.

   a) In Windows Explorer, right-click ***c:\wmq*** > **Properties**.

   b) Click the **Security** tab and click **Advanced** > **Edit...**.

   c) Clear the check box for **Include inheritable permissions from this object's owner**. Click **Copy** in the Windows Security window.

   d) Select the lines for Users in the list of **Permission entries** and click **Remove**. Leave the lines for SYSTEM, Administrators, and CREATOR OWNER in the list of **Permission entries**.

   e) Click **Add...**, and type the name of the domain local group *`mqm`*. Click **Check Names**

   f) In response to a Windows Security window, Type the name and password of the `Domain Administrator` and click **OK** > **OK**.

   g) In the Permission Entry for `wmq` window, select **Full Control** in the list of **Permissions**.

   h) Click **OK** > **Apply** > **OK** > **OK** > **OK**

   i) Repeat steps e to h to add `Domain Administrators`.

   j) In Windows Explorer, right-click ***c:\wmq*** > **Share...**.

   k) Click **Advanced Sharing...** and select the **Share this folder** check box. Leave the share name as *`wmq`*.

   l) Click **Permissions** > **Add...**, and type the name of the domain local group *`mqm`*`; Domain Administrators`. Click **Check Names**.

   m) In response to a Windows Security window, Type the name and password of the `Domain Administrator` and click **OK** > **OK**.

3. Create the queue manager *`QMGR`* on the first domain controller, *`sun`*.

   ```
   crtmqm -sax -u SYSTEM.DEAD.LETTER.QUEUE -md \\mars\wmq\data -ld \\mars\wmq\logs  QMGR
   ```

   The system response:

   ```
   WebSphere MQ queue manager created.
   Directory '\\mars\wmq\data\QMGR' created.
   The queue manager is associated with installation 'Installation1'.
   Creating or replacing default objects for queue manager 'QMGR'.
   Default objects statistics : 74 created. 0 replaced. 0 failed.
   Completing setup.
   Setup completed.
   ```

4. Start the queue manager on *`sun`*, permitting a standby instance.

   ```
   strmqm -x QMGR
   ```

   The system response:

   ```
   WebSphere MQ queue manager 'QMGR' starting.
   The queue manager is associated with installation 'Installation1'.
   5 log records accessed on queue manager 'QMGR' during the log
   replay phase.
   Log replay for queue manager 'QMGR' complete.
   Transaction manager state recovered for queue manager 'QMGR'.
   WebSphere MQ queue manager 'QMGR' started using V7.1.0.0.
   ```

5. Create a second instance of *`QMGR`* on *`earth`*.

a) Check the values of the `Prefix` and `InstallationName` parameters are correct for *earth*.

On *sun*, run the **dspmqinf** command:

```
dspmqinf QMGR
```

The system response:

```
QueueManager:
   Name=QMGR
   Directory=QMGR
   Prefix=C:\Program Files\IBM\WebSphere MQ
   DataPath=\\mars\wmq\data\QMGR
   InstallationName=Installation1
```

b) Copy the machine-readable form of the **QueueManager** stanza to the clipboard.

On *sun* run the **dspmqinf** command again, with the `-o command` parameter.

```
dspmqinf -o command QMGR
```

The system response:

```
addmqinf -s QueueManager -v Name=QMGR
-v Directory=QMGR -v Prefix="C:\Program Files\IBM\WebSphere MQ"
-v DataPath=\\mars\wmq\data\QMGR
```

c) On *earth* run the **addmqinf** command from the clipboard to create an instance of the queue manager on *earth*.

Adjust the command, if necessary, to accommodate differences in the `Prefix` or `InstallationName` parameters.

```
addmqinf -s QueueManager -v Name=QMGR
-v Directory=QMGR -v Prefix="C:\Program Files\IBM\WebSphere MQ"
-v DataPath=\\mars\wmq\data\QMGR
```

```
WebSphere MQ configuration information added.
```

6. Start the standby instance of the queue manager on *earth*.

```
strmqm -x QMGR
```

The system response:

```
WebSphere MQ queue manager 'QMGR' starting.
The queue manager is associated with installation 'Installation1'.
A standby instance of queue manager 'QMGR' has been started. The active
instance is running elsewhere.
```

Results

Verify that the queue manager switches over from *sun* to *earth*:

1. On *sun*, run the command:

```
endmqm -i -r -s QMGR
```

The system response on *sun*:

```
WebSphere MQ queue manager 'QMGR' ending.
WebSphere MQ queue manager 'QMGR' ending.
WebSphere MQ queue manager 'QMGR' ending.
WebSphere MQ queue manager 'QMGR' ending.
WebSphere MQ queue manager 'QMGR' ending.
```

```
WebSphere MQ queue manager 'QMGR' ending.
WebSphere MQ queue manager 'QMGR' ended, permitting switchover to
a standby instance.
```

2. On *earth* repeatedly type the command:

```
dspmq
```

The system responses:

```
QMNAME(QMGR) STATUS(Running as standby)
QMNAME(QMGR) STATUS(Running as standby)
QMNAME(QMGR) STATUS(Running)
```

What to do next

To verify a multi-instance queue manager using sample programs; see "Verify the multi-instance queue manager on Windows" on page 361.

Related tasks

"Adding a second domain controller to the wmq.example.com domain" on page 357

"Installing IBM WebSphere MQ on domain controllers in the wmq.example.com domain" on page 359

Related information

Windows 2000, Windows Server 2003, and Windows Server 2008 cluster nodes as domain controllers

Adding a second domain controller to the wmq.example.com domain

Add a second domain controller to the *wmq.example.com* domain to construct a Windows domain in which to run multi-instance queue managers on domain controllers and file servers.

The example configuration consists of three servers:

sun
: A Windows Server 2008 server used as the first domain controller. It defines the *wmq.example.com* domain that contains *sun*, *earth*, and *mars*. It contains one instance of the multi-instance queue manager called *QMGR*.

earth
: A Windows Server 2008 used as the second domain controller IBM WebSphere MQ server. It contains the second instance of the multi-instance queue manager called *QMGR*.

mars
: A Windows Server 2008 used as the file server.

Replace the italicized names in the example, with names of your choosing.

Before you begin

1. Do the steps in "Creating an Active Directory and DNS domain for IBM WebSphere MQ" on page 343 to create a domain controller, *sun*, for the domain *wmq.example.com*. Change the italicized names to suit your configuration.
2. Install Windows Server 2008 on a server in the default workgroup, WORKGROUP. For the example, the server is named *earth*.

About this task

In this task you configure a Windows Server 2008, called *earth*, as a second domain controller in the *wmq.example.com* domain.

This task is one of a set of related tasks that illustrate accessing queue manager data and log files. The tasks show how to create a queue manager authorized to read and write data and log files that are stored in a directory of your choosing. They accompany the task, "Windows domains and multi-instance queue managers" on page 339.

Procedure

1. Add the domain controller, *sun.wmq.example.com* to *earth* as a DNS server.
   a) On *earth*, log on as *earth*\Administrator and click **Start**.
   b) Right-click **Network** > **Properties** > **Manage network connections**.
   c) Right-click the network adapter, click **Properties**.

      The system responds with the Local Area Connection Properties window listing items the connection uses.
   d) Select the **Internet Protocol Version 4** or **Internet Protocol Version 6** from the list of items in the Local Area Connection Properties window. Click **Properties** > **Advanced...** and click the **DNS** tab.
   e) Under the DNS server addresses, click **Add...**.
   f) Type the IP address of the domain controller, which is also the DNS server, and click **Add**.
   g) Click **Append these DNS suffixes** > **Add...**.
   h) Type *wmq.example.com* and click **Add**.
   i) Type *wmq.example.com* in the **DNS suffix for this connection** field.
   j) Select **Register this connection's address in DNS** and **Use this connection's suffix in DNS registration**. Click **OK** > **OK** > **Close**
   k) Open a command window, and type the command **ipconfig /all** to review the TCP/IP settings.
2. Log on to the domain controller, *sun*, as the local or Workgroup administrator.

   If the server is already configured as a domain controller, you must log on as a domain administrator.
3. Run the Active Directory Domain Services wizard.
   a) Click **Start** > **Run...** Type dcpromo and click **OK**.

   If the Active Directory binary files are not already installed, Windows installs the files automatically.
4. Configure *earth* as the second domain controller in the *wmq.example.com* domain.
   a) In the first window of the wizard, leave the **Use advanced mode installation** check box clear. Click **Next** > **Next** and click **Create Add a domain controller to an existing domain** > **Next**.
   b) Type *wmq* into the **Type the name of any domain in this forest ...** field. The **Alternate credentials** radio button is clicked, click **Set...**. Type in the name and password of the domain administrator and click **OK** > **Next** > **Next** > **Next**.
   c) In the Additional Domain Controller Options window accept the **DNS server** and **Global catalog** options, which are selected; click **Next** > **Next**.
   d) On the Directory Services Restore Mode Administrator Password, type in a **Password** and **Confirm password** and click **Next** > **Next**.
   e) When prompted for **Network Credentials**, type in the password of the domain administrator. Select **Reboot on completion** in the final wizard window.
   f) After a while, a window might open with a **DCPromo** error concerning DNS delegation; click **OK**. The server reboots.

Results

When *earth* has rebooted, log on as Domain Administrator. Check that the wmq.example.com domain has been replicated to *earth*.

What to do next

Continue with installing IBM WebSphere MQ; see "Installing IBM WebSphere MQ on domain controllers in the wmq.example.com domain" on page 359.

Related tasks

"Creating an Active Directory and DNS domain for IBM WebSphere MQ" on page 343

Installing IBM WebSphere MQ on domain controllers in the wmq.example.com domain
Install and configure installations of IBM WebSphere MQ on both domain controllers in the *wmq.example.com* domain.
Put your short description here; used for first paragraph and abstract.

The example configuration consists of three servers:

sun
A Windows Server 2008 server used as the first domain controller. It defines the *wmq.example.com* domain that contains *sun*, *earth*, and *mars*. It contains one instance of the multi-instance queue manager called *QMGR*.

earth
A Windows Server 2008 used as the second domain controller IBM WebSphere MQ server. It contains the second instance of the multi-instance queue manager called *QMGR*.

mars
A Windows Server 2008 used as the file server.

Replace the italicized names in the example, with names of your choosing.

Before you begin

1. Do the steps in "Creating an Active Directory and DNS domain for IBM WebSphere MQ" on page 343 to create a domain controller, *sun*, for the domain *wmq.example.com*. Change the italicized names to suit your configuration.
2. Do the steps in "Adding a second domain controller to the wmq.example.com domain" on page 357 to create a second domain controller, *earth*, for the domain *wmq.example.com*. Change the italicized names to suit your configuration.
3. See Hardware and software requirements on Windows systems for other Windows versions you can run IBM WebSphere MQ on.

About this task

Install and configure installations of IBM WebSphere MQ on both domain controllers in the *wmq.example.com* domain.

Procedure

1. Install IBM WebSphere MQ on *sun* and *earth*.

   For further information about running the IBM WebSphere MQ for Windows installation wizard; see Installing IBM WebSphere MQ server on Windows .

   a) On both *sun* and *earth*, log on as the domain administrator, *wmq*\Administrator.

   b) Run the **Setup** command on the IBM WebSphere MQ for Windows installation media.

      The IBM WebSphere MQ Launchpad application starts.

   c) Click **Software Requirements** to check that the prerequisite software is installed.

   d) Click **Network Configuration** > **No**.

      You can configure either a domain user ID or not for this installation. The user ID that is created is a domain local user ID.

   e) Click **WebSphere MQ Installation**, select an installation language and click Launch IBM IBM WebSphere MQ Installer.

   f) Confirm the license agreement and click **Next** > **Next** > **Install** to accept the default configuration. Wait for the installation to complete, and click **Finish**.

      If you want to change the name of the installation, install different components, configure a different directory for queue manager data and logs, or install into a different directory, click **Custom** rather than **Typical**.

IBM WebSphere MQ is installed, and the installer starts the "Prepare IBM WebSphere MQ" wizard.

The IBM WebSphere MQ for Windows installation configures a domain local group `mqm`, and a domain group `Domain mqm`. It makes `Domain mqm` a member of `mqm`. Subsequent domain controllers in the same domain share the `mqm` and `Domain mqm` groups.

2. On both *earth* and *sun*, run the "Prepare IBM WebSphere MQ" wizard.

   For further information about running the "Prepare IBM WebSphere MQ" wizard, see Configuring WebSphere MQ with the Prepare WebSphere MQ wizard .

   a) The IBM WebSphere MQ installer runs the "Prepare IBM WebSphere MQ" automatically.

      To start the wizard manually, find the shortcut to the "Prepare IBM WebSphere MQ" in the **Start** > **All programs** > **IBM WebSphere MQ** folder. Select the shortcut that corresponds to the installation of IBM WebSphere MQ in a multi-installation configuration.

   b) Click **Next** and leave **No** clicked in response to the question "Identify if there is a Windows 2000 or later domain controller in the network"[1].

   c) In the final page of the wizard, select or clear the check boxes as you require and click **Finish**.

   The "Prepare IBM WebSphere MQ" wizard creates a domain local user `MUSR_MQADMIN` on the first domain controller, and another domain local user `MUSR_MQADMIN1` on the second domain controller. The wizard creates the IBM IBM WebSphere MQ service on each controller, with `MUSR_MQADMIN` or `MUSR_MQADMIN1` as the user that logs on the service.

3. Define a user that has permission to create a queue manager.

   The user must have the right to log on locally, and be a member of the domain local `mqm` group. On domain controllers, domain users do not have the right to log on locally, but administrators do. By default, no user has both these attributes. In this task, add domain administrators to the domain local `mqm` group.

   a) Open **Server Manager** > **Roles** > **Active Directory Domain Services** > ***wmq.example.com*** > **Users**.

   b) Right-click **Domain Admins** > **Add to a group...** and type `mqm`; click **Check names** > **OK** > **OK**

Results

1. Check that the "Prepare IBM WebSphere MQ" created the domain user, `MUSR_MQADMIN`:

   a. Open **Server Manager** > **Roles** > **Active Directory Domain Services** > ***wmq.example.com*** > **Users**.

   b. Right-click **MUSR_MQADMIN** > **Properties...** > **Member Of**, and see that it is a member of `Domain users` and `mqm`.

2. Check that `MUSR_MQADMIN` has the right to run as a service:

   a. Click Click **Start** > **Run...**, type the command **`secpol.msc`** and click **OK**.

   b. Open **Security Settings** > **Local Policies** > **User Rights Assignments**. In the list of policies, right-click **Log on as a service** > **Properties**, and see `MUSR_MQADMIN` is listed as having the right to log on as a service. Click **OK**.

What to do next

1. Do the task, “Reading and writing data and log files authorized by the local mqm group” on page 368, to verify that the installation and configuration are working correctly.
2. Go back to the task, “Create a multi-instance queue manager on domain controllers” on page 354, to complete the task of configuring a multi-instance queue manager on domain controllers.

Related concepts

User rights required for a WebSphere MQ Windows Service

[1] You can configure the installation for the domain. As all users and groups on a domain controller have domain scope, it does not make any difference. It is simpler to install IBM WebSphere MQ as if it is not in domain.

Verify the multi-instance queue manager on Windows
Use the sample programs **amqsghac**, **amqsphac** and **amqsmhac** to verify a multi-instance queue manager configuration. This topic provides an example configuration to verify a multi-instance queue manager configuration on Windows Server 2003.

The high availability sample programs use automatic client reconnection. When the connected queue manager fails, the client attempts to reconnect to a queue manager in the same queue manager group. The description of the samples, High availability sample programs, demonstrates client reconnection using a single instance queue manager for simplicity. You can use the same samples with multi-instance queue managers to verify a multi-instance queue manager configuration.

This example uses the multi-instance configuration described in “Create a multi-instance queue manager on domain controllers” on page 354. Use the configuration to verify that the multi-instance queue manager switches over to the standby instance. Stop the queue manager with the **endmqm** command and use the `-s`, switchover, option. The client programs reconnect to the new queue manager instance and continue to work with the new instance after a slight delay.

The client is installed in a 400 MB VMware image that is running Windows XP Service Pack 2. For security reasons, it is connected on the same VMware host-only network as the domain servers that are running the multi-instance queue manager. It is sharing the `/MQHA` folder, which contains the client connection table, to simplify configuration.

Verifying failover using WebSphere MQ Explorer

Before using the sample applications to verify failover, run the WebSphere MQ Explorer on each server. Add both queue manager instances to each explorer using the **Add Remote Queue Manager > Connect directly to a multi-instance queue manager** wizard. Ensure that both instances are running, permitting standby. Close the window running the VMware image with the active instance, virtually powering off the server, or stop the active instance, allowing switchover to standby instance and reconnectable clients to reconnect.

Note: If you power off the server, make sure that it is not the one hosting the `MQHA` folder!

Note: The **Allow switchover to a standby instance** option might not be available on the **Stop Queue Manager** dialog. The option is missing because the queue manager is running as a single instance queue manager. You must have started it without the **Permit a standby instance** option. If your request to stop the queue manager is rejected, look at the **Details** window, possibly there is no standby instance running.

Verifying failover using the sample programs

Choose a server to run the active instance

You might have chosen one of the servers to host the `MQHA` directory or file system. If you plan to test failover by closing the VMware window running the active server, make sure that it is not the one hosting `MQHA`!

On the server running the active queue manager instance

1. Modify *ipaddr1* and *ipaddr2* and save the following commands in `N:\hasample.tst`.

```
DEFINE QLOCAL(SOURCE) REPLACE
DEFINE QLOCAL(TARGET) REPLACE
DEFINE CHANNEL(CHANNEL1) CHLTYPE(SVRCONN)   TRPTYPE(TCP) +
       MCAUSER(' ') REPLACE
DEFINE CHANNEL(CHANNEL1) CHLTYPE(CLNTCONN)  TRPTYPE(TCP) +
       CONNAME('ipaddr1(1414),ipaddr2(1414)') QMNAME(QM1) REPLACE
START  CHANNEL(CHANNEL1)
DEFINE LISTENER(LISTENER.TCP) TRPTYPE(TCP) CONTROL(QMGR)
DISPLAY LISTENER(LISTENER.TCP) CONTROL
DISPLAY LSSTATUS(LISTENER.TCP) STATUS
```

Note: By leaving the **MCAUSER** parameter blank, the client user ID is sent to the server. The client user ID must have the correct permissions on the servers. An alternative is to set the **MCAUSER** parameter in the SVRCONN channel to the user ID you have configured on the server.

2. Open a command prompt with the path `N:\` and run the command:

```
runmqsc -m QM1 < hasample.tst
```

3. Verify that the listener is running and has queue manager control, either by inspecting the output of the **runmqsc** command.

```
 LISTENER(LISTENER.TCP)CONTROL(QMGR)
 LISTENER(LISTENER.TCP)STATUS(RUNNING)
```

   Or, using the WebSphere MQ Explorer that the TCPIP listener is running and has `Control = Queue Manager`.

On the client

1. Map the shared directory `C:\MQHA` on the server to `N:\` on the client.
2. Open a command prompt with the path `N:\` . Set the environment variable `MQCHLLIB` to point to the client channel definition table (CCDT) on the server:

```
SET MQCHLLIB=N:\data\QM1\@ipcc
```

3. At the command prompt type the commands:

```
start amqsghac TARGET QM1
start amqsmhac -s SOURCE -t TARGET -m QM1
start amqsphac SOURCE QM1
```

   Note: If you have problems, start the applications at a command prompt so that the reason code is printed out on the console, or look at the AMQERR01.LOG file in the `N:\data\QM1\errors` folder.

On the server running the active queue manager instance

1. Either:
   - Close the window running the VMware image with the active server instance.
   - Using the WebSphere MQ Explorer, stop the active queue manager instance, allowing switchover to the standby instance and instructing reconnectable clients to reconnect.
2. The three clients eventually detect the connection is broken, and then reconnect. In this configuration, if you close the server window, it is taking about seven minutes for all three connections to be re-established. Some connections are reestablished well before others.

Results

```
N:\>amqsphac SOURCE QM1
Sample AMQSPHAC start
target queue is SOURCE
message <Message 1>
message <Message 2>
message <Message 3>
message <Message 4>
message <Message 5>
17:05:25 : EVENT : Connection Reconnecting (Delay: 0ms)
17:05:47 : EVENT : Connection Reconnecting (Delay: 0ms)
17:05:52 : EVENT : Connection Reconnected
message <Message 6>
message <Message 7>
message <Message 8>
message <Message 9>
```

```
N:\>amqsmhac -s SOURCE -t TARGET -m QM1
Sample AMQSMHA0 start

17:05:25 : EVENT : Connection Reconnecting (Delay: 97ms)
17:05:48 : EVENT : Connection Reconnecting (Delay: 0ms)
17:05:53 : EVENT : Connection Reconnected
```

```
N:\>amqsghac TARGET QM1
Sample AMQSGHAC start
message <Message 1>
message <Message 2>
message <Message 3>
message <Message 4>
message <Message 5>
17:05:25 : EVENT : Connection Reconnecting (Delay: 156ms)
17:05:47 : EVENT : Connection Reconnecting (Delay: 0ms)
17:05:52 : EVENT : Connection Reconnected
message <Message 6>
message <Message 7>
message <Message 8>
message <Message 9>
```

Secure shared queue manager data and log directories and files on Windows
This topic describes how you can secure a shared location for queue manager data and log files using a global alternative security group. You can share the location between different instances of a queue manager running on different servers.

Typically you do not set up a shared location for queue manager data and log files. When you install IBM WebSphere MQ for Windows, the installation program creates a home directory of your choosing for any queue managers that are created on that server. It secures the directories with the local `mqm` group, and configures a user ID for the IBM IBM WebSphere MQ service to access the directories.

When you secure a shared folder with a security group, a user that is permitted to access the folder must have the credentials of the group. Suppose that a folder on a remote file server is secured with the local `mqm` group on a server called *`mars`*. Make the user that runs queue manager processes a member of the local `mqm` group on *`mars`*. The user has the credentials that match the credentials of the folder on the remote file server. Using those credentials, the queue manager is able to access its data and logs files in the folder. The user that runs queue manager processes on a different server is a member of a different local `mqm` group which does not have matching credentials. When the queue manager runs on a different server to *`mars`*, it cannot access the data and log files it created when it ran on *`mars`* . Even if you make the user a domain user, it has different credentials, because it must acquire the credentials from the local `mqm` group on *`mars`*, and it cannot do that from a different server.

Providing the queue manager with a global alternative security group solves the problem; see Figure 64 on page 364 . Secure a remote folder with a global group. Pass the name of the global group to the queue manager when you create it on *`mars`*. Pass the global group name as the alternative security group using the `-a[r]` parameter on the **`crtmqm`** command. If you transfer the queue manager to run on a different server, the name of the security group is transferred with it. The name is transferred in the **AccessMode** stanza in the `qm.ini` file as a `SecurityGroup`; for example:

```
AccessMode:
   SecurityGroup=wmq\wmq
```

The **AccessMode** stanza in the `qm.ini` also includes the `RemoveMQMAccess`; for example:

```
AccessMode:
   RemoveMQMAccess=<true\false>
```

If this attribute is specified with value `true`, and an access group has also been given, the local mqm group is not granted access to the queue manager data files.

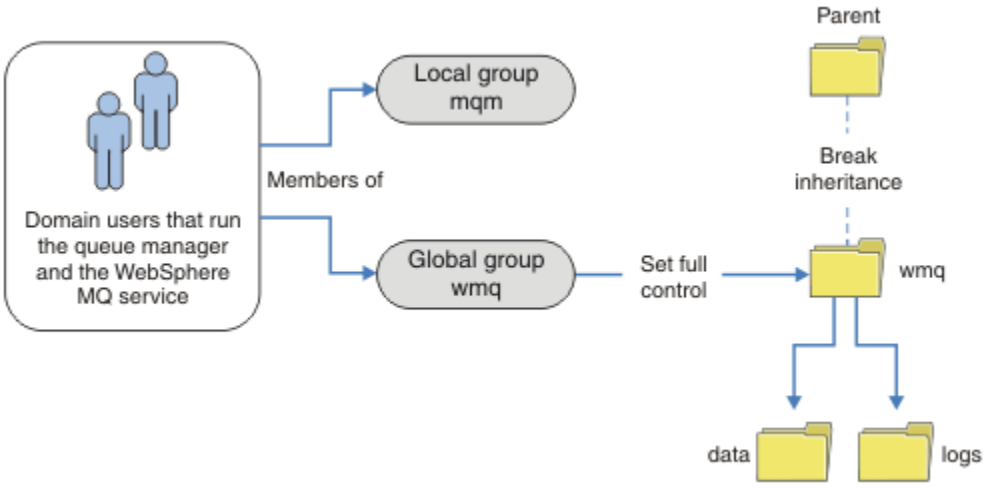


Figure 64. Securing queue manager data and logs using an alternative global security group (1)

For the user ID that queue manager processes are to run with to have the matching credentials of the global security group, the user ID must also have global scope. You cannot make a local group or principal a member of a global group. In Figure 64 on page 364 , the users that run the queue manager processes are shown as domain users.

If you are deploying many IBM WebSphere MQ servers, the grouping of users in Figure 64 on page 364 is not convenient. You would need to repeat the process of adding users to local groups for every IBM WebSphere MQ server. Instead, create a `Domain mqm` global group on the domain controller, and make the users that run IBM WebSphere MQ members of the `Domain mqm` group; see Figure 65 on page 364 . When you install IBM WebSphere MQ as a domain installation, the "Prepare IBM WebSphere MQ" wizard automatically makes the `Domain mqm` group a member of the local `mqm` group. The same users are in both the global groups `Domain mqm` and `wmq`.

Tip: The same users can run IBM WebSphere MQ on different servers, but on an individual server you must have different users to run IBM WebSphere MQ as a service, and run interactively. You must also have different users for every installation on a server. Typically, therefore `Domain mqm` contains a number of users.

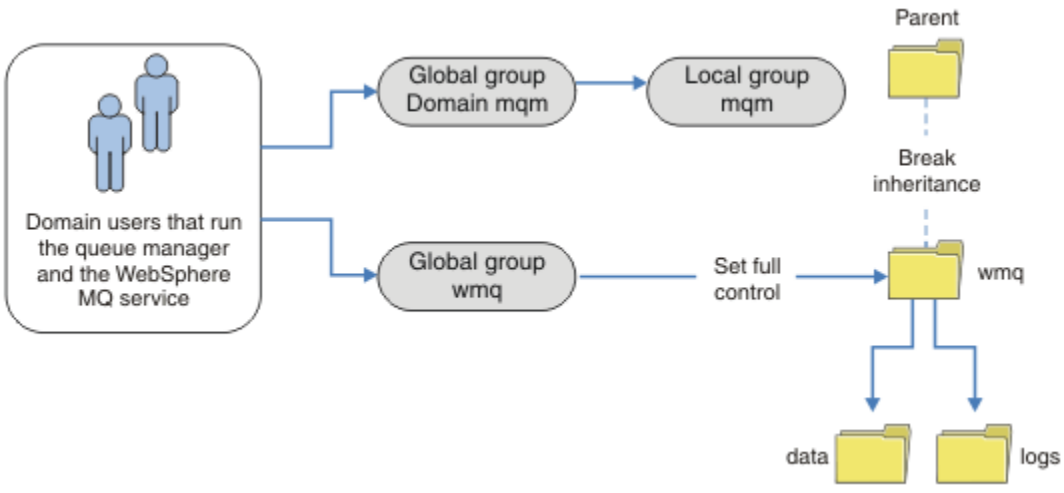


Figure 65. Securing queue manager data and logs using an alternative global security group (2)

The organization in Figure 65 on page 364 is unnecessarily complicated as it stands. The arrangement has two global groups with identical members. You might simplify the organization, and define only one global group; see Figure 66 on page 365.

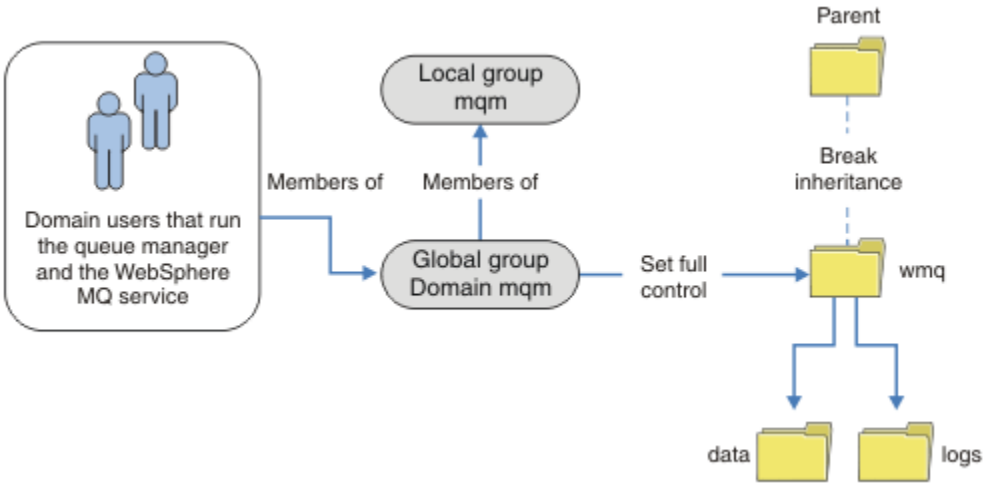


Figure 66. Securing queue manager data and logs using an alternative global security group (3)

Alternatively, you might need a finer degree of access control, with different queue managers restricted to being able to access different folders; see Figure 67 on page 366. In Figure 67 on page 366, two groups of domain users are defined, in separate global groups to secure different queue manager log and data files. Two different local `mqm` groups are shown, which must be on different IBM WebSphere MQ servers. In this example, the queue managers are partitioned into two sets, with different users allocated to the two sets. The two sets might be test and production queue managers. The alternate security groups are called `wmq1` and `wmq2` . You must manually add the global groups `wmq1` and `wmq2` to the correct queue managers according to whether they are in the test or production department. The configuration cannot take advantage that the installation of IBM WebSphere MQ propagates `Domain mqm` to the local `mqm` group as in Figure 66 on page 365, because there are two groups of users.

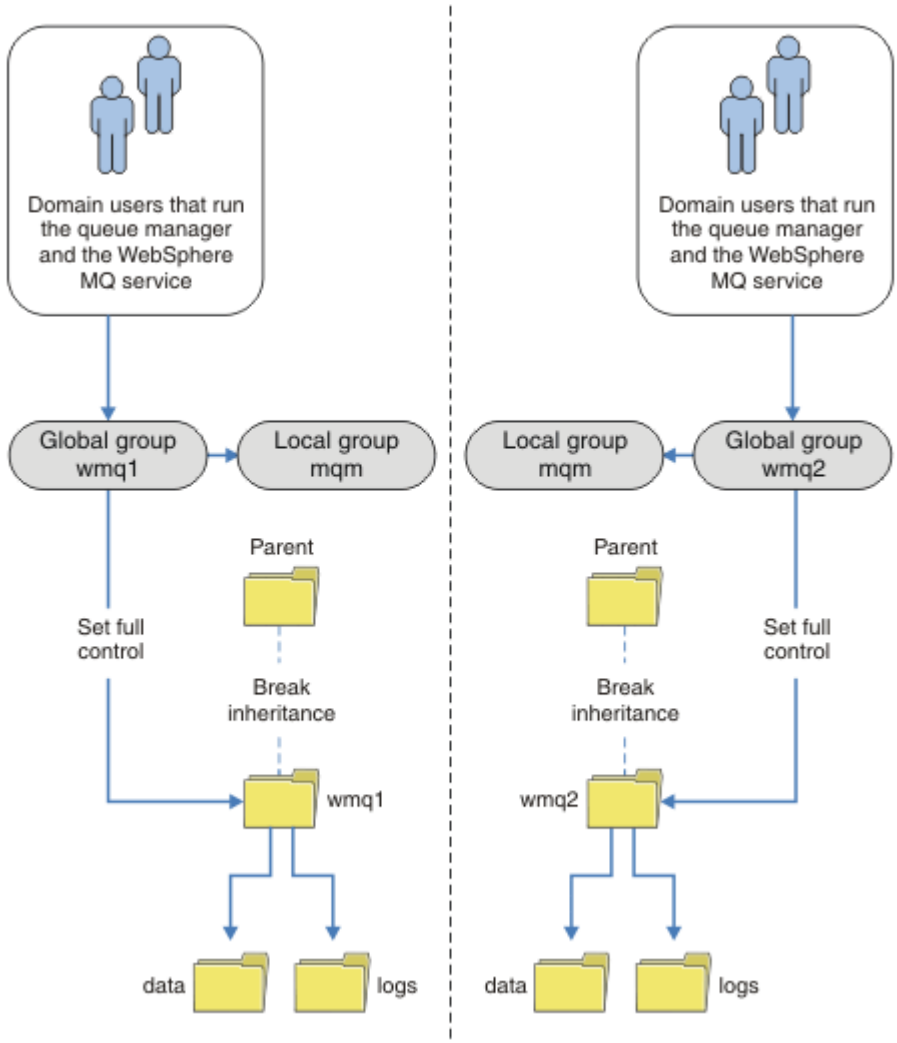


Figure 67. Securing queue manager data and logs using an alternative global security principal (4)

An alternative way to partition two departments would be to place them in two windows domains. In that case, you could return to using the simpler model shown in Figure 66 on page 365.

Secure unshared queue manager data and log directories and files on Windows
This topic describes how you can secure an alternative location for queue manager data and log files, both by using the local `mqm` group and an alternative security group.

Typically you do not set up an alternative location for queue manager data and log files. When you install IBM WebSphere MQ for Windows, the installation program creates a home directory of your choosing for any queue managers that are created. It secures the directories with the local `mqm` group, and configures a user ID for the IBM IBM WebSphere MQ service to access the directories.

Two examples demonstrate how to configure access control for IBM WebSphere MQ. The examples show how to create a queue manager with its data and logs in directories that are not on the data and log paths created by the installation. In the first example, “Reading and writing data and log files authorized by the local mqm group” on page 368, you permit access to the queue and log directories by authorizing by the local `mqm` group. The second example, “Reading and writing data and log files authorized by an alternative local security group” on page 371, differs in that access to the directories is authorized by an alternative security group. When the directories are accessed by a queue manager running on only one server, securing the data and log files with the alternative security group gives you the choice of securing different queue managers with different local groups or principals. When the directories are accessed by a queue manager running on different servers, such as with a multi-instance queue manager, securing the

data and log files with the alternate security group is the only choice; see “Secure shared queue manager data and log directories and files on Windows” on page 363.

Configuring the security permissions of queue manager data and log files is not a common task on Windows. When you install IBM WebSphere MQ for Windows, you either specify directories for queue manager data and logs, or accept the default directories. The installation program automatically secures these directories with the local `mqm` group, giving it full control permission. The installation process makes sure the user ID that runs queue managers is a member of the local `mqm` group. You can modify the other access permissions on the directories to meet your access requirements.

If you move the data and log files directory to new locations, you must configure the security of the new locations. You might change the location of the directories if you back up a queue manager and restore it onto a different computer, or if you change the queue manager to be a multi-instance queue manager. You have a choice of two ways of securing the queue manager data and log directories in their new location. You can secure the directories by restricting access to the local `mqm` group, or you can restrict access to any security group of your choosing.

It takes the least number of steps to secure the directories using the local `mqm` group. Set the permissions on the data and log directories to allow the local `mqm` group full control. A typical approach is to copy the existing set of permissions, removing inheritance from the parent. You can then remove or restrict the permissions of other principals.

If you run the queue manager under a different user ID to the service set up by the Prepare IBM WebSphere MQ wizard, that user ID must be a member of the local `mqm` group. The task, “Reading and writing data and log files authorized by the local mqm group” on page 368, takes you through the steps.

You can also secure queue manager data and log files using an alternative security group. The process of securing the queue manager data and log files with the alternative security group has a number of steps that refer to Figure 68 on page 367. The local group, `wmq`, is an example of an alternative security group.

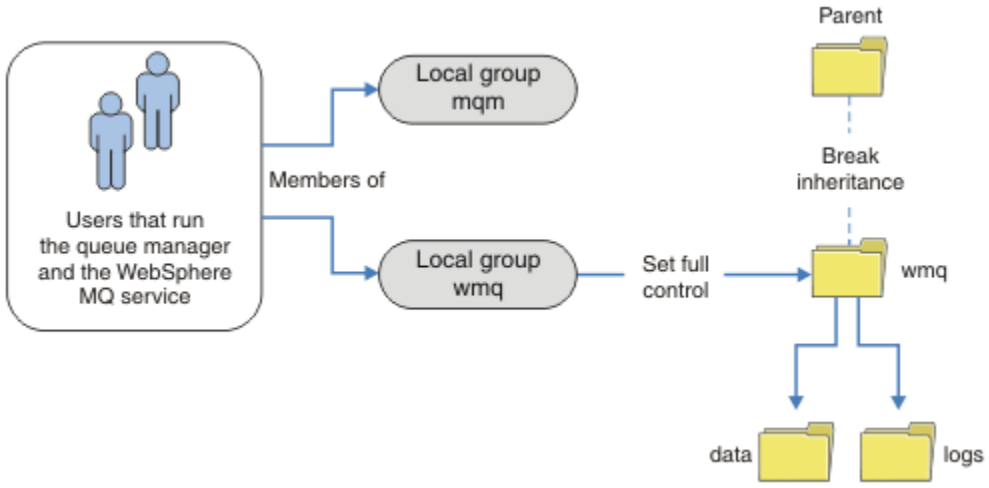


Figure 68. Securing queue manager data and logs using an alternative local security group, `wmq`

1. Either create separate directories for the queue manager data and logs, a common directory, or a common parent directory.
2. Copy the existing set of inherited permissions for the directories, or parent directory, and modify them according to your needs.
3. Secure the directories that are to contain the queue manager and logs by giving the alternative group, `wmq`, full control permission to the directories.
4. Give all user IDs that run queue manager processes the credentials of the alternative security group or principal:

a. If you define a user as the alternative security principal, the user must be same user as the queue manager is going to run under. The user must be a member of the local `mqm` group.

b. If you define a local group as the alternative security group, add the user that the queue manager is going to run under to the alternative group. The user must also be a member of the local `mqm` group.

c. If you define an global group as the alternative security group, then see "Secure shared queue manager data and log directories and files on Windows" on page 363.

5. Create the queue manager specifying the alternative security group or principal on the **crtmqm** command, with the `-a` parameter.

Reading and writing data and log files authorized by the local mqm group
The task illustrates how to create a queue manager with its data and logs files stored in any directory of your choosing. Access to the files is secured by the local `mqm` group. The directory is not shared.

Before you begin

1. Install IBM WebSphere MQ for Windows as the primary installation.
2. Run the "Prepare IBM WebSphere MQ" wizard. For this task, configure the installation either to run with a local user ID, or a domain user ID. Eventually, to complete all the tasks in "Windows domains and multi-instance queue managers" on page 339, the installation must be configured for a domain.
3. Log on with Administrator authority to perform the first part of the task.

About this task

This task is one of a set of related tasks that illustrate accessing queue manager data and log files. The tasks show how to create a queue manager authorized to read and write data and log files that are stored in a directory of your choosing. They accompany the task, "Windows domains and multi-instance queue managers" on page 339.

On Windows, you can create the default data and log paths for a IBM WebSphere MQ for Windows in any directories of your choosing. The installation and configuration wizard automatically gives the local `mqm` group, and the user ID that is running the queue manager processes, access to the directories. If you create a queue manager specifying different directories for queue manager data and log files, you must configure full control permission to the directories.

In this example, you give the queue manager full control over its data and log files by giving the local `mqm` group permission to the directory *c:\wmq*.

The **crtmqm** command creates a queue manager that starts automatically when the workstation starts using the IBM IBM WebSphere MQ service.

The task is illustrative; it uses specific values that you can change. The values you can change are in italics. At the end of the task, follow the instructions to remove all the changes you made.

Procedure

1. Open a command prompt.
2. Type the command:

```
md c:\wmq\data , c:\wmq\logs
```

3. Set the permissions on the directories to allow the local `mqm` group read and write access.

```
cacls c:\wmq /T /E /G mqm:F
```

The system response:

```
processed dir: c:\wmq
processed dir: c:\wmq\data
processed dir: c:\wmq\logs
```

4. Optional: Switch to a user ID that is a member of the local `mqm` group.

   You can continue as Administrator, but for a realistic production configuration, continue with a user ID with more restricted rights. The user ID must at least be a member of the local `mqm` group.
   If the IBM WebSphere MQ installation is configured as part of a domain, make the user ID a member of the `Domain mqm` group. The "Prepare IBM WebSphere MQ" wizard makes the `Domain mqm` global group a member of the local `mqm` group, so you do not have to make the user ID directly a member of the local `mqm` group.
5. Create the queue manager.

   ```
   crtmqm -sax -u SYSTEM.DEAD.LETTER.QUEUE -md c:\wmq\data -ld c:\wmq\logs  QMGR
   ```

   The system response:

   ```
   WebSphere MQ queue manager created.
   Directory 'c:\wmq\data\QMGR' created.
   The queue manager is associated with installation '1'
   Creating or replacing default objects for queue manager 'QMGR'
   Default objects statistics : 74 created. 0 replaced.
   Completing setup.
   Setup completed.
   ```
6. Check that the directories created by the queue manager are in the *c:\wmq* directory.

   ```
   dir c:\wmq /D /B /S
   ```
7. Check that the files have read and write, or full control permission for the local `mqm` group.

   ```
   cacls c:\wmq\*.*
   ```

What to do next

Test the queue manager by putting and getting a message to a queue.

1. Start the queue manager.

   ```
   strmqm QMGR
   ```

   The system response:

   ```
   WebSphere MQ queue manager 'QMGR' starting.
   The queue manager is associated with installation '1'.
   5 log records accessed on queue manager 'QMGR' during the log
   replay phase.
   Log replay for queue manager 'QMGR' complete.
   Transaction manager state recovered for queue manager 'QMGR'.
   WebSphere MQ queue manager 'QMGR' started using V7.1.0.0.
   ```
2. Create a test queue.

   ```
   echo define qlocal(QTEST) | runmqsc QMGR
   ```

   The system response:

   ```
   5724-H72 (C) Copyright IBM Corp. 1994, 2025. ALL RIGHTS RESERVED.
   Starting MQSC for queue manager QMGR.


        1 : define qlocal(QTEST)
   AMQ8006: WebSphere MQ queue created.
   One MQSC command read.
   No commands have a syntax error.
   All valid MQSC commands were processed.
   ```

3. Put a test message using the sample program **amqsput**.

   ```
   echo 'A test message' | amqsput QTEST QMGR
   ```

   The system response:

   ```
   Sample AMQSPUT0 start
   target queue is QTEST
   Sample AMQSPUT0 end
   ```

4. Get the test message using the sample program **amqsget**.

   ```
   amqsget QTEST QMGR
   ```

   The system response:

   ```
   Sample AMQSGET0 start
   message <A test message>
   Wait 15 seconds ...
   no more messages
   Sample AMQSGET0 end
   ```

5. Stop the queue manager.

   ```
   endmqm -i QMGR
   ```

   The system response:

   ```
   WebSphere MQ queue manager 'QMGR' ending.
   WebSphere MQ queue manager 'QMGR' ended.
   ```

6. Delete the queue manager.

   ```
   dltmqm QMGR
   ```

   The system response:

   ```
   WebSphere MQ queue manager 'QMGR' deleted.
   ```

7. Delete the directories you created.

   Tip: Add the /Q option to the commands to prevent the command prompting to delete each file or directory.

   ```
   del /F /S C:\wmq\*.*
   rmdir /S C:\wmq
   ```

Related concepts

“Windows domains and multi-instance queue managers” on page 339
A multi-instance queue manager on Windows requires its data and logs to be shared. The share must be accessible to all instances of the queue manager running on different servers or workstations. Configure the queue managers and share as part of a Windows domain. The queue manager can run on a domain workstation or server, or on the domain controller.

Related tasks

“Reading and writing data and log files authorized by an alternative local security group” on page 371
This task shows how to use the -a flag on the **crtmqm** command. The flag provides the queue manager with an alternative local security group to give it access to its log and data files.

“Reading and writing shared data and log files authorized by an alternative global security group” on page 351

“Create a multi-instance queue manager on domain workstations or servers” on page 340

Reading and writing data and log files authorized by an alternative local security group
This task shows how to use the -a flag on the **crtmqm** command. The flag provides the queue manager with an alternative local security group to give it access to its log and data files.

Before you begin

1. Install IBM WebSphere MQ for Windows as the primary installation.
2. Run the "Prepare IBM WebSphere MQ" wizard. For this task, configure the installation either to run with a local user ID, or a domain user ID. Eventually, to complete all the tasks in “Windows domains and multi-instance queue managers” on page 339, the installation must be configured for a domain.
3. Log on with Administrator authority to perform the first part of the task.

About this task

This task is one of a set of related tasks that illustrate accessing queue manager data and log files. The tasks show how to create a queue manager authorized to read and write data and log files that are stored in a directory of your choosing. They accompany the task, “Windows domains and multi-instance queue managers” on page 339.

On Windows, you can create the default data and log paths for a IBM WebSphere MQ for Windows in any directories of your choosing. The installation and configuration wizard automatically gives the local `mqm` group, and the user ID that is running the queue manager processes, access to the directories. If you create a queue manager specifying different directories for queue manager data and log files, you must configure full control permission to the directories.

In this example, you provide the queue manager with an alternative security local group that has full control authorization to the directories. The alternative security group gives the queue manager permission to manage files in the directory. The primary purpose of the alternate security group is to authorize an alternate security global group. Use an alternate security global group to set up a multi-instance queue manager. In this example, you configure a local group to familiarize yourself with the use of an alternate security group without installing IBM WebSphere MQ in a domain. It is unusual to configure a local group as an alternative security group.

The **crtmqm** command creates a queue manager that starts automatically when the workstation starts using the IBM IBM WebSphere MQ service.

The task is illustrative; it uses specific values that you can change. The values you can change are in italics. At the end of the task, follow the instructions to remove all the changes you made.

Procedure

1. Set up an alternative security group.

   The alternative security group is typically a domain group. In the example, you create a queue manager that uses a local alternate security group. With a local alternate security group, you can do the task with an IBM WebSphere MQ installation that is not part of a domain.

   a) Run the **lusrmgr.msc** command to open the Local Users and Groups window.

   b) Right-click **Groups** > **New Group...**

   c) In the **Group name** field, type *altmqm* and click **Create** > **Close**.

   d) Identify the user ID that runs the IBM IBM WebSphere MQ service.

      i) Click **Start** > **Run...**, type `services.msc` and click **OK**.

      ii) Click the IBM IBM WebSphere MQ service in the list of services, and click the Log On tab.

      iii) Remember the user ID and close the Services Explorer.

   e) Add the user ID that runs the IBM IBM WebSphere MQ service to the *altmqm* group. Also add the user ID that you log on with to create a queue manager, and run it interactively.

Windows checks the authority of the queue manager to access the data and logs directories by checking the authority of the user ID that is running queue manager processes. The user ID must be a member, directly or indirectly through a global group, of the *altmqm* group that authorized the directories.

If you installed IBM WebSphere MQ as part of a domain, and are going to do the tasks in "Create a multi-instance queue manager on domain workstations or servers" on page 340, the domain user IDs created in "Creating an Active Directory and DNS domain for IBM WebSphere MQ" on page 343 are *wmquser1* and *wmquser2*.

If you did not install the queue manager as part of a domain, the default local user ID that runs the IBM IBM WebSphere MQ service is MUSR_MQADMIN. If you intend to do the tasks without Administrator authority, create a user that is a member of the local mqm group.

Follow these steps to add *wmquser1* and *wmquser2* to *altmqm*. If your configuration is different, substitute your names for the user IDs and group.

i) In the list of groups right-click **altmqm** > **Properties** > **Add...**.

ii) In the Select Users, Computers, or Groups window type *wmquser1; wmquser2* and click **Check Names**.

iii) Type the name and password of a domain administrator in the Windows Security window, then click **OK** > **OK** > **Apply** > **OK**.

2. Open a command prompt.
3. Restart the IBM IBM WebSphere MQ service.

   You must restart the service so that the user ID it runs under acquires the additional security credentials you configured for it.

   Type the commands:

   ```
   endmqsvc
   strmqsvc
   ```

   The system responses:

   ```
   5724-H72 (C) Copyright IBM Corp. 1994, 2025. ALL RIGHTS RESERVED.
   The MQ service for installation 'Installation1' ended successfully.
   And:
   5724-H72 (C) Copyright IBM Corp. 1994, 2025. ALL RIGHTS RESERVED.
   The MQ service for installation 'Installation1' started successfully.
   ```

4. Type the command:

   ```
   md c:\wmq\data , c:\wmq\logs
   ```

5. Set the permissions on the directories to allow the local user *user* read and write access.

   ```
   cacls c:\wmq /T /E /G altmqm:F
   ```

   The system response:

   ```
   processed dir: c:\wmq
   processed dir: c:\wmq\data
   processed dir: c:\wmq\logs
   ```

6. Optional: Switch to a user ID that is a member of the local mqm group.

   You can continue as Administrator, but for a realistic production configuration, continue with a user ID with more restricted rights. The user ID must at least be a member of the local mqm group.

   If the IBM WebSphere MQ installation is configured as part of a domain, make the user ID a member of the Domain mqm group. The "Prepare IBM WebSphere MQ" wizard makes the Domain mqm global group a member of the local mqm group, so you do not have to make the user ID directly a member of the local mqm group.

7. Create the queue manager.

```
crtmqm -a altmqm -sax -u SYSTEM.DEAD.LETTER.QUEUE -md c:\wmq\data -ld c:\wmq\logs  QMGR
```

The system response:

```
WebSphere MQ queue manager created.
Directory 'c:\wmq1\data\QMGR' created.
The queue manager is associated with installation '1'
Creating or replacing default objects for queue manager 'QMGR'
Default objects statistics : 74 created. 0 replaced.
Completing setup.
Setup completed.
```

8. Check that the directories created by the queue manager are in the *c:\wmq* directory.

```
dir c:\wmq /D /B /S
```

9. Check that the files have read and write, or full control permission for the local mqm group.

```
cacls c:\wmq\*.*
```

What to do next

Test the queue manager by putting and getting a message to a queue.

1. Start the queue manager.

```
strmqm QMGR
```

The system response:

```
WebSphere MQ queue manager 'QMGR' starting.
The queue manager is associated with installation '1'.
5 log records accessed on queue manager 'QMGR' during the log
replay phase.
Log replay for queue manager 'QMGR' complete.
Transaction manager state recovered for queue manager 'QMGR'.
WebSphere MQ queue manager 'QMGR' started using V7.1.0.0.
```

2. Create a test queue.

```
echo define qlocal(QTEST) | runmqsc QMGR
```

The system response:

```
5724-H72 (C) Copyright IBM Corp. 1994, 2025. ALL RIGHTS RESERVED.
Starting MQSC for queue manager QMGR.


     1 : define qlocal(QTEST)
AMQ8006: WebSphere MQ queue created.
One MQSC command read.
No commands have a syntax error.
All valid MQSC commands were processed.
```

3. Put a test message using the sample program **amqsput**.

```
echo 'A test message' | amqsput QTEST QMGR
```

The system response:

```
Sample AMQSPUT0 start
target queue is QTEST
Sample AMQSPUT0 end
```

4. Get the test message using the sample program **amqsget**.

   ```
   amqsget QTEST QMGR
   ```

   The system response:

   ```
   Sample AMQSGET0 start
   message <A test message>
   Wait 15 seconds ...
   no more messages
   Sample AMQSGET0 end
   ```

5. Stop the queue manager.

   ```
   endmqm -i QMGR
   ```

   The system response:

   ```
   WebSphere MQ queue manager 'QMGR' ending.
   WebSphere MQ queue manager 'QMGR' ended.
   ```

6. Delete the queue manager.

   ```
   dltmqm QMGR
   ```

   The system response:

   ```
   WebSphere MQ queue manager 'QMGR' deleted.
   ```

7. Delete the directories you created.

   Tip: Add the /Q option to the commands to prevent the command prompting to delete each file or directory.

   ```
   del /F /S C:\wmq\*.*
   rmdir /S C:\wmq
   ```

Create a multi-instance queue manager on Linux
An example shows how to set up a multi-instance queue manager on Linux. The setup is small to illustrate the concepts involved. The example is based on Linux Red Hat Enterprise 5. The steps differ on other UNIX platforms.

The example is set up on a 2 GHz notebook computer with 3 GB RAM running Windows XP Service Pack 2. Two VMware virtual machines, Server1 and Server2, run Linux Red Hat Enterprise 5 in 640 MB images. Server1 hosts the network file system (NFS), the queue manager logs and an HA instance. It is not usual practice for the NFS server also to host one of the queue manager instances; this is to simplify the example. Server2 mounts Server1's queue manager logs with a standby instance. A WebSphere MQ MQI client is installed on an additional 400 MB VMware image that runs Windows XP Service Pack 2 and runs the sample high availability applications. All the virtual machines are configured as part of a VMware host-only network for security reasons.

Note: You should put only queue manager data on an NFS server. On the NFS, use the following three options with the mount command to make the system secure:

noexec
By using this option, you stop binary files from being run on the NFS, which prevents a remote user from running unwanted code on the system.

nosuid
By using this option, you prevent the use of the set-user-identifier and set-group-identifier bits, which prevents a remote user from gaining higher privileges.

nodev
By using this option, you stop character and block special devices from being used or defined, which prevents a remote user from getting out of a chroot jail.

Example

Table 30. Illustrative multi-instance queue manager configuration on Linux

Server1	Server2
Log in as *root*	
Follow the instructions in Installing IBM WebSphere MQ to install WebSphere MQ, create the mqm user and group if these do not exist, and define `/var/mqm`. Check what `uid` and `gid` in `/etc/passwd` on the first machine displays for mqm; for example, `mqm:x:501:100:MQ User:/var/mqm:/bin/bash` Match the `uid` and `gid` for mqm in `/etc/passwd` on the second machine to ensure that these values are identical. Reboot this machine if you have to change the values.	
Complete the task Verifying shared file system behavior to check that the file system supports multi-instance queue managers.	
Create log and data directories in a common folder, `/MQHA`, that is to be shared. For example: 1. **mkdir** */MQHA* 2. **mkdir** */MQHA/logs* 3. **mkdir** */MQHA/qmgrs*	Create the folder, `/MQHA`, to mount the shared file system. Keep the path the same as on Server1; for example: 1. **mkdir** */MQHA*
Ensure that the `MQHA` directories are owned by user and group mqm, and the access permissions are set to `rwx` for user and group; for example **ls -al** displays, `drwxrwxr-x mqm mqm 4096 Nov 27 14:38 MQDATA` 1. **chown -R** `mqm:mqm` */MQHA* 2. **chmod -R** `ug+rwx` */MQHA*	
Create the queue manager: **crtmqm -ld** */MQHA/logs* **-md** */MQHA/qmgrs* *QM1*	
Add[2] `/MQHA *(rw,sync,no_wdelay,fsid=0)` to `/etc/exports`	
Start the NFS daemon: */etc/init.d/***nfs** `start`	Mount the exported file system `/MQHA`: **mount -t** `nfs4` **-o** `hard,intr` *Server1:/ /MQHA*

[2] The '*' allows all machines that can reach this one mount /MQHA for read/write. Restrict access on a production machine.

Table 30. Illustrative multi-instance queue manager configuration on Linux (continued)

Server1	Server2

Copy the queue manager configuration details from Server1:
`dspmqinf -o command QM1`
and copy the result to the clipboard:
`addmqinf -s QueueManager`
`-v Name=QM1`
`-v Directory=QM1`
`-v Prefix=/var/mqm`
`-v DataPath=/MQHA/qmgrs/QM1`
	Paste the queue manager configuration command into Server2:
	`addmqinf -s QueueManager`
	`-v Name=QM1`
	`-v Directory=QM1`
	`-v Prefix=/var/mqm`
	`-v DataPath=/MQHA/qmgrs/QM1`

Start the queue manager instances, in either order, with the **-x** parameter: **strmqm -x** QM1 The command used to start the queue manager instances must be issued from the same IBM WebSphere MQ installation as the **addmqinf** command. To start and stop the queue manager from a different installation, you must first set the installation associated with the queue manager using the **setmqm** command. For more information, see setmqm.	

Verifying the multi-instance queue manager on Linux

Use the sample programs **amqsghac**, **amqsphac** and **amqsmhac** to verify a multi-instance queue manager configuration. This topic provides an example configuration to verify a multi-instance queue manager configuration on Linux Red Hat Enterprise 5.

The high availability sample programs use automatic client reconnection. When the connected queue manager fails, the client attempts to reconnect to a queue manager in the same queue manager group. The description of the samples, High availability sample programs, demonstrates client reconnection using a single instance queue manager for simplicity. You can use the same samples with multi-instance queue managers to verify a multi-instance queue manager configuration.

The example uses the multi-instance configuration described in “Create a multi-instance queue manager on Linux” on page 374. Use the configuration to verify that the multi-instance queue manager switches over to the standby instance. Stop the queue manager with the **endmqm** command and use the `-s`, switchover, option. The client programs reconnect to the new queue manager instance and continue to work with the new instance after a slight delay.

In the example, the client is running on a Windows XP Service Pack 2 system. The system is hosting two VMware Linux servers that are running the multi-instance queue manager.

Verifying failover using WebSphere MQ Explorer

Before using the sample applications to verify failover, run the WebSphere MQ Explorer on each server. Add both queue manager instances to each explorer using the **Add Remote Queue Manager > Connect directly to a multi-instance queue manager** wizard. Ensure that both instances are running, permitting standby. Close the window running the VMware image with the active instance, virtually powering off the server, or stop the active instance, allowing switchover to standby instance.

Note: If you power off the server, make sure that it is not the one hosting `/MQHA`!

Note: The **Allow switchover to a standby instance** option might not be available on the **Stop Queue Manager** dialog. The option is missing because the queue manager is running as a single instance queue manager. You must have started it without the **Permit a standby instance** option. If your request to stop the queue manager is rejected, look at the **Details** window, it is possibly because there is no standby instance running.

Verifying failover using the sample programs

Choose a server to be to run the active instance

You might have chosen one of the servers to host the MQHA directory or file system. If you plan to test failover by closing the VMware window running the active server, make sure that it is not the one hosting MQHA!

On the server running the active queue manager instance

Note: Running the SVRCONN channel with the MCAUSER set to mqm, is a convenience to reduce the number of configuration steps in the example. If another user ID is chosen, and your system is set up differently to the one used in the example, you might experience access permission problems. Do not use mqm as a MCAUSER on an exposed system; it is likely to compromise security greatly.

1. Modify *ipaddr1* and *ipaddr2* and save the following commands in /MQHA/hasamples.tst.

```
DEFINE  QLOCAL(SOURCE) REPLACE
DEFINE  QLOCAL(TARGET) REPLACE
DEFINE  CHANNEL(CHANNEL1) CHLTYPE(SVRCONN)   TRPTYPE(TCP) +
        MCAUSER('mqm') REPLACE
DEFINE  CHANNEL(CHANNEL1) CHLTYPE(CLNTCONN)  TRPTYPE(TCP) +
        CONNAME('ipaddr1(1414),ipaddr2
(1414)') QMNAME(QM1) REPLACE
START   CHANNEL(CHANNEL1)
DEFINE  LISTENER(LISTENER.TCP) TRPTYPE(TCP) CONTROL(QMGR)
DISPLAY LISTENER(LISTENER.TCP) CONTROL
START   LISTENER(LISTENER.TCP)
DISPLAY LSSTATUS(LISTENER.TCP) STATUS
```

2. Open a terminal window with the path /MQHA and run the command:

```
runmqsc -m QM1 < hasamples.tst
```

3. Verify that the listener is running and has queue manager control, either by inspecting the output of the **runmqsc** command.

```
 LISTENER(LISTENER.TCP)CONTROL(QMGR)
 LISTENER(LISTENER.TCP)STATUS(RUNNING)
```

   Or, using the WebSphere MQ Explorer that the TCPIP listener is running and has Control = Queue Manager.

On the client

1. Copy the client connection table AMQCLCHL.TAB from /MQHA/qmgrs/QM1.000/@ipcc on the server to C:\ on the client.
2. Open a command prompt with the path C:\ and set the environment variable MQCHLLIB to point to the client channel definition table (CCDT)

```
SET MQCHLLIB=C:\
```

3. At the command prompt type the commands:

```
start amqsghac TARGET QM1
start amqsmhac -s SOURCE -t TARGET -m QM1
start amqsphac SOURCE QM1
```

On the server running the active queue manager instance

1. Either:
   - Close the window running the VMware image with the active server instance.
   - Using the WebSphere MQ Explorer, stop the active queue manager instance, allowing switchover to the standby instance and instructing reconnectable clients to reconnect.
2. The three clients eventually detect the connection is broken, and then reconnect. In this configuration, if you close the server window, it is taking about seven minutes for all three connections to be re-established. Some connections are reestablished well before others.

Results

```
N:\>amqsphac SOURCE QM1
Sample AMQSPHAC start
target queue is SOURCE
message <Message 1>
message <Message 2>
message <Message 3>
message <Message 4>
message <Message 5>
17:05:25 : EVENT : Connection Reconnecting (Delay: 0ms)
17:05:47 : EVENT : Connection Reconnecting (Delay: 0ms)
17:05:52 : EVENT : Connection Reconnected
message <Message 6>
message <Message 7>
message <Message 8>
message <Message 9>
```

```
N:\>amqsmhac -s SOURCE -t TARGET -m QM1
Sample AMQSMHA0 start

17:05:25 : EVENT : Connection Reconnecting (Delay: 97ms)
17:05:48 : EVENT : Connection Reconnecting (Delay: 0ms)
17:05:53 : EVENT : Connection Reconnected
```

```
N:\>amqsghac TARGET QM1
Sample AMQSGHAC start
message <Message 1>
message <Message 2>
message <Message 3>
message <Message 4>
message <Message 5>
17:05:25 : EVENT : Connection Reconnecting (Delay: 156ms)
17:05:47 : EVENT : Connection Reconnecting (Delay: 0ms)
17:05:52 : EVENT : Connection Reconnected
message <Message 6>
message <Message 7>
message <Message 8>
message <Message 9>
```

Deleting a multi-instance queue manager

To delete a multi-instance queue manager completely, you need to use the **dltmqm** command to delete the queue manager, and then remove instances from other servers using either the **rmvmqinf** or **dltmqm** commands.

Run the **dltmqm** command to delete a queue manager that has instances defined on other servers, on any server where that queue manager is defined. You do not need to run the **dltmqm** command on the same server that you created it on. Then run the **rmvmqinf** or **dltmqm** command on all the other servers which have a definition of the queue manager.

You can only delete a queue manager when it is stopped. At the time you delete it no instances are running, and the queue manager, strictly speaking, is neither a single or a multi-instance queue manager; it is simply a queue manager that has its queue manager data and logs on a remote share. When you delete a queue manager, its queue manager data and logs are deleted, and the queue manager stanza is removed from the `mqs.ini` file on the server on which you issued the **dltmqm** command. You need to have access to the network share containing the queue manager data and logs when you delete the queue manager.

On other servers where you have previously created instances of the queue manager there are also entries in the `mqs.ini` files on those servers. You need to visit each server in turn, and remove the queue manager stanza by running the command **rmvmqinf** *Queue manager stanza name*.

On UNIX and Linux systems, if you have placed a common `mqs.ini` file in network storage and referenced it from all the servers by setting the `AMQ_MQS_INI_LOCATION` environment variable on each server, then you need to delete the queue manager from only one of its servers as there is only one `mqs.ini` file to update.

Example

First server
dltmqm QM1

Other servers where instances are defined
rmvmqinf QM1, or

dltmqm QM1

Starting and stopping a multi-instance queue manager

Starting and stopping a queue manager configured either as a single instance or a multi-instance queue manager.

When you have defined a multi-instance queue manager on a pair of servers, you can run the queue manager on either server, either as a single instance queue manager, or as a multi-instance queue manager.

To run a multi-instance queue manager, start the queue manager on one of the servers using the **strmqm** -x *QM1* command; the -x option permits the instance to failover. It becomes the *active instance*. Start the standby instance on the other server using the same **strmqm** -x *QM1* command; the -x option permits the instance to start as a standby.

The queue manager is now running with one active instance that is processing all requests, and a standby instance that is ready to take over if the active instance fails. The active instance is granted exclusive access to the queue manager data and logs. The standby waits to be granted exclusive access to the queue manager data and logs. When the standby is granted exclusive access, it becomes the active instance.

You can also manually switch control to the standby instance by issuing the **endmqm** -s command on the active instance. The **endmqm** -s command shuts down the active instance without shutting down the standby. The exclusive access lock on the queue manager data and logs is released, and the standby takes over.

You can also start and stop a queue manager configured with multiple instances on different servers as a single instance queue manager. If you start the queue manager without using the -x option on the **strmqm** command, the instances of the queue manager configured on other machines are prevented from starting as standby instances. If you attempt to start another instance you receive the response that the queue manager instance is not permitted to run as a standby.

If you stop the active instance of a multi-instance queue manager using the **endmqm** command without the -s option, then the active and standby instances both stop. If you stop the standby instance using the **endmqm** command with the -x option, then it stops being a standby, and the active instance continues running. You cannot issue **endmqm** without the -x option on the standby.

Only two queue manager instances can run at the same time; one is the active instance, and the other is a standby instance. If you start two instances at the same time, WebSphere MQ has no control over which instance becomes the active instance; it is determined by the network file system. The first instance to acquire exclusive access to the queue manager data becomes the active instance.

Note: Before you restart a failed queue manager, you must disconnect your applications from that instance of the queue manager. If you do not, the queue manager might not restart correctly.

Shared file system

A multi-instance queue manager uses a networked file system to manage queue manager instances.

A multi-instance queue manager automates failover using a combination of file system locks and shared queue manager data and logs. Only one instance of a queue manager can have exclusive access to the shared queue manager data and logs. When it gets access it becomes the active instance. The other instance that does not succeed in getting exclusive access waits as a standby instance until the queue manager data and logs become available.

The networked file system is responsible for releasing the locks it holds for the active queue manager instance. If the active instance fails in some way, the networked file system releases the locks it is holding

for the active instance. As soon as the exclusive lock is released, a standby queue manager waiting for the lock attempts to acquire it. If it succeeds, it becomes the active instance and has exclusive access to the queue manager data and logs on the shared file system. It then continues to start.

The related topic, Planning file system support describes how to set up and check that your file system supports multi-instance queue managers.

A multi-instance queue manager does not protect you against a failure in the file system. There are a number of ways to protect your data.

- Invest in reliable storage, such as redundant disk arrays (RAID), and include them in a networked file system that has network resilience.
- Back up WebSphere MQ linear logs to alternative media, and if your primary log media fails, recover using the logs on the alternative media. You can use a backup queue manager to administer this process.

Multiple queue manager instances

A multi-instance queue manager is resilient because it uses a standby queue manager instance to restore queue manager availability after failure.

Replicating queue manager instances is a very effective way to improve the availability of queue manager processes. Using a simple availability model, purely for illustration: if the reliability of one instance of a queue manager is 99% (over one year, cumulative downtime is 3.65 days) then adding another instance of the queue manager increases the availability to 99.99% (over one year, cumulative downtime of about an hour).

This is too simple a model to give you practical numeric estimates of availability. To model availability realistically, you need to collect statistics for the mean time between failures (MTBF) and the mean time to repair (MTTR), and the probability distribution of time between failures and of repair times.

The term, multi-instance queue manager, refers to the combination of active and standby instances of the queue manager that share the queue manager data and logs. Multi-instance queue managers protect you against the failure of queue manager processes by having one instance of the queue manager active on one server, and another instance of the queue manager on standby on another server, ready to take over automatically should the active instance fail.

Failover or switchover

A standby queue manager instance takes over from the active instance either on request (switchover), or when the active instance fails (failover).

- *Switchover* takes place when a standby instance starts in response to the **endmqm** `-s` command being issued to the active queue manager instance. You can specify the **endmqm** parameters `-c`, `-i` or `-p` to control how abruptly the queue manager is stopped.

  Note: Switchover only takes place if a standby queue manager instance is already started. The **endmqm** `-s` command releases the active queue manager lock and permits switchover: it does not start a standby queue manager instance.
- *Failover* occurs when the lock on queue manager data held by the active instance is released because the instance appeared to stop unexpectedly (that is, without an **endmqm** command being issued).

When the standby instance takes over as the active instance, it writes a message to the queue manager error log.

Reconnectable clients are automatically reconnected when a queue manager fails or switches over. You do not need to include the `-r` flag on the **endmqm** command to request client reconnection. Automatic client reconnect is not supported by WebSphere MQ classes for Java.

If you find that you cannot restart a failed instance, even though failover has occurred and the standby instance has become active, check that applications connected locally to the failed instance have not disconnected from the failed instance. Locally connected applications end or disconnect from a failed queue manager instance to ensure that the failed instance can be restarted. Any locally connected applications using shared bindings (which is the default setting) which hold on to a connection to a failed

instance act to prevent the instance from being restarted. If it is not possible to end the locally connected applications, or ensure that they disconnect when the local queue manager instance fails, consider using isolated bindings. Locally connected applications using isolated bindings do not prevent the local queue manager instance from being restarted, even if they do not disconnect.

Channel and client reconnection

Channel and client reconnection is an essential part of restoring message processing after a standby queue manager instance has become active.

Multi-instance queue manager instances are installed on servers with different network addresses. You need to configure IBM WebSphere MQ channels and clients with connection information for all queue manager instances. When a standby takes over, clients and channels are automatically reconnected to the newly active queue manager instance at the new network address. Automatic client reconnect is not supported by WebSphere MQ classes for Java.

The design is different from the way high availability environments such as HA-CMP work. HA-CMP provides a virtual IP address for the cluster and transfer the address to the active server. WebSphere MQ reconnection does not change or reroute IP addresses. It works by reconnecting using the network addresses you have defined in channel definitions and client connections. As an administrator, you need to define the network addresses in channel definitions and client connections to all instances of any multi-instance queue manager. The best way to configure network addresses to a multi-instance queue manager depends on the connection:

Queue manager channels

The CONNAME attribute of channels is a comma-separated list of connection names; for example, `CONNAME('127.0.0.1(1234), 192.0.2.0(4321)')`. The connections are tried in the order specified in the connection list until a connection is successfully established. If no connection is successful, the channel attempts to reconnect.

Cluster channels

Typically, no additional configuration is required to make multi-instance queue managers work in a cluster.

If a queue manager connects to a repository queue manager, the repository discovers the network address of the queue manager. It refers to the CONNAME of the CLUSRCVR channel at the queue manager. On TCPIP, the queue manager automatically sets the CONNAME if you omit it, or configure it to blanks. When a standby instance takes over, its IP address replaces the IP address of the previous active instance as the CONNAME.

If it is necessary, you can manually configure CONNAME with the list of network addresses of the queue manager instances.

Client connections

Client connections can use connection lists, or queue manager groups to select alternative connections. For more information about client reconnection to a multi-instance queue manager see “Automatic client reconnection” on page 304. Clients need to be compiled to run with WebSphere MQ Version 7.0.1 client libraries or better. They must be connected to at least a Version 7.0.1 queue manager.

When failover occurs, reconnection takes some time. The standby queue manager has to complete its startup. The clients that were connected to the failed queue manager have to detect the connection failure, and start a new client connection. If a new client connection selects the standby queue manager that has become newly active, then the client is reconnected to the same queue manager.

If the client is in the middle of an MQI call during the reconnection, it must tolerate an extended wait before the call completes.

If the failure takes place during a batch transfer on a message channel, the batch is rolled back and restarted.

Switching over is faster than failing over, and takes only as long as stopping one instance of the queue manager and starting another. For a queue manager with only few log records to replay, at best switchover might take of the order of a few seconds. To estimate how long failover takes, you need to add the time

that it takes for the failure to be detected. At best the detection takes of the order of 10 seconds, and might be several minutes, depending on the network and the file system.

Application recovery

Application recovery is the automated continuation of application processing after failover. Application recovery following failover requires careful design. Some applications need to be aware failover has taken place.

The objective of application recovery is for the application to continue processing with only a short delay. Before continuing with new processing, the application must back out and resubmit the unit of work that it was processing during the failure.

A problem for application recovery is losing the context that is shared between the WebSphere MQ MQI client and the queue manager, and stored in the queue manager. The WebSphere MQ MQI client restores most of the context, but there are some parts of the context that cannot be reliably restored. The following sections describe some properties of application recovery and how they affect the recovery of applications connected to a multi-instance queue manager.

Transactional messaging

From the perspective of delivering messages, failover does not change the persistent properties of WebSphere MQ messaging. If messages are persistent, and correctly managed within units of work, then messages are not lost during a failover.

From the perspective of transaction processing, transactions are either backed out or committed after failover.

Uncommitted transactions are rolled back. After failover, a reconnectable application receives a `MQRC_BACKED_OUT` reason code to indicate that the transaction has failed, and it needs to roll back the transaction by issuing `MQBACK`. It then needs to restart the transaction again.

Committed transactions are transactions that have reached the second phase of a two-phase commit, or single phase (message only) transactions that have begun `MQCMIT` .

If the queue manager is the transaction coordinator and `MQCMIT` has begun the second phase of its two-phase commit before the failure, the transaction successfully completes. The completion is under the control of the queue manager and continues when the queue manager is running again. In a reconnectable application, the `MQCMIT` call completes normally.

In a single phase commit, which involves only messages, a transaction that has started commit processing completes normally under the control of the queue manager once it is running again. In a reconnectable application, the `MQCMIT` completes normally.

Reconnectable clients can use single phase transactions under the control of the queue manager as the transaction coordinator. The extended transactional client does not support reconnection. If reconnection is requested when the transactional client connects, the connection succeeds, but without the ability to be reconnected. The connection behaves as if it is not reconnectable.

Application restart or resume

Failover interrupts an application. After a failure an application can restart from the beginning, or it can resume processing following the interruption. The latter is called *automatic client reconnection*. Automatic client reconnect is not supported by WebSphere MQ classes for Java.

With a WebSphere MQ MQI client application, you can set a connection option to reconnect the client automatically. The options are `MQCNO_RECONNECT` or `MQCNO_RECONNECT_Q_MGR` . If no option is set, the client does not try to reconnect automatically and the queue manager failure returns `MQRC_CONNECTION_BROKEN` to the client. You might design the client to try and start a new connection by issuing a new `MQCONN` or `MQCONNX` call.

Server programs have to be restarted; they cannot be automatically reconnected by the queue manager at the point they were processing when the queue manager or server failed. WebSphere MQ server programs

are typically not restarted on the standby queue manager instance when a multi-instance queue manager instance fails.

You can automate a WebSphere MQ server program to restart on the standby server in two ways:

1. Package your server application as a queue manager service. It is restarted when the standby queue manager restarts.
2. Write your own failover logic, triggered for example, by the failover log message written by a standby queue manager instance when it starts. The application instance then needs to call MQCONN or MQCONNX after it starts, to create a connection to the queue manager.

Detecting failover

Some applications do need to be aware of failover, others do not. Consider these two examples.

1. A messaging application that gets or receives messages over a messaging channel does not normally require the queue manager at the other end of the channel to be running: it is unlikely to be affected if the queue manager at the other end of the channel restarts on a standby instance.
2. A WebSphere MQ MQI client application processes persistent message input from one queue and puts persistent message responses onto another queue as part of a single unit of work: if it handles an MQRC_BACKED_OUT reason code from MQPUT, MQGET or MQCMIT within sync point by issuing MQBACK and restarting the unit of work, then no messages are lost. Additionally the application does not need to do any special processing to deal with a connection failure.

Suppose however, in the second example, that the application is browsing the queue to select the message to process by using the MQGET option, MQGMO_MSG_UNDER_CURSOR. Reconnection resets the browse cursor, and the MQGET call does not return the correct message. In this example, the application has to be aware failover has occurred. Additionally, before issuing another MQGET for the message under the cursor, the application must restore the browse cursor.

Losing the browse cursor is one example of how the application context changes following reconnection. Other cases are documented in “Recovery of an automatically reconnected client” on page 384.

You have three alternative design patterns for WebSphere MQ MQI client applications following failover. Only one of them does not need to detect the failover.

No reconnection

In this pattern, the application stops all processing on the current connection when the connection is broken. For the application to continue processing, it must establish a new connection with the queue manager. The application is entirely responsible for transferring any state information it requires to continue processing on the new connection. Existing client applications that reconnect with a queue manager after losing their connection are written in this way.

The client receives a reason code, such as MQRC_CONNECTION_BROKEN , or MQRC_Q_MGR_NOT_AVAILABLE from the next MQI call after the connection is lost. The application must discard all its WebSphere MQ state information, such as queue handles, and issue a new MQCONN or MQCONNX call to establish a new connection, and then reopen the WebSphere MQ objects it needs to process.

The default MQI behavior is for the queue manager connection handle to become unusable after a connection with the queue manager is lost. The default is equivalent to setting the MQCNO_RECONNECT_DISABLED option on MQCONNX to prevent application reconnection after failover.

Failover tolerant

Write the application so it is unaffected by failover. Sometimes careful error handling is sufficient to deal with failover.

Reconnection aware

Register an MQCBT_EVENT_HANDLER event handler with the queue manager. The event handler is posted with MQRC_RECONNECTING when the client starts to try to reconnect to the server, and MQRC_RECONNECTED after a successful reconnection. You can then run a routine to reestablish a predictable state so that the client application is able to continue processing.

Recovery of an automatically reconnected client

Failover is an unexpected event, and for an automatically reconnected client to work as designed the consequences of reconnection must be predictable.

A major element of turning an unexpected failure into a predictable and reliable recovery is the use of transactions.

In the previous section, an example, “2” on page 383, was given of an WebSphere MQ MQI client using a local transaction to coordinate MQGET and MQPUT. The client issues an MQCMIT or MQBACK call in response to a MQRC_BACKED_OUT error and then resubmits the backed out transaction. The queue manager failure causes the transaction to be backed out, and the behavior of the client application ensures no transactions, and no messages, are lost.

Not all program state is managed as part of a transaction, and therefore the consequences of reconnection become harder to understand. You need to know how reconnection changes the state of a WebSphere MQ MQI client in order to design your client application to survive queue manager failover.

You might decide to design your application without any special failover code, handling reconnection errors with the same logic as other errors. Alternatively, you might choose to recognize that reconnection requires special error processing, and register an event handler with WebSphere MQ to run a routine to handle failover. The routine might handle the reconnection processing itself, or set a flag to indicate to the main program thread that when it resumes processing it needs to perform recovery processing.

The WebSphere MQ MQI client environment is aware of failover itself, and restores as much context as it can, following reconnection, by storing some state information in the client, and issuing additional MQI calls on behalf of the client application to restore its WebSphere MQ state. For example, handles to objects that were open at the point of failure are restored, and temporary dynamic queues are opened with the same name. But there are changes that are unavoidable and you need your design to deal with these changes. The changes can be categorized into five kinds:

1. New, or previously undiagnosed errors, are returned from MQI calls until a consistent new context state is restored by the application program.

   An example of receiving a new error is the return code MQRC_CONTEXT_NOT_AVAILABLE when trying to pass context after saving context before the reconnection. The context cannot be restored after reconnection because the security context is not passed to an unauthorized client program. To do so would let a malicious application program obtain the security context.

   Typically, applications handle common and predictable errors in a carefully designed way, and relegate uncommon errors to a generic error handler. The error handler might disconnect from WebSphere MQ and reconnect again, or even stop the program altogether. To improve continuity might need to deal with some errors in a different way.
2. Non-persistent messages might be lost.
3. Transactions are rolled back.
4. MQGET or MQPUT calls used outside a sync point might be interrupted with the possible loss of a message.
5. Timing induced errors, due to a prolonged wait in an MQI call.

Some details about lost context are listed in the following section.

- Non-persistent messages are discarded, unless put to a queue with the NPMCLASS(HIGH) option, and the queue manager failure did not interrupt the option of storing non-persistent messages on shutdown.
- A non-durable subscription is lost when a connection is broken. On reconnection, it is re-established. Consider using a durable subscription.
- The get-wait interval is recomputed; if its limit is exceeded it returns MQRC_NO_MSG_AVAILABLE. Similarly, subscription expiry is recomputed to give the same overall expiry time.
- The position of the browse cursor in a queue is lost; it is typically reestablished before the first message.
  - MQGET calls that specify MQGMO_BROWSE_MSG_UNDER_CURSOR or MQGMO_MSG_UNDER_CURSOR, fail with reason code MQRC_NO_MSG_AVAILABLE.

  - Messages locked for browsing are unlocked.
  - Browse marked messages with handle scope are unmarked and can be browsed again.
  - Cooperatively browse marked messages are unmarked in most cases.
- Security context is lost. Attempts to use saved message context, such as putting a message with MQPMO_PASS_ALL_CONTEXT fail with MQRC_CONTEXT_NOT_AVAILABLE.
- Message tokens are lost. MQGET using a message token returns the reason code MQRC_NO_MSG_AVAILABLE .

  Note: *MsgId* and *CorrelId*, as they are part of the message, are preserved with the message during failover, and so MQGET using MsgId or CorrelId work as expected.
- Messages put on a queue under sync point in an uncommitted transaction are no longer available.
- Processing messages in a logical order, or in a message group, results in a return code of MQRC_RECONNECT_INCOMPATIBLE after reconnection.
- An MQI call might return MQRC_RECONNECT_FAILED rather than the more general MQRC_CONNECTION_BROKEN that clients typically receive today.
- Reconnection during an MQPUT call outside sync point returns MQRC_CALL_INTERRUPTED if the WebSphere MQ MQI client does not know if the message was delivered to the queue manager successfully. Reconnection during MQCMIT behaves similarly.
- MQRC_CALL_INTERRUPTED is returned - after a successful reconnect - if the WebSphere MQ MQI client has received no response from the queue manager to indicate the success or failure of
  - the delivery of a persistent message using an MQPUT call outside sync point.
  - the delivery of a persistent message or a message with default persistence using an MQPUT1 call outside sync point.
  - the commit of a transaction using an MQCMIT call. The response is only ever returned after a successful reconnect.
- Channels are restarted as new instances (they might also be different channels), and so no channel exit state is retained.
- Temporary dynamic queues are restored as part of the process of recovering reconnectable clients that had temporary dynamic queues open. No messages on a temporary dynamic queue are restored, but applications that had the queue open, or had remembered the name of the queue, are able to continue processing.

  There is the possibility that if the queue is being used by an application other than the one that created it, that it might not be restored quickly enough to be present when it is next referenced. For example, if a client creates a temporary dynamic queue as a reply-to queue, and a reply message is to be placed on the queue by a channel, the queue might not be recovered in time. In this case, the channel would typically place the reply-to message on the dead letter queue.

  If a reconnectable client application opens a temporary dynamic queue by name (because another application has already created it), then when reconnection occurs, the WebSphere MQ MQI client is unable to recreate the temporary dynamic queue because it does not have the model to create it from. In the MQI, only one application can open the temporary dynamic queue by model. Other applications that wish to use the temporary dynamic queue must use MQPUT1, or server bindings, or be able to try the reconnection again if it fails.

  Only non-persistent messages might be put to a temporary dynamic queue, and these messages are lost during failover; this loss is true for messages being put to a temporary dynamic queue using MQPUT1 during reconnection. If failover occurs during the MQPUT1, the message might not be put, although the MQPUT1 succeeds. One workaround to this problem is to use permanent dynamic queues. Any server bindings application can open the temporary dynamic queue by name because it is not reconnectable.

Data recovery and high availability

High availability solutions using multi-instance queue managers must include a mechanism to recover data after a storage failure.

A multi-instance queue manager increases the availability of queue manager processes, but not the availability of other components, such as the file system, that the queue manager uses to store messages, and other information.

One way to make data highly available is to use networked resilient data storage. You can either build your own solution using a networked file system and resilient data storage, or you can buy an integrated solution. If you want to combine resilience with disaster recovery, then asynchronous disk replication, which permits disk replication over tens, or hundreds of kilometers, is available.

You can configure the way different WebSphere MQ directories are mapped to storage media, to make the best use of the media. For *multi-instance* queue managers there is an important distinction between two types of WebSphere MQ directories and files.

Directories that must be shared between the instances of a queue manager.
The information that must be shared between different instances of a queue manager is in two directories: the `qmgrs` and `logs` directories. The directories must be on a shared networked file system. You are advised to use a storage media that provides continuous high availability and excellent performance because the data is constantly changing as messages are created and deleted.

Directories and files that do not *have* to be shared between instances of a queue manager.
Some other directories do not have to be shared between different instances of a queue manager, and are quickly restored by means other than using a mirrored file system.

- WebSphere MQ executable files and the tools directory. Replace by reinstalling or by backing up and restoring from a backed up file archive.
- Configuration information that is modified for the installation as a whole. The configuration information is either managed by WebSphere MQ, such as the `mqs.ini` file on Windows, UNIX and Linux systems, or part of your own configuration management such as **MQSC** configuration scripts. Back up and restore using a file archive.
- Installation-wide output such as traces, error logs and FFDC files. The files are stored in the `errors` and `trace` subdirectories in the default data directory. The default data directory on UNIX and Linux systems is `/var/mqm` . On Windows the default data directory is the `WebSphere MQ` installation directory.

You can also use a backup queue manager to take regular media backups of a multi-instance queue manager using linear logging. A backup queue manager does not provide recovery that is as fast as from a mirrored file system, and it does not recover changes since the last backup. The backup queue manager mechanism is more appropriate for use in off-site disaster recovery scenarios than recovering a queue manager after a localized storage failure.

Combining IBM WebSphere MQ Availability solutions

Applications are using other IBM WebSphere MQ capabilities to improve availability. Multi-instance queue managers complement other high availability capabilities.

IBM WebSphere MQ Clusters increase queue availability

You can increase queue availability by creating multiple definitions of a cluster queue; up to one of every queue on each manager in the cluster.

Suppose a member of the cluster fails and then a new message is sent to a cluster queue. Unless the message *has* to go to the queue manager that has failed, the message is sent to another running queue manager in the cluster that has a definition of the queue.

Although clusters greatly increase availability, there are two related failure scenarios that result in messages getting delayed. Building a cluster with multi-instance queue managers reduces the chance of a message being delayed.

Marooned messages
If a queue manager in the cluster fails, no more messages that can be routed to other queue managers in the cluster are routed to the failed queue manager. Messages that have already been sent are marooned until the failed queue manager is restarted.

Affinities

Affinity is the term used to describe information shared between two otherwise separate computations. For example, an affinity exists between an application sending a request message to a server and the same application expecting to process the reply. Another example would be a sequence of messages, the processing of each message depending on the previous messages.

If you send messages to clustered queues you need to consider affinities. Do you need to send successive messages to the same queue manager, or can each message go to any member of the cluster?

If you do need to send messages to the same queue manager in the cluster and it fails, the messages wait in the transmission queue of the sender until the failed cluster queue manager is running again.

If the cluster is configured with multi-instance queue managers the delay waiting for the failed queue manager to restart is limited to the order of a minute or so while the standby takes over. When the standby is running, marooned messages resume processing, channels to the newly activated queue manager instance are started, and the messages that were waiting in transmission queues start flowing.

A possible way to configure a cluster to overcome messages being delayed by a failed queue manager, is to deploy two different queue managers to each server in the cluster, and arrange for one to be the active and one to be the standby instance of the different queue managers. This is an active-standby configuration, and it increases the availability of the cluster.

As well as having the benefits of reduced administration and increased scalability, clusters continue to provide additional elements of availability to complement multi-instance queue managers. Clusters protect against other types of failure that affect both the active and standby instances of a queue manager.

Uninterrupted service
A cluster provides an uninterrupted service. New messages received by the cluster are sent to active queue managers to be processed. Do not rely on a multi-instance queue manager to provide an uninterrupted service because it takes time for the standby queue manager to detect the failure and complete its startup, for its channels to be reconnected, and for failed batches of messages to be resubmitted.

Localized outage
There are practical limitations to how far apart the active, standby, and file system servers can be, as they need to interact at millisecond speeds to deliver acceptable performance.

Clustered queue managers require interaction speeds of the order of many seconds, and can be geographically dispersed anywhere in the world.

Operational error
By using two different mechanisms to increase availability you reduce the chances that an operational error, such as a human error, compromises your availability efforts.

Queue sharing groups increase message processing availability

Queue sharing groups, provided only on z/OS, allow a group of queue managers to share servicing a queue. If one queue manager fails, the other queue managers continue to process all the messages on the queue. Multi-instance queue managers are not supported on z/OS and complement queue sharing groups only as part of a wider messaging architecture.

WebSphere MQ Clients increase application availability

WebSphere MQ MQI client programs can connect to different queue managers in a queue manager group based on queue manager availability, connection weightings, and affinities. By running an application on a different machine from the one on which the queue manager is running, you can to improve the overall

availability of a solution as long as there is a way to reconnect the application if the queue manager instance it is connected to fails.

Queue manager groups are used to increase client availability by uncoupling a client from a queue manager that is stopped, and load balancing client connections across a group of queue managers, rather like an IP sprayer. The client application must have no affinities with the failed queue manager, such as a dependency on a particular queue, or it cannot resume processing.

Automatic client reconnection and multi-instance queue managers increase client availability by resolving some affinity problems. Automatic client reconnect is not supported by WebSphere MQ classes for Java.

You can set the `MQCNO` option `MQCNO_RECONNECT_Q_MGR`, to force a client to reconnect to the same queue manager:

1. If the previously connected single instance queue manager is not running the connection is tried again until the queue manager is running again.
2. If the queue manager is configured as a multi-instance queue manager, then the client reconnects to whichever instance is active.

By automatically reconnecting to the same queue manager, much of the state information the queue manager was holding on behalf of the client, such as the queues it had open and the topic it was subscribed to, are restored. If the client had opened a dynamic reply-to queue to receive a reply to a request, the connection to the reply-to queue is restored too.

Making sure that messages are not lost (logging)

WebSphere MQ logs all the information you need to recover from a queue manager failure.

WebSphere MQ records all significant changes to the data controlled by the queue manager in a recovery log.

This includes creating and deleting objects, persistent message updates, transaction states, changes to object attributes, and channel activities. The log contains the information you need to recover all updates to message queues by:

- Keeping records of queue manager changes
- Keeping records of queue updates for use by the restart process
- Enabling you to restore data after a hardware or software failure

However, WebSphere MQ also relies on the disk system hosting its files. If the disk system is itself unreliable, information, including log information, can still be lost.

What logs look like

Logs consist of primary and secondary files, and a control file. You define the number and size of log files and where they are stored in the file system.

A WebSphere MQ log consists of two components:

1. One or more files of log data.
2. A log control file

A file of log data is also known as a log extent.

There are a number of log files that contain the data being recorded. You can define the number and size (as explained in “Changing IBM WebSphere MQ and queue manager configuration information” on page 407), or take the system default of three files.

In WebSphere MQ for Windows, each of the three files defaults to 1 MB. In WebSphere MQ for UNIX and Linux systems, each of the three files defaults to 4 MB.

When you create a queue manager, the number of log files you define is the number of *primary* log files allocated. If you do not specify a number, the default value is used.

In WebSphere MQ for Windows, if you have not changed the log path, log files are created under the directory:

```
C:\Program Files\IBM\WebSphere MQ\log\<QMgrName>
```

In WebSphere MQ for UNIX and Linux systems, if you have not changed the log path, log files are created under the directory:

```
/var/mqm/log/<QMgrName>
```

WebSphere MQ starts with these primary log files, but if the primary log space is not sufficient, it allocates *secondary* log files. It does this dynamically and removes them when the demand for log space reduces. By default, up to two secondary log files can be allocated. You can change this default allocation, as described in “Changing IBM WebSphere MQ and queue manager configuration information” on page 407.

The log control file

The log control file contains the information needed to control the use of log files, such as their size and location and the name of the next available file.

The log control file is for internal queue manager use only.

The queue manager keeps control data associated with the state of the recovery log in the log control file and you must not modify the contents of the log control file.

Note: Ensure that the logs created when you start a queue manager are large enough to accommodate the size and volume of messages that your applications will handle. You will probably need to change the default log numbers and sizes to meet your requirements. For more information, see “Calculating the size of the log” on page 392.

Types of logging

In WebSphere MQ, the number of files that are required for logging depends on the file size, the number of messages you have received, and the length of the messages. There are two ways of maintaining records of queue manager activities: circular logging and linear logging.

Circular logging

Use circular logging if all you want is restart recovery, using the log to roll back transactions that were in progress when the system stopped.

Circular logging keeps all restart data in a ring of log files. Logging fills the first file in the ring, then moves on to the next, and so on, until all the files are full. It then goes back to the first file in the ring and starts again. This continues as long as the product is in use, and has the advantage that you never run out of log files.

WebSphere MQ keeps the log entries required to restart the queue manager without loss of data until they are no longer required to ensure queue manager data recovery. The mechanism for releasing log files for reuse is described in “Using checkpointing to ensure complete recovery” on page 390.

Linear logging

Use linear logging if you want both restart recovery and media recovery (re-creating lost or damaged data by replaying the contents of the log). Linear logging keeps the log data in a continuous sequence of files. Space is not reused, so you can always retrieve any record logged in any log extent that has not been deleted.

As disk space is finite, you might have to think about some form of archiving. It is an administrative task to manage your disk space for the log, reusing or extending the existing space as necessary.

The number of log files used with linear logging can be very large, depending on your message flow and the age of your queue manager. However, there are a number of files that are said to be *active*. Active files

contain the log entries required to restart the queue manager. Collectively, active log files are known as the *active log*. The number of active log files is usually less than the number of primary log files as defined in the configuration files. (See "Calculating the size of the log" on page 392 for information about defining the number.)

The key event that controls whether a log file is termed active or not is a *checkpoint*. A WebSphere MQ checkpoint is a point of consistency between the recovery log and object files. A checkpoint determines the set of log files needed to perform restart recovery. Log files that are not active are not required for restart recovery, and are termed inactive. In some cases inactive log files are required for media recovery. (See "Using checkpointing to ensure complete recovery" on page 390 for further information about checkpointing.)

Inactive log files can be archived because they are not required for restart recovery. Inactive log files that are not required for media recovery can be considered as superfluous log files. You can delete superfluous log files if they are no longer of interest to your operation. Refer to "Managing logs" on page 393 for further information about the disposition of log files.

If a new checkpoint is recorded in the second, or later, primary log file, the first file can become inactive and a new primary file is formatted and added to the end of the primary pool, restoring the number of primary files available for logging. In this way the primary log file pool can be seen to be a current set of files in an ever-extending list of log files. Again, it is an administrative task to manage the inactive files according to the requirements of your operation.

Although secondary log files are defined for linear logging, they are not used in normal operation. If a situation arises when, probably due to long-lived transactions, it is not possible to free a file from the active pool because it might still be required for a restart, secondary files are formatted and added to the active log file pool.

If the number of secondary files available is used up, requests for most further operations requiring log activity will be refused with an MQRC_RESOURCE_PROBLEM return code being returned to the application.

Both types of logging can cope with unexpected loss of power, assuming that there is no hardware failure.

Using checkpointing to ensure complete recovery

Checkpoints synchronize queue manager data and log files, and mark a point of consistency from which log records can be discarded. Frequent checkpointing makes recovery quicker.

Persistent updates to message queues happen in two stages. First, the records representing the update are written to the log, then the queue file is updated. The log files can thus become more up to date than the queue files. To ensure that restart processing begins from a consistent point, WebSphere MQ uses checkpoints. A checkpoint is a point in time when the record described in the log is the same as the record in the queue. The checkpoint itself consists of the series of log records needed to restart the queue manager; for example, the state of all transactions (units of work) active at the time of the checkpoint.

WebSphere MQ generates checkpoints automatically. They are taken when the queue manager starts, at shutdown, when logging space is running low, and after every 10 000 operations logged.

As the queues handle further messages, the checkpoint record becomes inconsistent with the current state of the queues.

When WebSphere MQ restarts, it finds the latest checkpoint record in the log. This information is held in the checkpoint file that is updated at the end of every checkpoint. The checkpoint record represents the most recent point of consistency between the log and the data. All the operations that have taken place since the checkpoint are replayed forward. This is known as the replay phase. The replay phase brings the queues back to the logical state they were in before the system failure or shutdown. During the replay phase a list is created of the transactions that were in-flight when the system failure or shutdown occurred. Messages AMQ7229 and AMQ7230 are issued to indicate the progression of the replay phase.

In order to know which operations to back out or commit, WebSphere MQ accesses each active log record associated with an in-flight transaction. This is known as the recovery phase. Messages AMQ7231, AMQ7232 and AMQ7234 are issued to indicate the progression of the recovery phase.

Once all the necessary log records have been accessed during the recovery phase, each active transaction is in turn resolved and each operation associated with the transaction will be either backed out or committed. This is known as the resolution phase. Message AMQ7233 is issued to indicate the progression of the resolution phase.

WebSphere MQ maintains internal pointers to the head and tail of the log. It moves the head pointer to the most recent checkpoint consistent with recovering message data.

Checkpoints are used to make recovery more efficient, and to control the reuse of primary and secondary log files.

In Figure 69 on page 391, all records before the latest checkpoint, Checkpoint 2, are no longer needed by WebSphere MQ. The queues can be recovered from the checkpoint information and any later log entries. For circular logging, any freed files before the checkpoint can be reused. For a linear log, the freed log files no longer need to be accessed for normal operation and become inactive. In the example, the queue head pointer is moved to point at the latest checkpoint, Checkpoint 2, which then becomes the new queue head, Head 2. Log File 1 can now be reused.

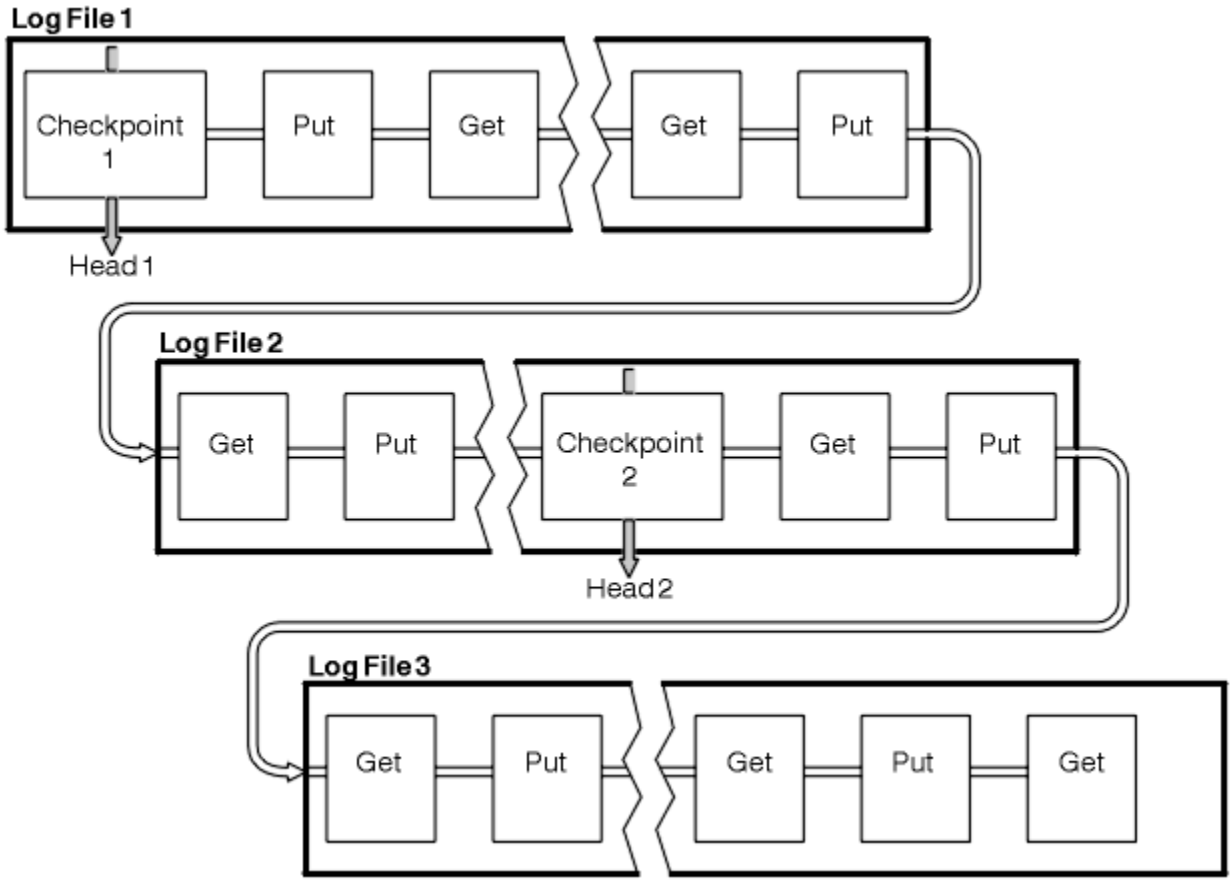


Figure 69. Checkpointing

Checkpointing with long-running transactions

How a long-running transaction affects reuse of log files.

Figure 70 on page 392 shows how a long-running transaction affects reuse of log files. In the example, a long-running transaction has made an entry to the log, shown as LR 1, after the first checkpoint shown. The transaction does not complete (at point LR 2) until after the third checkpoint. All the log information from LR 1 onwards is retained to allow recovery of that transaction, if necessary, until it has completed.

After the long-running transaction has completed, at LR 2, the head of the log is moved to Checkpoint 3, the latest logged checkpoint. The files containing log records before Checkpoint 3, Head 2, are no longer needed. If you are using circular logging, the space can be reused.

If the primary log files are completely full before the long-running transaction completes, secondary log files are used to avoid the logs getting full.

When the log head is moved and you are using circular logging, the primary log files might become eligible for reuse and the logger, after filling the current file, reuses the first primary file available to it. If you are using linear logging, the log head is still moved down the active pool and the first file becomes inactive. A new primary file is formatted and added to the bottom of the pool in readiness for future logging activities.

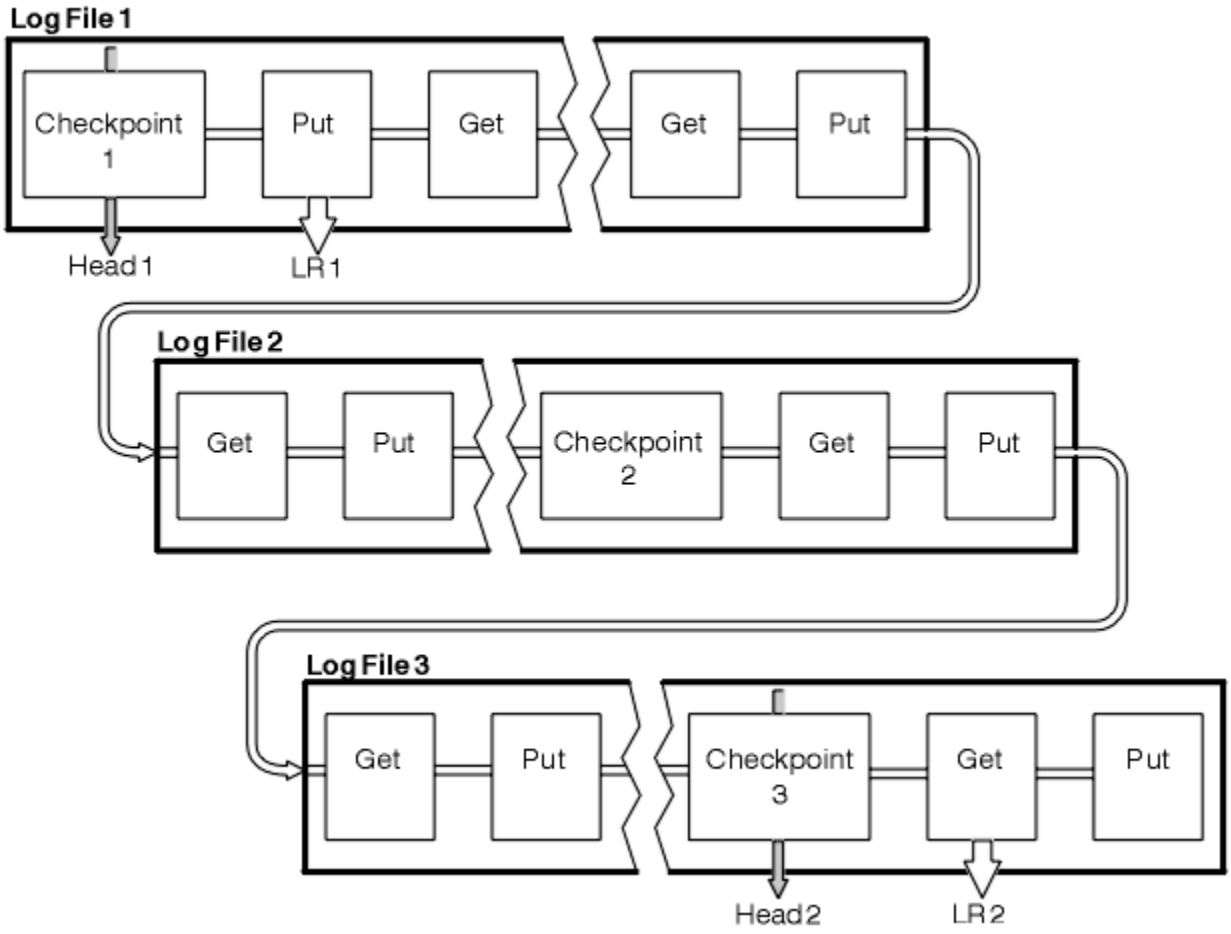


Figure 70. Checkpointing with a long-running transaction

Calculating the size of the log

Estimating the size of log a queue manager needs.

After deciding whether the queue manager uses circular or linear logging, you need to estimate the size of the log that the queue manager needs. The size of the log is determined by the following log configuration parameters:

LogFilePages
The size of each primary and secondary log file in units of 4K pages

LogPrimaryFiles
The number of preallocated primary log files

LogSecondaryFiles
The number of secondary log files that can be created for use when the primary log files are full

Table 31 on page 393 shows the amount of data the queue manager logs for various operations. Most queue manager operations need a minimal amount of log space. However, when a persistent message is put to a queue, **all** the message data must be written to the log to make it possible to recover the message. The size of the log depends, typically, on the number and size of the persistent messages the queue manager needs to handle.

Table 31. Log entry sizes (all values are approximate)	
Operation	**Size**
Put persistent message	750 bytes + message length If the message is large, it is divided into segments of 261844 bytes, each segment adding an extra 300 bytes.
Get message	260 bytes
Sync point, commit	750 bytes
Sync point, rollback	1000 bytes + 12 bytes for each get or put to be rolled back
Create object	1500 bytes
Delete object	300 bytes
Alter attributes	1024 bytes
Record media image	800 bytes + image The image is divided into segments of 260 000 bytes, each segment adding an extra 300 bytes.
Checkpoint	750 bytes + 200 bytes for each active unit of work + 380 bytes for each cluster-sender channel, if you are using multiple cluster transmission queues per queue manager. Additional data might be logged for any uncommitted puts or gets that are buffered for performance reasons. If you have cluster-sender channels, then at each checkpoint an extra 380 bytes is written to the log per cluster-sender channel.

Note:

1. You can change the number of primary and secondary log files each time the queue manager starts.
2. You cannot change the log file size; you must determine it **before** creating the queue manager.
3. The number of primary log files and the log file size determine the amount of log space that is preallocated when the queue manager is created.
4. The total number of primary and secondary log files cannot exceed 511 on UNIX and Linux systems, or 255 on Windows, which in the presence of long-running transactions, limits the maximum amount of log space available to the queue manager for restart recovery. The amount of log space the queue manager might need for media recovery does not share this limit.
5. When *circular* logging is being used, the queue manager reuses primary log space. This means that the queue manager's log can be smaller than the amount of data you estimated that the queue manager needs to log. The queue manager will, up to a limit, allocate a secondary log file when a log file becomes full, and the next primary log file in the sequence is not available.
6. Primary log files are made available for reuse during a checkpoint. The queue manager takes both the primary and secondary log space in to consideration before taking a checkpoint because the amount of log space is running low.

   If you do not define more primary log files than secondary log files, the queue manager might allocate secondary log files before a checkpoint is taken. This makes the primary log files available for reuse.

Managing logs

Logs are nearly self-managing, but sometimes need managing to resolve space problems.

Over time, some of the log records written become unnecessary for restarting the queue manager. If you are using circular logging, the queue manager reclaims freed space in the log files. This activity is not

apparent to the user and you do not usually see the amount of disk space used reduce because the space allocated is quickly reused.

Of the log records, only those written since the start of the last complete checkpoint, and those written by any active transactions, are needed to restart the queue manager. Thus, the log might fill if a checkpoint has not been taken for a long time, or if a long-running transaction wrote a log record a long time ago. The queue manager tries to take checkpoints often enough to avoid the first problem.

When a long-running transaction fills the log, attempts to write log records fail and some MQI calls return MQRC_RESOURCE_PROBLEM. (Space is reserved to commit or roll back all in-flight transactions, so **MQCMIT** or **MQBACK** should not fail.)

The queue manager rolls back transactions that consume too much log space. An application that has a transaction is rolled back in this way cannot perform subsequent **MQPUT** or **MQGET** operations specifying sync point under the same transaction. An attempt to put or get a message under sync point in this state returns MQRC_BACKED_OUT. The application can then issue **MQCMIT**, which returns MQRC_BACKED_OUT, or **MQBACK** and start a new transaction. When the transaction consuming too much log space has been rolled back, its log space is released and the queue manager continues to operate normally.

If the log fills, message AMQ7463 is issued. In addition, if the log fills because a long-running transaction has prevented the space being released, message AMQ7465 is issued.

Finally, if records are being written to the log faster than the log can process them, message AMQ7466 is issued. If you see this message, increase the number of log files or reduce the amount of data being processed by the queue manager.

What happens when a disk gets full

The queue manager logging component can cope with a full disk, and with full log files. If the disk containing the log fills, the queue manager issues message AMQ6708 and an error record is taken.

The log files are created at their maximum size, rather than being extended as log records are written to them. This means that WebSphere MQ can run out of disk space only when it is creating a new file; it cannot run out of space when it is writing a record to the log. WebSphere MQ always knows how much space is available in the existing log files, and manages the space within the files accordingly.

If you fill the drive containing the log files, you might be able to free some disk space. If you are using a linear log, there might be some inactive log files in the log directory, and you can copy these files to another drive or device. If you still run out of space, check that the configuration of the log in the queue manager configuration file is correct. You might be able to reduce the number of primary or secondary log files so that the log does not outgrow the available space. You cannot alter the size of the log files for an existing queue manager. The queue manager assumes that all log files are the same size.

Managing log files

Allocate sufficient space for your log files. For linear logging, you can delete old log files when they are no longer required.

If you are using circular logging, ensure that there is sufficient space to hold the log files when you configure your system (see “Log defaults for IBM WebSphere MQ” on page 415 and “Queue manager logs” on page 423). The amount of disk space used by the log does not increase beyond the configured size, including space for secondary files to be created when required.

If you are using a linear log, the log files are added continually as data is logged, and the amount of disk space used increases with time. If the rate of data being logged is high, disk space is used rapidly by new log files.

Over time, the older log files for a linear log are no longer required to restart the queue manager or to perform media recovery of any damaged objects. The following are methods for determining which log files are still required:

Logger event messages

When enabled, logger event messages are generated when queue managers starts writing log records to a new log file. The contents of logger event messages specify the log files that are still required for

queue manager restart, and media recovery. For more information about logger event messages, see Logger events

Queue manager status
Running the MQSC command, DISPLAY QMSTATUS, or the PCF command, Inquire Queue Manager Status, returns queue manager information, including details of the required log files. For more information about MQSC commands, see Script (MQSC) Commands , and for information about PCF commands, see Automating administration tasks.

Queue manager messages
Periodically, the queue manager issues a pair of messages to indicate which of the log files are needed:

- Message AMQ7467 gives the name of the oldest log file required to restart the queue manager. This log file and all newer log files must be available during queue manager restart.
- Message AMQ7468 gives the name of the oldest log file needed for media recovery.

Only log files required for queue manager restart, active log files, are required to be online. Inactive log files can be copied to an archive medium such as tape for disaster recovery, and removed from the log directory. Inactive log files that are not required for media recovery can be considered as superfluous log files. You can delete superfluous log files if they are no longer of interest to your operation.

To determine "older" and "newer" log files, use the log file number rather than the modification times applied by the file system.

If any log file that is needed cannot be found, operator message AMQ6767 is issued. Make the log file, and all subsequent log files, available to the queue manager and try the operation again.

Note: When performing media recovery, all the required log files must be available in the log file directory at the same time. Make sure that you take regular media images of any objects you might want to recover to avoid running out of disk space to hold all the required log files. To take a media image of all your objects in your queue manager, run the **rcdmqimg** command as shown in the following examples:

On Windows

```
rcdmqimg -m QMNAME -t all *
```

On UNIX and Linux

```
rcdmqimg -m QMNAME -t all "*"
```

Running **rcdmqimg** moves the media log sequence number (LSN) forwards. For further details on log sequence numbers, see “Dumping the contents of the log using the dmpmqlog command” on page 398. **rcdmqimg** does not run automatically, therefore must be run manually or from an automatic task you have created. For more information about this command, see rcdmqimg and dmpmqlog.

Note: Messages AMQ7467 and AMQ7468 can also be issued at the time of running the `rcdmqimg` command.

Determining superfluous log files
When managing linear log files, it is important to be sure which files can be deleted or archived. This information will assist you in making this decision.

Do not use the file system's modification times to determine "older" log files. Use only the log file number. The queue manager's use of log files follows complex rules, including pre-allocation and formatting of log files before they are needed. You might see log files with modification times that would be misleading if you try to use these times to determine relative age.

To determine the oldest log file needed to restart the queue manager, issue the command `DISPLAY QMSTATUS RECLOG`.

To determine the oldest log file needed to perform media recovery, issue the command `DISPLAY QMSTATUS MEDIALOG`.

In general a lower log file number implies an older log. Unless you have a very high log file turnover, of the order of 3000 log files per day for 10 years, then you do not need to cater for the number wrapping at 9 999 999. In this case, you can archive any log file with a number less than the RECLOG value, and you can delete any log file with a number less than both the RECLOG and MEDIALOG values.

If however you have a very high turnover of log files, or otherwise want to be confident of coping with the general case, then the following algorithm can typically be used:

```
Let S == restart log file number
         (from DISPLAY QMSTATUS RECLOG).
Let M == media recovery log file number
          (from DISPLAY QMSTATUS MEDIALOG).
Let L == a log file number with eligibility for deletion or archiving
          that needs to be determined.

function minlog (a, b) {
  if (abs (a - b) < 5000000)
    return min (a, b);  # Not wrapped.
  else
    return max (a, b);  # Wrapped. }

A log file L can be deleted if
       (L != S && L != M && minlog (L, minlog (S, M)) == L).
A log file L can be archived if
       (L != S && minlog (L, S) == L).
```

Log file location

When choosing a location for your log files, remember that operation is severely affected if WebSphere MQ fails to format a new log because of lack of disk space.

If you are using a circular log, ensure that there is sufficient space on the drive for at least the configured primary log files. Also leave space for at least one secondary log file, which is needed if the log has to grow.

If you are using a linear log, allow considerably more space; the space consumed by the log increases continuously as data is logged.

Ideally, place the log files on a separate disk drive from the queue manager data. This has benefits in terms of performance. It might also be possible to place the log files on multiple disk drives in a mirrored arrangement. This protects against failure of the drive containing the log. Without mirroring, you could be forced to go back to the last backup of your WebSphere MQ system.

Using the log for recovery

Using logs to recover from failures.

There are several ways that your data can be damaged. WebSphere MQ helps you to recover from:

- A damaged data object
- A power loss in the system
- A communications failure

This section looks at how the logs are used to recover from these problems.

Recovering from power loss or communications failures

WebSphere MQ can recover from both communications failures and loss of power. In addition, it can sometimes recover from other types of problem, such as inadvertent deletion of a file.

In the case of a communications failure, messages remain on queues until they are removed by a receiving application. If the message is being transmitted, it remains on the transmission queue until it

can be successfully transmitted. To recover from a communications failure, you can usually restart the channels using the link that failed.

If you lose power, when the queue manager is restarted WebSphere MQ restores the queues to their committed state at the time of the failure. This ensures that no persistent messages are lost. Nonpersistent messages are discarded; they do not survive when WebSphere MQ stops abruptly.

Recovering damaged objects

There are ways in which an IBM WebSphere MQ object can become unusable, for example because of inadvertent damage. You must then recover either your complete system or some part of it. The action required depends on when the damage is detected, whether the log method selected supports media recovery, and which objects are damaged.

Media recovery

Media recovery re-creates objects from information recorded in a linear log. For example, if an object file is inadvertently deleted, or becomes unusable for some other reason, media recovery can re-create it. The information in the log required for media recovery of an object is called a *media image*.

A media image is a sequence of log records containing an image of an object from which the object itself can be re-created.

The first log record required to re-create an object is known as its *media recovery record*; it is the start of the latest media image for the object. The media recovery record of each object is one of the pieces of information recorded during a checkpoint.

When an object is re-created from its media image, it is also necessary to replay any log records describing updates performed on the object since the last image was taken.

Consider, for example, a local queue that has an image of the queue object taken before a persistent message is put onto the queue. In order to re-create the latest image of the object, it is necessary to replay the log entries recording the putting of the message to the queue, in addition to replaying the image itself.

When an object is created, the log records written contain enough information to completely re-create the object. These records make up the first media image of the object. Then, at each shutdown, the queue manager records media images automatically as follows:

- Images of all process objects and queues that are not local
- Images of empty local queues

Media images can also be recorded manually using the **rcdmqimg** command, described in rcdmqimg. This command writes a media image of the IBM WebSphere MQ object. When a media image has been written, only the logs that hold the media image, and all the logs created after this time, are required to re-create damaged objects. The benefit of creating media images depends on such factors as the amount of free storage available, and the speed at which log files are created.

Recovering from media images

A queue manager automatically recovers some objects from their media image during startup of the queue manager. It recovers a queue automatically if it was involved in any transaction that was incomplete when the queue manager last shut down, and is found to be corrupted or damaged during the restart processing.

You must recover other objects manually, using the **rcrmqobj** command, which replays the records in the log to re-create the IBM WebSphere MQ object. The object is re-created from its latest image found in the log, together with all applicable log events between the time the image was saved and the time the re-create command was issued. If an IBM WebSphere MQ object becomes damaged, the only valid actions that can be performed are either to delete it or to re-create it by this method. Nonpersistent messages cannot be recovered in this way.

See rcrmqobj for further details of the **rcrmqobj** command.

The log file containing the media recovery record, and all subsequent log files, must be available in the log file directory when attempting media recovery of an object. If a required file cannot be found, operator message AMQ6767 is issued and the media recovery operation fails. If you do not take regular media images of the objects that you want to re-create, you might have insufficient disk space to hold all the log files required to re-create an object.

Recovering damaged objects during startup

If the queue manager discovers a damaged object during startup, the action it takes depends on the type of object and whether the queue manager is configured to support media recovery.

If the queue manager object is damaged, the queue manager cannot start unless it can recover the object. If the queue manager is configured with a linear log, and thus supports media recovery, IBM WebSphere MQ automatically tries to re-create the queue manager object from its media images. If the log method selected does not support media recovery, you can either restore a backup of the queue manager or delete the queue manager.

If any transactions were active when the queue manager stopped, the local queues containing the persistent, uncommitted messages put or got inside these transactions are also required to start the queue manager successfully. If any of these local queues is found to be damaged, and the queue manager supports media recovery, it automatically tries to re-create them from their media images. If any of the queues cannot be recovered, IBM WebSphere MQ cannot start.

If any damaged local queues containing uncommitted messages are discovered during startup processing on a queue manager that does not support media recovery, the queues are marked as damaged objects and the uncommitted messages on them are ignored. This situation is because it is not possible to perform media recovery of damaged objects on such a queue manager and the only action left is to delete them. Message AMQ7472 is issued to report any damage.

Recovering damaged objects at other times

Media recovery of objects is automatic only during startup. At other times, when object damage is detected, operator message AMQ7472 is issued and most operations using the object fail. If the queue manager object is damaged at any time after the queue manager has started, the queue manager performs a pre-emptive shutdown. When an object has been damaged you can delete it or, if the queue manager is using a linear log, attempt to recover it from its media image using the `rcrmqobj` command (see rcrmqobj for further details).

Protecting IBM WebSphere MQ log files

Do not touch the log files when a queue manager is running, recovery might be impossible. Use super user or mqm authority to protect log files against inadvertent modification.

Do not remove the active log files manually when an IBM WebSphere MQ queue manager is running. If a user inadvertently deletes the log files that a queue manager needs to restart, IBM WebSphere MQ **does not** issue any errors and continues to process data *including persistent messages*. The queue manager shuts down normally, but can fail to restart. Recovery of messages then becomes impossible.

Users with the authority to remove logs that are being used by an active queue manager also have authority to delete other important queue manager resources (such as queue files, the object catalog, and IBM WebSphere MQ executable files). They can therefore damage, perhaps through inexperience, a running or dormant queue manager in a way against which IBM WebSphere MQ cannot protect itself.

Exercise caution when conferring super user or mqm authority.

Dumping the contents of the log using the dmpmqlog command

How to use the `dmpmqlog` command to dump the contents of the queue manager log.

Use the `dmpmqlog` command to dump the contents of the queue manager log. By default all active log records are dumped, that is, the command starts dumping from the head of the log (usually the start of the last completed checkpoint).

The log can usually be dumped only when the queue manager is not running. Because the queue manager takes a checkpoint during shutdown, the active portion of the log usually contains a small number of log records. However, you can use the `dmpmqlog` command to dump more log records using one of the following options to change the start position of the dump:

- Start dumping from the *base* of the log. The base of the log is the first log record in the log file that contains the head of the log. The amount of additional data dumped in this case depends on where the head of the log is positioned in the log file. If it is near the start of the log file, only a small amount of additional data is dumped. If the head is near the end of the log file, significantly more data is dumped.
- Specify the start position of the dump as an individual log record. Each log record is identified by a unique *log sequence number (LSN)*. In the case of circular logging, this starting log record cannot be before the base of the log; this restriction does not apply to linear logs. You might need to reinstate inactive log files before running the command. You must specify a valid LSN, taken from previous `dmpmqlog` output, as the start position.

  For example, with linear logging you can specify the `nextlsn` from your last `dmpmqlog` output. The `nextlsn` appears in `Log File Header` and indicates the LSN of the next log record to be written. Use this as a start position to format all log records written since the last time the log was dumped.
- **For linear logs only**, you can instruct `dmpmqlog` to start formatting log records from any given log file extent. In this case, `dmpmqlog` expects to find this log file, and each successive one, in the same directory as the active log files. This option does not apply to circular logs, where `dmpmqlog` cannot access log records prior to the base of the log.

The output from the `dmpmqlog` command is the `Log File Header` and a series of formatted log records. The queue manager uses several log records to record changes to its data.

Some of the information that is formatted is only of use internally. The following list includes the most useful log records:

Log File Header
Each log has a single log file header, which is always the first thing formatted by the `dmpmqlog` command. It contains the following fields:

logactive	The number of primary log extents.
loginactive	The number of secondary log extents.
logsize	The number of 4 KB pages per extent.
baselsn	The first LSN in the log extent containing the head of the log.
nextlsn	The LSN of the next log record to be written.
headlsn	The LSN of the log record at the head of the log.
tailsn	The LSN identifying the tail position of the log.
hflag1	Whether the log is CIRCULAR or LOG RETAIN (linear).
HeadExtentID	The log extent containing the head of the log.

Log Record Header
Each log record within the log has a fixed header containing the following information:

LSN	The log sequence number.
LogRecdType	The type of the log record.
XTranid	The transaction identifier associated with this log record (if any). A *TranType* of MQI indicates a WebSphere MQ-only transaction. A *TranType* of XA is involved with other resource managers. Updates involved within the same unit of work have the same *XTranid*.
QueueName	The queue associated with this log record (if any).

Qid The unique internal identifier for the queue.

PrevLSN The LSN of the previous log record within the same transaction (if any).

Start Queue Manager
This logs that the queue manager has started.

StartDate The date that the queue manager started.

StartTime The time that the queue manager started.

Stop Queue Manager
This logs that the queue manager has stopped.

StopDate The date that the queue manager stopped.

StopTime The time that the queue manager stopped.

ForceFlag The type of shutdown used.

Start Checkpoint
This denotes the start of a queue manager checkpoint.

End Checkpoint
This denotes the end of a queue manager checkpoint.

ChkPtLSN The LSN of the log record that started this checkpoint.

Put Message
This logs a persistent message put to a queue. If the message was put under sync point, the log record header contains a non-null *XTranid*. The remainder of the record contains:

MapIndex An identifier for the message on the queue. It can be used to match the corresponding MQGET that was used to get this message from the queue. In this case a subsequent *Get Message* log record can be found containing the same QueueName and MapIndex. At this point the MapIndex identifier can be reused for a subsequent put message to that queue.

Data Contained in the hex dump for this log record is various internal data followed by the Message Descriptor (eyecatcher MD) and the message data itself.

Put Part
Persistent messages that are too large for a single log record are logged as multiple *Put Part* log records followed by a single *Put Message* record. If there are *Put Part* records, then the *PrevLSN* field will chain the *Put Part* records and the final *Put Message* record together.

Data Continues the message data where the previous log record left off.

Get Message
Only gets of persistent messages are logged. If the message was got under sync point, the log record header contains a non-null *XTranid*. The remainder of the record contains:

MapIndex Identifies the message that was retrieved from the queue. The most recent *Put Message* log record containing the same *QueueName* and *MapIndex* identifies the message that was retrieved.

QPriority The priority of the message retrieved from the queue.

Start Transaction
Indicates the start of a new transaction. A TranType of MQI indicates a WebSphere MQ-only transaction. A TranType of XA indicates one that involves other resource managers. All updates made by this transaction will have the same *XTranid*.

Prepare Transaction
Indicates that the queue manager is prepared to commit the updates associated with the specified *XTranid*. This log record is written as part of a two-phase commit involving other resource managers.

Commit Transaction
Indicates that the queue manager has committed all updates made by a transaction.

Rollback Transaction
This denotes the queue manager's intention to roll back a transaction.

End Transaction
This denotes the end of a rolled-back transaction.

Transaction Table
This record is written during sync point. It records the state of each transaction that has made persistent updates. For each transaction the following information is recorded:

XTranid	The transaction identifier.
FirstLSN	The LSN of the first log record associated with the transaction.
LastLSN	The LSN of the last log record associated with the transaction.

Transaction Participants
This log record is written by the XA Transaction Manager component of the queue manager. It records the external resource managers that are participating in transactions. For each participant the following is recorded:

RMName	The name of the resource manager.
RMID	The resource manager identifier. This is also logged in subsequent *Transaction Prepared* log records that record global transactions in which the resource manager is participating.
SwitchFile	The switch load file for this resource manager.
XAOpenString	The XA open string for this resource manager.
XACloseString	The XA close string for this resource manager.

Transaction Prepared
This log record is written by the XA Transaction Manager component of the queue manager. It indicates that the specified global transaction has been successfully prepared. Each of the participating resource managers will be instructed to commit. The *RMID* of each prepared resource manager is recorded in the log record. If the queue manager itself is participating in the transaction a *Participant Entry* with an *RMID* of zero will be present.

Transaction Forget
This log record is written by the XA Transaction Manager component of the queue manager. It follows the *Transaction Prepared* log record when the commit decision has been delivered to each participant.

Purge Queue
This logs the fact that all messages on a queue have been purged, for example, using the MQSC command CLEAR QUEUE.

Queue Attributes
This logs the initialization or change of the attributes of a queue.

Create Object
This logs the creation of a WebSphere MQ object.

ObjName The name of the object that was created.

UserId The user ID performing the creation.

Delete Object
This logs the deletion of a WebSphere MQ object.

ObjName The name of the object that was deleted.

Backing up and restoring IBM WebSphere MQ queue manager data

Backing up queue managers and queue manager data.

Periodically, you can take measures to protect queue managers against possible corruption caused by hardware failures. There are three ways of protecting a queue manager:

Back up the queue manager data
If the hardware fails, a queue manager might be forced to stop. If any queue manager log data is lost due to the hardware failure, the queue manager might be unable to restart. If you back up queue manager data you might be able to recover some, or all, of the lost queue manager data.

In general, the more frequently you back up queue manager data, the less data you lose in the event of hardware failure that results in loss of integrity of the recovery log.

To back up queue manager data, the queue manager must not be running.

To back up and restore queue manager data see:

- “Backing up queue manager data” on page 403.
- “Restoring queue manager data” on page 403.

Use a backup queue manager
If the hardware failure is severe, a queue manager might be unrecoverable. In this situation, if the unrecoverable queue manager has a dedicated backup queue manager, the backup queue manager can be activated in place of the unrecoverable queue manager. If it was updated regularly, the backup queue manager log can contain log data that includes the last complete log from the unrecoverable queue manager.

A backup queue manager can be updated while the existing queue manager is still running.

To create and activate a backup queue manager see:

- “Creating a backup queue manager” on page 404.
- “Starting a backup queue manager” on page 405.

Back up the queue manager configuration only
If the hardware fails, a queue manager might be forced to stop. If both the queue manager configuration and log data is lost due to the hardware failure, the queue manager will be unable to restart or to be recovered from the log. If you back up the queue manager configuration you would be able to recreate the queue manager and all of its objects from saved definitions.

To back up queue manager configuration, the queue manager must be running.

To back up and restore the queue manager configuration see:

- “Backing up queue manager configuration” on page 406
- “Restoring queue manager configuration” on page 406

Backing up queue manager data

Backing up queue manager data can help you to guard against possible loss of data caused by hardware errors.

Before you begin

Ensure that the queue manager is not running. If you try to take a backup of a running queue manager, the backup might not be consistent because of updates in progress when the files were copied. If possible, stop your queue manager by running the `endmqm -w` command (a wait shutdown), only if that fails, use the `endmqm -i` command (an immediate shutdown).

About this task

To take a backup copy of a queue manager's data, complete the following tasks:

1. Search for the directories under which the queue manager places its data and its log files, by using the information in the configuration files. For more information, see “Changing IBM WebSphere MQ and queue manager configuration information” on page 407.

   Note: You might have some difficulty in understanding the names that appear in the directory. The names are transformed to ensure that they are compatible with the platform on which you are using WebSphere MQ. For more information about name transformations, see Understanding WebSphere MQ file names.
2. Take copies of all the queue manager's data and log file directories, including all subdirectories.

   Make sure that you do not miss any files, especially the log control file, as described in “What logs look like” on page 388, and the configuration files as described in “Initialization and configuration files” on page 69. Some of the directories might be empty, but you need them all to restore the backup at a later date.
3. Preserve the ownerships of the files. For WebSphere MQ for UNIX and Linux systems, you can do this with the `tar` command. (If you have queues larger than 2 GB, you cannot use the `tar` command. For more information, see Enabling large queues.

   Note: When you upgrade to WebSphere MQ Version 7.5 and later, ensure to take a backup of the **`.ini`** file and the registry entries. The queue manager information is stored in the **`.ini`** file and can be used to revert to a previous version of WebSphere MQ.

Restoring queue manager data

Follow these steps to restore a backup of a queue manager's data.

Before you begin

Ensure that the queue manager is not running.

About this task

To restore a backup of a queue manager's data:

1. Find the directories under which the queue manager places its data and its log files, by using the information in the configuration files.
2. Empty the directories into which you are going to place the backed-up data.
3. Copy the backed-up queue manager data and log files into the correct places.
4. Update the configuration information files.

Check the resulting directory structure to ensure that you have all the required directories.

For more information about IBM WebSphere MQ directories and subdirectories, see Directory structure on Windows systems and Directory content on UNIX and Linux systems .

Make sure that you have a log control file as well as the log files. Also check that the IBM WebSphere MQ and queue manager configuration files are consistent so that WebSphere MQ can look for the restored data in the correct places.

For circular logging, back up the queue manager data and log file directories at the same time so that you can restore a consistent set of queue manager data and logs.

For linear logging, back up the queue manager data and log file directories at the same time. It is possible to restore only the queue manager data files if a corresponding complete sequence of log files is available.

Note: When you upgrade to WebSphere MQ Version 7.5 and later, ensure to take a backup of the **.ini** file and the registry entries. The queue manager information is stored in the **.ini** file and can be used to revert to a previous version of WebSphere MQ.

Results

If the data was backed up and restored correctly, the queue manager will now start.

Using a backup queue manager

An existing queue manager can have a dedicated backup queue manager.

A backup queue manager is an inactive copy of the existing queue manager. If the existing queue manager becomes unrecoverable due to severe hardware failure, the backup queue manager can be brought online to replace the unrecoverable queue manager.

The existing queue manager log files must regularly be copied to the backup queue manager to ensure that the backup queue manager remains an effective method for disaster recovery. The existing queue manager does not need to be stopped for log files to be copied, however you should only copy a log file if the queue manager has finished writing to it. Because the existing queue manager log is continually updated, there is always a slight discrepancy between the existing queue manager log and the log data copied to the backup queue manager log. Regular updates to the backup queue manager minimizes the discrepancy between the two logs.

If a backup queue manager is required to be brought online it must be activated, and then started. The requirement to activate a backup queue manager before it is started is a preventive measure to protect against a backup queue manager being started accidentally. Once a backup queue manager is activated it can no longer be updated.

For information on how to create, update, and start a backup queue manager, see the following topics:

- “Creating a backup queue manager” on page 404
- “Updating a backup queue manager” on page 405
- “Starting a backup queue manager” on page 405

Creating a backup queue manager

You can only use a backup queue manager when using linear logging.

To create a backup queue manager for an existing queue manager, do the following:

1. Create a backup queue manager for the existing queue manager using the control command `crtmqm`. The backup queue manager requires the following:
   - To have the same attributes as the existing queue manager, for example the queue manager name, the logging type, and the log file size.
   - To be on the same platform as the existing queue manager.
   - To be at an equal, or higher, code level than the existing queue manager.
2. Take copies of all the existing queue manager's data and log file directories, including all subdirectories, as described in “Backing up queue manager data” on page 403.
3. Overwrite the backup queue manager's data and log file directories, including all subdirectories, with the copies taken from the existing queue manager.

4. Execute the following control command on the backup queue manager:

```
strmqm -r BackupQMName
```

   This flags the queue manager as a backup queue manager within WebSphere MQ, and replays all the copied log extents to bring the backup queue manager in step with the existing queue manager.

Updating a backup queue manager

To ensure that a backup queue manager remains an effective method for disaster recovery it must be updated regularly.

Regular updating lessens the discrepancy between the backup queue manager log, and the current queue manager log. There is no need to stop the queue manager to be backed up.

To update a backup queue manager, do the following:

1. Issue the following Script (MQSC) command on the queue manager to be backed up:

```
RESET QMGR TYPE(ADVANCELOG)
```

   This stops any writing to the current log, and then advances the queue manager logging to the next log extent. This ensures you back up all information logged up to the current time.
2. Obtain the (new) current active log extent number by issuing the following Script (MQSC) command on the queue manager to be backed up:

```
DIS QMSTATUS CURRLOG
```

3. Copy the updated log extent files from the current queue manager log directory to the backup queue manager log directory - copy all the log extents since the last update, and up to (but not including) the current extent noted in step 2. Copy only log extent files, the ones beginning with "S...".
4. Issue the following control command on the backup queue manager:

```
strmqm -r BackupQMName
```

   This replays all the copied log extents and brings the backup queue manager into step with the queue manager. When the replay finishes you receive a message that identifies all the log extents required for restart recovery, and all the log extents required for media recovery.

   Warning: If you copy a `non-contiguous` set of logs to the backup queue manager log directory, only the logs up to the point where the first missing log is found will be replayed.

Starting a backup queue manager

You can substitute a backup queue manager for an unrecoverable queue manager.

To do this, perform the following steps:

1. Execute the following control command to activate the backup queue manager:

```
strmqm -a BackupQMName
```

   The backup queue manager is activated. Now active, the backup queue manager can no longer be updated.
2. Execute the following control command to start the backup queue manager:

```
strmqm BackupQMName
```

WebSphere MQ regards this as restart recovery, and utilizes the log from the backup queue manager. During the last update to the backup queue manager replay will have occurred, therefore only the active transactions from the last recorded checkpoint are rolled back.

When an unrecoverable queue manager is substituted for a backup queue manager some of the queue manager data from the unrecoverable queue manager can be lost. The amount of lost data is dependent on how recently the backup queue manager was last updated. The more recently the last update, the less queue manager data loss.

3. Restart all channels.

Check the resulting directory structure to ensure that you have all the required directories.

See Planning file system support for more information about WebSphere MQ directories and subdirectories.

Make sure that you have a log control file as well as the log files. Also check that the WebSphere MQ and queue manager configuration files are consistent so that WebSphere MQ can look in the correct places for the restored data.

If the data was backed up and restored correctly, the queue manager will now start.

Note: Even though the queue manager data and log files are held in different directories, back up and restore the directories at the same time. If the queue manager data and log files have different ages, the queue manager is not in a valid state and will probably not start. If it does start, your data is likely to be corrupt.

Backing up queue manager configuration

Backing up queue manager configuration can help you to rebuild a queue manager from its definitions.

To take a backup copy of a queue manager's configuration:

1. Ensure that the queue manager is running.
2. a. On AIX, HP-UX, Linux, Solaris, or Windows: Execute the Dump MQ Configuration command (dmpmqcfg) using the default formatting option of (-f mqsc) MQSC and all attributes (-a), use standard output redirection to store the definitions into a file, for example:

```
dmpmqcfg -m MYQMGR -a > /mq/backups/MYQMGR.mqsc
```

Restoring queue manager configuration

Follow these steps to restore a backup of a queue manager's configuration.

To restore a backup of a queue manager's configuration:

1. Ensure that the queue manager is running. Note that the queue manager may have been recreated if damage to the data and logs is unrecoverable by other means.
2. Depending on your platform, execute one of the following commands:
   a. On AIX, HP-UX, Linux, Solaris, or Windows: Execute runmqsc against the queue manager, use standard input redirection to restore the definitions from a script file generated by the Dump MQ Configuration (dmpmqcfg) command, for example:

```
runmqsc MYQMGR < /mq/backups/MYQMGR.mqsc
```

Related reference
dmpmqcfg

Changing IBM WebSphere MQ and queue manager configuration information

Change the behavior of IBM WebSphere MQ or an individual queue manager to suit the needs of your installation.

You can change IBM WebSphere MQ configuration information by changing the values specified on a set of configuration attributes (or parameters) that govern IBM WebSphere MQ.

Change attribute information by editing the IBM WebSphere MQ configuration files. On IBM WebSphere MQ for Windows and Linux (x86 and x86-64 platforms), the IBM WebSphere MQ configuration files can be edited using the IBM WebSphere MQ Explorer.

On Windows systems you can also use amqmdain to change configuration information, as described in amqmdain

To find out more about configuring IBM WebSphere MQ and queue managers for your platform, see the following subtopics:

Related concepts

"Configuring" on page 5
Create one or more queue managers on one or more computers, and configure them on your development, test, and production systems to process messages that contain your business data.

Related tasks

Planning

Administering WebSphere MQ

Changing configuration information on UNIX, Linux, and Windows systems

Configuration attributes are held in configuration files, at the level of the node and of the queue manager.

On Windows, UNIX and Linux platforms, you can change IBM WebSphere MQ configuration attributes within:

- An IBM WebSphere MQ configuration file (**mqs.ini**) to effect changes for IBM WebSphere MQ on the node as a whole. There is one mqs.ini file for each node.
- A queue manager configuration file (**qm.ini**) to effect changes for specific queue managers. There is one qm.ini file for each queue manager on the node.

Client configuration options are held separately, in the client configuration file.

A configuration file (or ***stanza*** file) contains one or more stanzas, which are groups of lines in the .ini file that together have a common function or define part of a system, such as log functions, channel functions, and installable services.

Because the IBM WebSphere MQ configuration file is used to locate the data associated with queue managers, a nonexistent or incorrect configuration file can cause some or all MQSC commands to fail. Also, applications cannot connect to a queue manager that is not defined in the IBM WebSphere MQ configuration file.

Any changes you make to a configuration file usually do not take effect until the next time the queue manager is started.

On Windows and Linux (x86 and x86-64 platforms) systems, you can edit configuration information from the IBM WebSphere MQ Explorer.

On Windows systems you can also use the the `amqmdain` command to edit the configuration files.

For more information about the configuration options on Windows, UNIX and Linux systems, see the following subtopics:

Related concepts

"Configuring" on page 5

Create one or more queue managers on one or more computers, and configure them on your development, test, and production systems to process messages that contain your business data.

"Changing IBM WebSphere MQ and queue manager configuration information" on page 407
Change the behavior of IBM WebSphere MQ or an individual queue manager to suit the needs of your installation.

Related tasks

Planning

Administering WebSphere MQ

Related reference

"Attributes for changing IBM WebSphere MQ configuration information" on page 413
On IBM WebSphere MQ for Windows systems and on IBM WebSphere MQ for Linux (x86 and x86-64 platforms) systems, modify configuration information using the IBM WebSphere MQ Explorer. On other systems, modify the information by editing the mqs.ini configuration file.

"Changing queue manager configuration information" on page 419
The attributes described here modify the configuration of an individual queue manager. They override any settings for WebSphere MQ.

Editing configuration files

Edit configuration files using commands or a standard text editor.

Before editing a configuration file, back it up so that you have a copy you can revert to if the need arises.

You can edit configuration files either:

- Automatically, using commands that change the configuration of queue managers on the node
- Manually, using a standard text editor

You can edit the default values in the WebSphere MQ configuration files after installation.

If you set an incorrect value on a configuration file attribute, the value is ignored and an operator message is issued to indicate the problem. (The effect is the same as missing out the attribute entirely.)

When you create a new queue manager:

- Back up the WebSphere MQ configuration file
- Back up the new queue manager configuration file

Comments can be included in configuration files by adding a ";" or a "#" character before the comment text. If you want to use a ";" or a "#" character without it representing a comment, you can prefix the character with a "\" character and it will be used as part of the configuration data.

When do you need to edit a configuration file?

Edit a configuration file to recover from backup, move a queue manager, change the default queue manager or to assist IBM support.

You might need to edit a configuration file if, for example:

- You lose a configuration file. (Recover from backup if you can.)
- You need to move one or more queue managers to a new directory.
- You need to change your default queue manager; this could happen if you accidentally delete the existing queue manager.
- You are advised to do so by your IBM Support Center.

Configuration file priorities

The value of an attribute is defined in multiple places. Attributes set in commands take precedence over attributes in configuration files.

The attribute values of a configuration file are set according to the following priorities:

- Parameters entered on the command line take precedence over values defined in the configuration files
- Values defined in the qm.ini files take precedence over values defined in the mqs.ini file

The IBM WebSphere MQ configuration file, mqs.ini

The IBM WebSphere MQ configuration file, mqs.ini, contains information relevant to all the queue managers on the node. It is created automatically during installation.

On IBM WebSphere MQ for UNIX and Linux products, the data directory and log directory are always `/var/mqm` and `/var/mqm/log` respectively.

On Windows systems, the location of the data directory `mqs.ini`, and the location of the log directory, are stored in the registry, as their location can vary.

In addition, on Windows systems, the installation configuration information (contained in `mqinst.ini` on IBM WebSphere MQ for UNIX and Linux systems) is in the registry, as there is no `mqinst.ini` file on Windows.

The mqs.ini file for Windows systems is given by the WorkPath specified in the `HKLM\SOFTWARE\IBM\WepSphere MQ` key. It contains:

- The names of the queue managers
- The name of the default queue manager
- The location of the files associated with each of them

The supplied `LogDefaults` stanza for a new IBM WebSphere MQ installation does not contain any explicit values for the attributes. The lack of an attribute means that the default for this value is used upon creation of a new queue manager. The default values are shown for the `LogDefaults` stanza in Figure 71 on page 410. A value of zero for the `LogBufferPages` attribute means 512.

If you require a non-default value, you must explicitly specify that value in the `LogDefaults` stanza.

```
#********************************************************************#
#* Module Name: mqs.ini                                             *#
#* Type       : WebSphere MQ Machine-wide Configuration File        *#
#* Function   : Define WebSphere MQ resources for an entire machine *#
#********************************************************************#
#* Notes      :                                                     *#
#* 1) This is the installation time default configuration           *#
#*                                                                  *#
#********************************************************************#
AllQueueManagers:
#********************************************************************#
#* The path to the qmgrs directory, below which queue manager data  *#
#* is stored                                                        *#
#********************************************************************#
DefaultPrefix=/var/mqm

LogDefaults:
   LogPrimaryFiles=3
   LogSecondaryFiles=2
   LogFilePages=4096
   LogType=CIRCULAR
   LogBufferPages=0
   LogDefaultPath=/var/mqm/log

QueueManager:
   Name=saturn.queue.manager
   Prefix=/var/mqm
   Directory=saturn!queue!manager
   InstallationName=Installation1

QueueManager:
   Name=pluto.queue.manager
   Prefix=/var/mqm
   Directory=pluto!queue!manager
   InstallationName=Installation2

DefaultQueueManager:
   Name=saturn.queue.manager

ApiExitTemplate:
   Name=OurPayrollQueueAuditor
   Sequence=2
   Function=EntryPoint
   Module=/usr/ABC/auditor
   Data=123

ApiExitCommon:
   Name=MQPoliceman
   Sequence=1
   Function=EntryPoint
   Module=/usr/MQPolice/tmqp
   Data=CheckEverything
```

Figure 71. Example of an IBM WebSphere MQ configuration file for UNIX systems

Queue manager configuration files, qm.ini

A queue manager configuration file, qm.ini, contains information relevant to a specific queue manager.

There is one queue manager configuration file for each queue manager. The qm.ini file is automatically created when the queue manager with which it is associated is created.

V 7.5.0.9 From IBM WebSphere MQ Version 7.5.0, Fix Pack 9, the **strmqm** command checks the syntax of the CHANNELS and SSL stanzas in the `qm.ini` file before starting the queue manager fully, which makes it much easier to see what is wrong, and correct it quickly if **strmqm** finds that the `qm.ini` file contains any errors. For more information, see strmqm.

Location of the `qm.ini` files

Windows UNIX Linux

On UNIX and Linux systems a qm.ini file is held in the root of the directory tree occupied by the queue manager. For example, the path and the name for a configuration file for a queue manager called QMNAME is:

```
/var/mqm/qmgrs/QMNAME/qm.ini
```

On Windows systems the location of the qm.ini file is given by the WorkPath specified in the HKLM\SOFTWARE\IBM\WebSphere MQ key. For example, the path and the name for a configuration file for a queue manager called QMNAME is:

```
C:\Program Files\IBM\WebSphere MQ\qmgrs\QMNAME\qm.ini
```

The queue manager name can be up to 48 characters in length. However, this does not guarantee that the name is valid or unique. Therefore, a directory name is generated based on the queue manager name. This process is known as *name transformation*. For a description, see Understanding WebSphere MQ file names.

Example `qm.ini` file

The following example shows how groups of attributes might be arranged in a queue manager configuration file in IBM WebSphere MQ for UNIX and Linux systems.

```
#* Module Name: qm.ini                                               *#
#* Type       : WebSphere MQ queue manager configuration file      *#
#  Function   : Define the configuration of a single queue manager *#
#*                                                                  *#
#********************************************************************#
#* Notes      :                                                     *#
#* 1) This file defines the configuration of the queue manager     *#
#*                                                                  *#
#********************************************************************#

ExitPath:
   ExitsDefaultPath=/var/mqm/exits
   ExitsDefaultPath64=/var/mqm/exits64

Service:
   Name=AuthorizationService
   EntryPoints=13

ServiceComponent:
   Service=AuthorizationService
   Name=MQSeries.UNIX.auth.service
   Module=opt/mqm/bin/amqzfu
   ComponentDataSize=0

Log:
   LogPrimaryFiles=3
   LogSecondaryFiles=2
   LogFilePages=4096
   LogType=CIRCULAR
   LogBufferPages=0[1]
   LogPath=/var/mqm/log/saturn!queue!manager/

AccessMode:
   SecurityGroup=wmq\wmq

XAResourceManager:
   Name=DB2 Resource Manager Bank
   SwitchFile=/usr/bin/db2swit
   XAOpenString=MQBankDB
   XACloseString=
   ThreadOfControl=THREAD

Channels: [2]
   MaxChannels=200
   MaxActiveChannels=100
   MQIBindType=STANDARD
```

```
AccessMode:
   SecurityGroup=wmq\wmq
TCP:
   KeepAlive = Yes
   SvrSndBuffSize=32768
   SvrRcvBuffSize=32768
   Connect_Timeout=0

QMErrorLog:
   ErrorLogSize=262144
   ExcludeMessage=7234
   SuppressMessage=9001,9002,9202
   SuppressInterval=30

ApiExitLocal:
   Name=ClientApplicationAPIchecker
   Sequence=3
   Function=EntryPoint
   Module=/usr/Dev/ClientAppChecker
   Data=9.20.176.20
```

Notes:

1. The value of zero for `LogBufferPages` gives a value of 512.
2. For more information on the Channel stanza, see "Initialization and configuration files" on page 69.
3. The maximum number of XAResourceManager stanzas is limited to 255. However, you should use only a small number of stanzas to avoid transaction performance degradation.

WebSphere MQ in Unix uses configuration files that have a file extension of .ini, for example, qm.ini. There are some utilities in WebSphereMQ, such as **setmqm**, that will make a temporary backup copy of the files. For example, the file qm.ini will create a backup copy named qm.ini.bak. A utility will modify the qm.ini file, store the updated file, then delete the qm.ini.bak file. If the utility cannot store the qm.ini file, then it restores the contents of qm.ini from the backup file qm.ini.bak, and then deletes the qm.ini.bak file.

If there is an existing qm.ini.bak file, the utility reverts the qm.ini file with the contents of the qm.ini.bak and deletes the qm.ini.bak file. Therefore, you should not create backup copies of the *.ini files by using a file extension of .bak, because these backup files might be deleted by WebSphere MQ utilities.

See "Changing configuration information on UNIX, Linux, and Windows systems" on page 407 for information on when your changes take effect.

Installation configuration file, mqinst.ini

UNIX and Linux systems

The installation configuration file, `mqinst.ini`, contains information about all the IBM WebSphere MQ installations on a UNIX or Linux system.

The `mqinst.ini` file is in the `/etc/opt/mqm` directory on UNIX and Linux systems. It contains information about which installation, if any, is the primary installation as well as the following information for each installation:

- The installation name
- The installation description
- The installation identifier
- The installation path

This file must not be edited or referenced directly since its format is not fixed, and could change. Instead, use the following commands to create, delete, query, and modify, the values in the mqinst.ini file:

crtmqinst to create entries.
dltmqinst to delete entries.
dspmqinst to display entries.
setmqinst to set entries.

The installation identifier, for internal use only, is set automatically and must not be changed.

Windows systems

Installation configuration information is held in the following key on Windows systems:

```
HKLM\SOFTWARE\IBM\WebSphere MQ\Installation\<InstallationName>
```

This key must not be edited or referenced directly since its format is not fixed, and could change. Instead, use the following commands to query, and modify, the values in the registry:

dspmqinst to display entries.
setmqinst to set entries.

On Windows, the **crtmqinst** and **dltmqinst** commands are not available. The installation and uninstallation processes handle the creation and deletion of the required registry entries.

Attributes for changing IBM WebSphere MQ configuration information

On IBM WebSphere MQ for Windows systems and on IBM WebSphere MQ for Linux (x86 and x86-64 platforms) systems, modify configuration information using the IBM WebSphere MQ Explorer. On other systems, modify the information by editing the mqs.ini configuration file.

See the following subtopics for attributes for specific components:

Related concepts

"Configuring" on page 5
Create one or more queue managers on one or more computers, and configure them on your development, test, and production systems to process messages that contain your business data.

"Changing IBM WebSphere MQ and queue manager configuration information" on page 407
Change the behavior of IBM WebSphere MQ or an individual queue manager to suit the needs of your installation.

Related tasks

Planning
Administering WebSphere MQ

Related reference

"Changing queue manager configuration information" on page 419
The attributes described here modify the configuration of an individual queue manager. They override any settings for WebSphere MQ.

All queue managers

Use the `General` and `Extended` WebSphere MQ properties page from the IBM WebSphere MQ Explorer, or the `AllQueueManagers` stanza in the mqs.ini file to specify the following information about all queue managers.

DefaultPrefix=*directory_name*

This attribute specifies the path to the qmgrs directory, within which the queue manager data is kept.

If you change the default prefix for the queue manager, replicate the directory structure that was created at installation time.

In particular, you must create the qmgrs structure. Stop WebSphere MQ before changing the default prefix, and restart WebSphere MQ only after you have moved the structures to the new location and changed the default prefix.

Note: Do not delete the `/var/mqm/errors` directory on UNIX and Linux systems, or the `\errors` directory on Windows systems.

As an alternative to changing the default prefix, you can use the environment variable MQSPREFIX to override the `DefaultPrefix` for the `crtmqm` command.

Because of operating system restrictions, keep the supplied path sufficiently short so that the sum of the path length and any queue manager name is a maximum of 70 characters long.

ConvEBCDICNewline=NL_TO_LF|TABLE|ISO
EBCDIC code pages contain a newline (NL) character that is not supported by ASCII code pages (although some ISO variants of ASCII contain an equivalent).

Use the ConvEBCDICNewline attribute to specify how WebSphere MQ is to convert the EBCDIC NL character into ASCII format.

NL_TO_LF
Convert the EBCDIC NL character (X'15') to the ASCII line feed character, LF (X'0A'), for all EBCDIC to ASCII conversions.

NL_TO_LF is the default.

TABLE
Convert the EBCDIC NL character according to the conversion tables used on your platform for all EBCDIC to ASCII conversions.

The effect of this type of conversion might vary from platform to platform and from language to language; even on the same platform, the behavior might vary if you use different CCSIDs.

ISO
Convert:

- ISO CCSIDs using the TABLE method
- All other CCSIDs using the NL_TO_CF method

Possible ISO CCSIDs are shown in Table 32 on page 414.

Table 32. List of possible ISO CCSIDs

CCSID	Code Set
819	ISO8859-1
912	ISO8859-2
915	ISO8859-5
1089	ISO8859-6
813	ISO8859-7
916	ISO8859-8
920	ISO8859-9
1051	roman8

If the ASCII CCSID is not an ISO subset, ConvEBCDICNewline defaults to NL_TO_LF.

Default queue manager

Use the `General` WebSphere MQ properties page from the IBM WebSphere MQ Explorer, or the `DefaultQueueManager` stanza in the mqs.ini file to specify the default queue manager.

Name=*default_queue_manager*
The default queue manager processes any commands for which a queue manager name is not explicitly specified. The `DefaultQueueManager` attribute is automatically updated if you create a new default queue manager. If you inadvertently create a new default queue manager and then want to revert to the original, alter the `DefaultQueueManager` attribute manually.

Exit properties

Use the `Extended` IBM WebSphere MQ properties page from the IBM WebSphere MQ Explorer, or the `ExitProperties` stanza in the mqs.ini file to specify configuration options used by queue manager exit programs.

CLWLMode=SAFE|FAST
The cluster workload (CLWL) exit allows you to specify which cluster queue in the cluster to open in response to an MQI call (for example, `MQOPEN`, `MQPUT`). The CLWL exit runs either in FAST mode or SAFE mode depending on the value you specify on the CLWLMode attribute. If you omit the CLWLMode attribute, the cluster workload exit runs in SAFE mode.

SAFE
Run the CLWL exit in a separate process from the queue manager. This is the default.

If a problem arises with the user-written CLWL exit when running in SAFE mode, the following happens:

- The CLWL server process (amqzlwa0) fails.
- The queue manager restarts the CLWL server process.
- The error is reported to you in the error log. If an MQI call is in progress, you receive notification in the form of a return code.

The integrity of the queue manager is preserved.

Note: Running the CLWL exit in a separate process can affect performance.

FAST
Run the cluster exit inline in the queue manager process.

Specifying this option improves performance by avoiding the process switching costs associated with running in SAFE mode, but does so at the expense of queue manager integrity. You should only run the CLWL exit in FAST mode if you are convinced that there are **no** problems with your CLWL exit, and you are particularly concerned about performance.

If a problem arises when the CLWL exit is running in FAST mode, the queue manager will fail and you run the risk of the integrity of the queue manager being compromised.

Log defaults for IBM WebSphere MQ

Use the `Default log settings` IBM WebSphere MQ properties page from the IBM WebSphere MQ Explorer, or the `LogDefaults` stanza in the mqs.ini file to specify information about log defaults for all queue managers.

If the stanza does not exist then the MQ defaults will be used. The log attributes are used as default values when you create a queue manager, but can be overridden if you specify the log attributes on the `crtmqm` command. See **crtmqm** for details of this command.

Once a queue manager has been created, the log attributes for that queue manager are taken from the settings described in “Queue manager logs” on page 423.

The default prefix (specified in “All queue managers” on page 413) and log path specified for the particular queue manager (specified in “Queue manager logs” on page 423) allow the queue manager and its log to be on different physical drives. This is the recommended method, although by default they are on the same drive.

For information about calculating log sizes, see “Calculating the size of the log” on page 392.

Note: The limits given in the following parameter list are limits set by WebSphere MQ. Operating system limits might reduce the maximum possible log size.

LogPrimaryFiles=3|*2-254* (Windows)|*2-510* (UNIX and Linux systems)
The log files allocated when the queue manager is created.

The minimum number of primary log files you can have is 2 and the maximum is 254 on Windows, or 510 on UNIX and Linux systems. The default is 3.

The total number of primary and secondary log files must not exceed 255 on Windows, or 511 on UNIX and Linux systems, and must not be less than 3.

The value is examined when the queue manager is created or started. You can change it after the queue manager has been created. However, a change in the value is not effective until the queue manager is restarted, and the effect might not be immediate.

LogSecondaryFiles=2|*1-253* (Windows)|*1-509* (UNIX and Linux systems)
The log files allocated when the primary files are exhausted.

The minimum number of secondary log files is 1 and the maximum is 253 on Windows, or 509 on UNIX and Linux systems. The default number is 2.

The total number of primary and secondary log files must not exceed 255 on Windows, or 511 on UNIX and Linux systems, and must not be less than 3.

The value is examined when the queue manager is started. You can change this value, but changes do not become effective until the queue manager is restarted, and even then the effect might not be immediate.

LogFilePages=*number*
The log data is held in a series of files called log files. The log file size is specified in units of 4 KB pages.

The default number of log file pages is 4096, giving a log file size of 16 MB.

On UNIX and Linux systems the minimum number of log file pages is 64, and on Windows the minimum number of log file pages is 32; in both cases the maximum number is 65 535.

Note: The size of the log files specified during queue manager creation cannot be changed for a queue manager.

LogType=CIRCULAR|LINEAR
The type of log to be used. The default is CIRCULAR.

CIRCULAR
Start restart recovery using the log to roll back transactions that were in progress when the system stopped.

See “Types of logging” on page 389 for a fuller explanation of circular logging.

LINEAR
For both restart recovery and media or forward recovery (creating lost or damaged data by replaying the contents of the log).

See “Types of logging” on page 389 for a fuller explanation of linear logging.

If you want to change the default, you can either edit the LogType attribute, or specify linear logging using the `crtmqm` command. You cannot change the logging method after a queue manager has been created.

LogBufferPages=0|*0-4096*
The amount of memory allocated to buffer records for writing, specifying the size of the buffers in units of 4 KB pages.

The minimum number of buffer pages is 18 and the maximum is 4096. Larger buffers lead to higher throughput, especially for larger messages.

If you specify 0 (the default), the queue manager selects the size. In WebSphere MQ Version 7.0 this is 512 (2048 KB).

If you specify a number in the range 1 through 17, the queue manager defaults to 18 (72 KB). If you specify a number in the range 18 and through 4096, the queue manager uses the number specified to set the memory allocated.

LogDefaultPath=*directory_name*
The directory in which the log files for a queue manager reside. The directory resides on a local device to which the queue manager can write and, preferably, on a different drive from the message queues. Specifying a different drive gives added protection in case of system failure.

The default is:

- `<DefaultPrefix>\log` for WebSphere MQ for Windows where `<DefaultPrefix>` is the value specified on the `DefaultPrefix` attribute on the `All Queue Managers` WebSphere MQ properties page. This value is set at install time.
- `/var/mqm/log` for WebSphere MQ for UNIX and Linux systems

Alternatively, you can specify the name of a directory on the `crtmqm` command using the -ld flag. When a queue manager is created, a directory is also created under the queue manager directory, and this is used to hold the log files. The name of this directory is based on the queue manager name. This ensures that the log file path is unique, and also that it conforms to any limitations on directory name lengths.

If you do not specify -ld on the `crtmqm` command, the value of the `LogDefaultPath` attribute in the mqs.ini file is used.

The queue manager name is appended to the directory name to ensure that multiple queue managers use different log directories.

When the queue manager is created, a `LogPath` value is created in the log attributes in the configuration information, giving the complete directory name for the queue manager's log. This value is used to locate the log when the queue manager is started or deleted.

LogWriteIntegrity=SingleWrite|DoubleWrite| `TripleWrite`
The method the logger uses to reliably write log records.

`TripleWrite`
This is the default method.

Note, that you can select **DoubleWrite**, but if you do so, the system interprets this as **TripleWrite**.

SingleWrite
You should use **SingleWrite**, only if the file-system or device hosting the WebSphere MQ recovery log explicitly guarantees the atomicity of 4KB writes.

That is, when a write of a 4KB page fails for any reason, the only two possible states are either the before image, or the after image. No intermediate state should be possible.

Advanced Configuration and Power Interface (ACPI)

Use the `ACPI` WebSphere MQ properties page from the IBM WebSphere MQ Explorer, to specify how WebSphere MQ is to behave when the system receives a suspend request.

Windows supports the Advanced Configuration and Power Interface (ACPI) standard. This enables Windows users with ACPI enabled hardware to stop and restart channels when the system enters and resumes from suspend mode.

Note that the settings specified in the `ACPI` WebSphere MQ properties page are applied only when the Alert Monitor is running. The Alert Monitor icon is present on the taskbar if the Alert Monitor is running.

DoDialog=Y | N
Displays the dialog at the time of a suspend request.

DenySuspend=Y | N
Denies the suspend request. This is used if DoDialog=N, or if DoDialog=Y and a dialog cannot be displayed, for example, because your notebook lid is closed.

CheckChannelsRunning=Y | N
Checks whether any channels are running. The outcome can determine the outcome of the other settings.

The following table outlines the effect of each combination of these parameters:

DoDialog	DenySuspend	CheckChannels Running	Action
N	N	N	Accept the suspend request.
N	N	Y	Accept the suspend request.
N	Y	N	Deny the suspend request.
N	Y	Y	If any channels are running deny the suspend request; if not accept the request.
Y	N	N	Display the dialog (see Note; accept the suspend request). This is the default.
Y	N	Y	If no channels are running accept the suspend request; if they are display the dialog (see Note; accept the request).
Y	Y	N	Display the dialog (Note; deny the suspend request).
Y	Y	Y	If no channels are running accept the suspend request; if they are display the dialog (Note; deny the request).

Note: In cases where the action is to display the dialog, if the dialog cannot be displayed (for example because your notebook lid is closed), the DenySuspend option is used to determine whether the suspend request is accepted or denied.

API exits

Use the IBM WebSphere MQ Explorer or the `amqmdain` command to change the entries for API exits.

Use the `Exits` IBM WebSphere MQ properties page from the IBM WebSphere MQ Explorer, or the `ApiExitTemplate` and `ApiExitCommon` stanza in the mqs.ini file to identify API exit routines for all queue managers. On Windows systems, you can also use the `amqmdain` command to change the entries for API exits. (To identify API exit routines for individual queue managers, you use the `ApiExitLocal` stanza, as described in “API exits” on page 432.)

For a complete description of the attributes for these stanzas, see Configuring API exits.

Queue managers

There is one `QueueManager` stanza for every queue manager. Use the stanza to specify the location of the queue manager directory.

On Windows, UNIX and Linux systems, there is one `QueueManager` stanza for every queue manager. These attributes specify the queue manager name, and the name of the directory containing the files associated with that queue manager. The name of the directory is based on the queue manager name, but is transformed if the queue manager name is not a valid file name. See, Understanding WebSphere MQ file names for more information about name transformation.

Name=*queue_manager_name*
The name of the queue manager.

Prefix=*prefix*
Where the queue manager files are stored. By default, this value is the same as the value specified on the DefaultPrefix attribute of the All Queue Managers information.

Directory=*name*
The name of the subdirectory under the <prefix>\QMGRS directory where the queue manager files are stored. This name is based on the queue manager name, but can be transformed if there is a duplicate name or if the queue manager name is not a valid file name.

DataPath=*path*
An explicit data path provided when the queue manager was created, this overrides Prefix and Directory as the path to the queue manager data.

InstallationName=*name*
The name of the WebSphere MQ installation associated with this queue manager. Commands from this installation must be used when interacting with this queue manager. If no InstallationName value is present, the queue manager is associated with an installation of WebSphere MQ earlier than version 7.1.

Related concepts

"Associating a queue manager with an installation" on page 16
When you create a queue manager, it is automatically associated with the installation that issued the **crtmqm** command. On UNIX, Linux, and Windows, you can change the installation associated with a queue manager using the **setmqm** command.

Security

Use the Security stanza in the qm.ini file to specify options for the Object Authority Manager (OAM).

ClusterQueueAccessControl=RQMName|Xmitq
Set this attribute to check the access control of cluster queues or fully qualified queues hosted on cluster queue managers.

RQMName
The profiles checked for access control of remotely hosted queues are named queues or named queue manager profiles.

Xmitq
The profiles checked for access control of remotely hosted queues are resolved to the SYSTEM.CLUSTER.TRANSMIT.QUEUE.

Xmitq is the default value.

GroupModel=GlobalGroups
This attribute determines whether the OAM checks global groups when determining the group membership of a user on Windows.

The default is not to check global groups.

GlobalGroups
The OAM checks global groups.

With GlobalGroups set, the authorization commands, **setmqaut**, **dspmqaut**, and **dmpmqaut** accept global groups names; see the **setmqaut** -gparameter.

Note: Setting the ClusterQueueAcessControl=RQMName and having a custom implementation of the Authorization Service at less than MQZAS_VERSION_6 results in the queue manager not starting. In this instance, either set ClusterQueueAcessControl=Xmitq or upgrade the custom Authorization Service to MQZAS_VERSION_6 or greater.

Changing queue manager configuration information

The attributes described here modify the configuration of an individual queue manager. They override any settings for WebSphere MQ.

On UNIX and Linux systems, you modify the queue manager configuration information by editing the qm.ini configuration file. When you are defining a stanza in qm.ini, you do not need to start each item on a new line. You can use either a semicolon (;) or a hash character (#) to indicate a comment.

On Windows and Linux (x86 and x86-64 platforms) systems, you can modify some configuration information by using the IBM WebSphere MQ Explorer. However, because there are significant implications to changing installable services and their components, the installable services are read-only in the IBM WebSphere MQ Explorer. You must therefore make any changes to installable services by using **regedit** on Windows, and by editing the qm.ini file on UNIX and Linux.

For more details on changing queue manager configuration information, see the following subtopics:

Related concepts

"Configuring" on page 5
Create one or more queue managers on one or more computers, and configure them on your development, test, and production systems to process messages that contain your business data.

"Changing IBM WebSphere MQ and queue manager configuration information" on page 407
Change the behavior of IBM WebSphere MQ or an individual queue manager to suit the needs of your installation.

Related tasks

Planning

Administering WebSphere MQ

Related reference

"Attributes for changing IBM WebSphere MQ configuration information" on page 413
On IBM WebSphere MQ for Windows systems and on IBM WebSphere MQ for Linux (x86 and x86-64 platforms) systems, modify configuration information using the IBM WebSphere MQ Explorer. On other systems, modify the information by editing the mqs.ini configuration file.

Access Mode

Access Mode applies to Windows servers only. The AccessMode stanza is set by the `-a [r]` option on the **crtmqm** command. Do not change the AccessMode stanza after the queue manager has been created.

Use the access group (`-a [r]`) option of the **crtmqm** command to specify a Windows security group, members of which will be granted full access to all queue manager data files. The group can either be a local or global group, depending upon the syntax used. Valid syntax for the group name is as follows:

LocalGroup
Domain name \ *GlobalGroup name*
GlobalGroup name@*Domain name*

You must define the additional access group before running the crtmqm command with the `-a [r]` option.

If you specify the group using `-ar` instead of `-a`, the local mqm group is not granted access to the queue manager data files. Use this option, if the file system hosting the queue manager data files does not support access control entries for locally defined groups.

The group is typically a global security group, which is used to provide multi-instance queue managers with access to a shared queue manager data and logs folder. Use the additional security access group to set read and write permissions on the folder or to share containing queue manager data and log files.

The additional security access group is an alternative to using the local group named `mqm` to set permissions on the folder containing queue manager data and logs. Unlike the local group `mqm`, you can make the additional security access group a local or a global group. It must be a global group to set permissions on the shared folders that contain the data and log files used by multi-instance queue managers.

The Windows operating system checks the access permissions to read and write queue manager data and log files. It checks the permissions of the user ID that is running queue manager processes. The user ID that is checked depends on whether you started the queue manager as a service or you started it interactively. If you started the queue manager as a service, the user ID checked by the Windows system is the user ID you configured with the **Prepare** IBM WebSphere MQ wizard. If you started the queue

manager interactively, the user ID checked by the Windows system is the user ID that ran the **strmqm** command.

The user ID must be a member of the local mqm group to start the queue manager. If the user ID is a member of the additional security access group, the queue manager can read and write files that are given permissions by using the group.

Restriction: You can specify an additional security access group only on Windows operating system. If you specify an additional security access group on other operating systems, the **crtmqm** command returns an error.

Related concepts

"Secure unshared queue manager data and log directories and files on Windows" on page 366

"Secure shared queue manager data and log directories and files on Windows" on page 363

Related tasks

"Create a multi-instance queue manager on domain workstations or servers" on page 340

Related reference

crtmqm

Installable services

You change installable services on Windows by using **regedit**, and on UNIX and Linux by using the Service stanza in the qm.ini file.

Note: There are significant implications to changing installable services and their components. For this reason, the installable services are read-only in the WebSphere MQ Explorer.

To change installable services on Windows systems, use regedit, or on UNIX and Linux systems, use the Service stanza in the qm.ini file. For each component within a service, you must also specify the name and path of the module containing the code for that component. On UNIX and Linux systems, use the ServiceComponent stanza for this.

Name=AuthorizationService|NameService
The name of the required service.

AuthorizationService
For WebSphere MQ, the Authorization Service component is known as the object authority manager, or OAM.

The AuthorizationService stanza and its associated ServiceComponent stanza are added automatically when the queue manager is created. Add other ServiceComponent stanzas manually.

NameService
No name service is provided by default. If you require a name service, you must add the NameService stanza manually.

EntryPoints=*number-of-entries*
The number of entry points defined for the service. This includes the initialization and termination entry points.

SecurityPolicy=Default|NTSIDsRequired (WebSphere MQ for Windows only)
The SecurityPolicy attribute applies only if the service specified is the default authorization service, that is, the OAM. The SecurityPolicy attribute allows you to specify the security policy for each queue manager. The possible values are:

Default
Use the default security policy to take effect. If a Windows security identifier (NT SID) is not passed to the OAM for a particular user ID, an attempt is made to obtain the appropriate SID by searching the relevant security databases.

NTSIDsRequired
Pass an NT SID to the OAM when performing security checks.

See Windows security identifiers (SIDs) for more information.

SharedBindingsUserId=*user-type*
The SharedBindingsUserId attribute applies only if the service specified is the default authorization service, that is, the OAM. The SharedBindingsUserId attribute is used with relation to shared bindings only. This value allows you to specify whether the *UserIdentifier* field in the *IdentityContext* structure, from the MQZ_AUTHENTICATE_USER function, is the effective user Id or the real user Id. For information on the MQZ_AUTHENTICATE_USER function, see MQZ_AUTHENTICATE_USER - Authenticate user . The possible values are:

Default
The value of the *UserIdentifier* field is set as the real user Id.

Real
The value of the *UserIdentifier* field is set as the real user Id.

Effective
The value of the *UserIdentifier* field is set as the effective user Id.

FastpathBindingsUserId=*user-type*
The FastpathBindingsUserId attribute applies only if the service specified is the default authorization service, that is, the OAM. The FastpathBindingsUserId attribute is used with relation to fastpath bindings only. This value allows you to specify whether the *UserIdentifier* field in the *IdentityContext* structure, from the MQZ_AUTHENTICATE_USER function, is the effective user Id or the real user Id. For information on the MQZ_AUTHENTICATE_USER function, see MQZ_AUTHENTICATE_USER - Authenticate user . The possible values are:

Default
The value of the *UserIdentifier* field is set as the real user Id.

Real
The value of the *UserIdentifier* field is set as the real user Id.

Effective
The value of the *UserIdentifier* field is set as the effective user Id.

IsolatedBindingsUserId =*user-type*
The IsolatedBindingsUserId attribute applies only if the service specified is the default authorization service, that is, the OAM. The IsolatedBindingsUserId attribute is used with relation to isolated bindings only. This value allows you to specify whether the *UserIdentifier* field in the *IdentityContext* structure, from the MQZ_AUTHENTICATE_USER function, is the effective user Id or the real user Id. For information on the MQZ_AUTHENTICATE_USER function, see MQZ_AUTHENTICATE_USER - Authenticate user . The possible values are:

Default
The value of the *UserIdentifier* field is set as the effective user Id.

Real
The value of the *UserIdentifier* field is set as the real user Id.

Effective
The value of the *UserIdentifier* field is set as the effective user Id.

For more information about installable services and components, see Installable services and components for UNIX, Linux and Windows.

For more information about security services in general, see Setting up security on Windows, UNIX and Linux systems.

Related reference
Installable services reference information

Service components
You must specify service component information when you add a new installable service. On Windows systems use `regedit`, and on UNIX and Linux systems use the `ServiceComponent` stanza in the qm.ini

file. The authorization service stanza is present by default, and the associated component, the OAM, is active.

Specify the service components as follows:

Service=*service_name*
The name of the required service. This must match the value specified on the Name attribute of the Service configuration information.

Name=*component_name*
The descriptive name of the service component. This must be unique and contain only characters that are valid for the names of WebSphere MQ objects (for example, queue names). This name occurs in operator messages generated by the service. We recommend that this name begins with a company trademark or similar distinguishing string.

Module=*module_name*
The name of the module to contain the code for this component. This must be a full path name.

ComponentDataSize=*size*
The size, in bytes, of the component data area passed to the component on each call. Specify zero if no component data is required.

For more information about installable services and components, see Installable services and components for UNIX, Linux and Windows.

Queue manager logs

Use the Log queue manager properties page from the IBM WebSphere MQ Explorer, or the Log stanza in the qm.ini file, to specify information about logging on a queue manager.

By default, these settings are inherited from the settings specified for the default log settings for the queue manager (described in "Log defaults for IBM WebSphere MQ" on page 415). Change these settings only if you want to configure this queue manager in a different way.

For information about calculating log sizes, see "Calculating the size of the log" on page 392.

Note: The limits given in the following parameter list are set by WebSphere MQ. Operating system limits might reduce the maximum possible log size.

LogPrimaryFiles=3 |*2-254* (Windows)|*2-510* (UNIX and Linux systems)
The log files allocated when the queue manager is created.

The minimum number of primary log files you can have is 2 and the maximum is 254 on Windows, or 510 on UNIX and Linux systems. The default is 3.

The total number of primary and secondary log files must not exceed 255 on Windows, or 511 on UNIX and Linux systems, and must not be less than 3.

The value is examined when the queue manager is created or started. You can change it after the queue manager has been created. However, a change in the value is not effective until the queue manager is restarted, and the effect might not be immediate.

LogSecondaryFiles=2 |*1-253* (Windows)|*1-509* (UNIX and Linux systems)
The log files allocated when the primary files are exhausted.

The minimum number of secondary log files is 1 and the maximum is 253 on Windows, or 509 on UNIX and Linux systems. The default number is 2.

The total number of primary and secondary log files must not exceed 255 on Windows, or 511 on UNIX and Linux systems, and must not be less than 3.

The value is examined when the queue manager is started. You can change this value, but changes do not become effective until the queue manager is restarted, and even then the effect might not be immediate.

LogFilePages=*number*
The log data is held in a series of files called log files. The log file size is specified in units of 4 KB pages.

The default number of log file pages is 4096, giving a log file size of 16 MB.

On UNIX and Linux systems the minimum number of log file pages is 64, and on Windows the minimum number of log file pages is 32; in both cases the maximum number is 65 535.

Note: The size of the log files specified during queue manager creation cannot be changed for a queue manager.

LogType=CIRCULAR |LINEAR
The type of logging to be used by the queue manager. You cannot change the type of logging to be used once the queue manager has been created. Refer to the description of the LogType attribute in "Log defaults for IBM WebSphere MQ" on page 415 for information about creating a queue manager with the type of logging you require.

CIRCULAR
Start restart recovery using the log to roll back transactions that were in progress when the system stopped.

See "Types of logging" on page 389 for a fuller explanation of circular logging.

LINEAR
For both restart recovery and media or forward recovery (creating lost or damaged data by replaying the contents of the log).

See "Types of logging" on page 389 for a fuller explanation of linear logging.

LogBufferPages=0 |*0-4096*
The amount of memory allocated to buffer records for writing, specifying the size of the buffers in units of 4 KB pages.

The minimum number of buffer pages is 18 and the maximum is 4096. Larger buffers lead to higher throughput, especially for larger messages.

If you specify 0 (the default), the queue manager selects the size. In WebSphere MQ Version 7.0 this is 512 (2048 KB).

If you specify a number in the range 1 through 17, the queue manager defaults to 18 (72 KB). If you specify a number in the range 18 through 4096, the queue manager uses the number specified to set the memory allocated.

The value is examined when the queue manager is started. The value can be increased or decreased within the limits stated. However, a change in the value is not effective until the next time the queue manager is started.

LogPath=*directory_name*
The directory in which the log files for a queue manager reside. This must exist on a local device to which the queue manager can write and, preferably, on a different drive from the message queues. Specifying a different drive gives added protection in case of system failure.

The default is:

- `C:\Program Files\IBM\WebSphere MQ\log` in WebSphere MQ for Windows .
- `/var/mqm/log` in WebSphere MQ for UNIX and Linux systems.

You can specify the name of a directory on the `crtmqm` command using the -ld flag. When a queue manager is created, a directory is also created under the queue manager directory, and this is used to hold the log files. The name of this directory is based on the queue manager name. This ensures that the log file path is unique, and also that it conforms to any limitations on directory name lengths.

If you do not specify -ld on the `crtmqm` command, the value of the `LogDefaultPath` attribute is used.

In WebSphere MQ for UNIX and Linux systems, user ID mqm and group mqm must have full authorities to the log files. If you change the locations of these files, you must give these authorities yourself. This is not required if the log files are in the default locations supplied with the product.

LogWriteIntegrity=SingleWrite|DoubleWrite| TripleWrite
The method the logger uses to reliably write log records.

TripleWrite
This is the default method.

Note, that you can select **DoubleWrite**, but if you do so, the system interprets this as **TripleWrite**.

SingleWrite
You should use **SingleWrite**, only if the file-system or device hosting the WebSphere MQ recovery log explicitly guarantees the atomicity of 4KB writes.

That is, when a write of a 4KB page fails for any reason, the only two possible states are either the before image, or the after image. No intermediate state should be possible.

Restricted mode

This option applies to UNIX and Linux systems only. The RestrictedMode stanza is set by the -g option on the **crtmqm** command. Do not change this stanza after the queue manager has been created. If you do not use the -g option, the stanza is not created in the qm.ini file.

There are some directories under which IBM WebSphere MQ applications create files while they are connected to the queue manager within the queue manager data directory. In order for applications to create files in these directories, they are granted world write access:

- /var/mqm/sockets/*QMgrName*/@ipcc/ssem/*hostname*/
- /var/mqm/sockets/*QMgrName*/@app/ssem/*hostname*/
- /var/mqm/sockets/*QMgrName*/zsocketapp/*hostname*/

where *<QMGRNAME>* is the name of the queue manager, and *<hostname>* is the host name.

On some systems, it is unacceptable to grant all users write access to these directories. For example, those users who do not need access the queue manager. Restricted mode modifies the permissions of the directories that store queue manager data. The directories can then only be accessed by members of the specified application group. The permissions on the System V IPC shared memory used to communicate with the queue manager are also modified in the same way.

The application group is the name of the group with members that have permission to do the following things:

- run MQI applications
- update all IPCC resources
- change the contents of some queue manager directories

To use restricted mode for a queue manager:

- The creator of the queue manager must be in the mqm group and in the application group.
- The mqm user ID must be in the application group.
- All users who want to administer the queue manager must be in the mqm group and in the application group.
- all users who want to run IBM WebSphere MQ applications must be in the application group.

Any MQCONN or MQCONNX call issued by a user who is not in the application group failed with reason code MQRC_Q_MGR_NOT_AVAILABLE.

Restricted mode operates with the IBM WebSphere MQ authorization service. Therefore you must also grant users the authority to connect to IBM WebSphere MQ and access the resources they require using the IBM WebSphere MQ authorization service.

Windows UNIX Linux Further information about configuring the IBM WebSphere MQ authorization service can be found in Setting up security on Windows, UNIX and Linux systems.

Only useIBM WebSphere MQ restricted mode when the control provided by the authorization service does not provide sufficient isolation of queue manager resources.

XA resource managers

Use the XA `resource manager` queue manager properties page from the IBM WebSphere MQ Explorer, or the XA`ResourceManager` stanza in the qm.ini file, to specify the following information about the resource managers involved in global units of work coordinated by the queue manager.

Add XA resource manager configuration information manually for each instance of a resource manager participating in global units of work; no default values are supplied.

See Database coordination for more information about resource manager attributes.

Name=*name* (mandatory)
This attribute identifies the resource manager instance.

The `Name` value can be up to 31 characters in length. You can use the name of the resource manager as defined in its XA-switch structure. However, if you are using more than one instance of the same resource manager, you must construct a unique name for each instance. You can ensure uniqueness by including the name of the database in the `Name` string, for example.

WebSphere MQ uses the `Name` value in messages and in output from the `dspmqtrn` command.

Do not change the name of a resource manager instance, or delete its entry from the configuration information, once the associated queue manager has started and the resource manager name is in effect.

SwitchFile=*name* (mandatory)
The fully-qualified name of the load file containing the resource manager's XA switch structure.

If you are using a 64-bit queue manager with 32-bit applications, the `name` value should contain only the base name of the load file containing the resource manager's XA switch structure.

The 32-bit file will be loaded into the application from the path specified by `ExitsDefaultPath`.

The 64-bit file will be loaded into the queue manager from the path specified by `ExitsDefaultPath64`.

XAOpenString=*string* (optional)
The string of data to be passed to the resource manager's xa_open entry point. The contents of the string depend on the resource manager itself. For example, the string could identify the database that this instance of the resource manager is to access. For more information about defining this attribute, see:

- Adding resource manager configuration information for DB2
- Adding resource manager configuration information for Oracle
- Adding resource manager configuration information for Sybase
- Adding resource manager configuration information for Informix®

and consult your resource manager documentation for the appropriate string.

XACloseString=*string* (optional)
The string of data to be passed to the resource manager's xa_close entry point. The contents of the string depend on the resource manager itself. For more information about defining this attribute, see:

- Adding resource manager configuration information for DB2
- Adding resource manager configuration information for Oracle
- Adding resource manager configuration information for Sybase
- Adding resource manager configuration information for Informix

and consult your database documentation for the appropriate string.

ThreadOfControl=THREAD|PROCESS
This attribute is mandatory for WebSphere MQ for Windows. The queue manager uses this value for serialization when it needs to call the resource manager from one of its own multithreaded processes.

THREAD
The resource manager is fully *thread aware*. In a multithreaded WebSphere MQ process, XA function calls can be made to the external resource manager from multiple threads at the same time.

PROCESS
The resource manager is not *thread safe*. In a multithreaded WebSphere MQ process, only one XA function call at a time can be made to the resource manager.

The `ThreadOfControl` entry does not apply to XA function calls issued by the queue manager in a multithreaded application process. In general, an application that has concurrent units of work on different threads requires this mode of operation to be supported by each of the resource managers.

Attributes of channels stanzas

These attributes determine the configuration of a channel.

This information is not applicable to WebSphere MQ for the z/OS platform.

Use the `Channels` queue manager properties page from the WebSphere MQ Explorer, or the `CHANNELS` stanza in the `qm.ini` file, to specify information about channels.

MaxChannels=100 |*number*
The maximum number of *current* channels allowed.

The value must be in the range 1 - 65535. The default is 100.

MaxActiveChannels=*MaxChannels_value*
The maximum number of channels allowed to be *active* at any time. The default is the value specified for the `MaxChannels` attribute.

MaxInitiators=3 |*number*
The maximum number of initiators. The default and maximum value is 3.

MQIBindType=FASTPATH| STANDARD
The binding for applications:

FASTPATH
Channels connect using `MQCONNX` FASTPATH; there is no agent process.

STANDARD
Channels connect using STANDARD.

PipeLineLength=1 |*number*
The maximum number of concurrent threads a channel will use. The default is 1. Any value greater than 1 is treated as 2.

When you use pipelining, configure the queue managers at both ends of the channel to have a *PipeLineLength* greater than 1.

Note: Pipelining is only effective for TCP/IP channels.

AdoptNewMCA=NO|SVR|SDR|RCVR|CLUSRCVR|ALL|FASTPATH
If WebSphere MQ receives a request to start a channel, but finds that an instance of the channel is already running, in some cases the existing channel instance must be stopped before the new one can start. The `AdoptNewMCA` attribute allows you to control which types of channels can be ended in this way.

If you specify the `AdoptNewMCA` attribute for a particular channel type, but the new channel fails to start because a matching channel instance is already running:

1. The new channel tries to stop the previous one by requesting it to end.

2. If the previous channel server does not respond to this request by the time the AdoptNewMCATimeout wait interval expires, the thread or process for the previous channel server is ended.
3. If the previous channel server has not ended after step 2, and after the AdoptNewMCATimeout wait interval expires for a second time, WebSphere MQ ends the channel with a `CHANNEL IN USE` error.

The AdoptNewMCA functionality applies to server, sender, receiver, and cluster-receiver channels. In the case of a sender or server channel, only one instance of a channel with a particular name can be running in the receiving queue manager. In the case of a receiver or cluster-receiver channel, multiple instances of a channel with a particular name might be running in the receiving queue manager, but only one instance can run at any one time from a particular remote queue manager.

Note: AdoptNewMCA is not supported on requester or server-connection channels.

Specify one or more values, separated by commas or blanks, from the following list:

NO
The `AdoptNewMCA` feature is not required. This is the default.

SVR
Adopt server channels.

SDR
Adopt sender channels.

RCVR
Adopt receiver channels.

CLUSRCVR
Adopt cluster receiver channels.

ALL
Adopt all channel types except FASTPATH channels.

FASTPATH
Adopt the channel if it is a FASTPATH channel. This happens only if the appropriate channel type is also specified, for example: `AdoptNewMCA=RCVR,SVR,FASTPATH`.

Attention!: The AdoptNewMCA attribute might behave in an unpredictable fashion with FASTPATH channels. Exercise great caution when enabling the AdoptNewMCA attribute for FASTPATH channels.

AdoptNewMCATimeout=60 |1 - 3600
The amount of time, in seconds, that the new channel instance waits for the old channel instance to end. Specify a value in the range 1 - 3600. The default value is 60.

AdoptNewMCACheck=QM|ADDRESS|NAME|ALL
The type of checking required when enabling the `AdoptNewMCA` attribute. If possible, perform full checking to protect your channels from being shut down, inadvertently or maliciously. At the very least, check that the channel names match.

Specify one or more of the following values, separated by commas or blanks in the case of *QM*, *NAME*, or *ALL*:

QM
Check that the queue manager names match.

Note that the queue manager name itself is matched, not the QMID.

ADDRESS
Check the communications source IP address. For example, the TCP/IP address.

Note: Comma separated CONNAME values apply to target addresses and are, therefore, not relevant to this option.

In the case that a multi-instance queue manager fails over from `hosta` to `hostb`, any outbound channels from that queue manager will use the source IP address of `hostb`. If this is different from `hosta`, then AdoptNewMCACheck=*ADDRESS* fails to match.

You can use SSL or TLS with mutual authentication to prevent an attacker from disrupting an existing running channel. Alternatively, use an HACMP type solution with IP-takeover instead of multi-instance queue managers, or use a network load balancer to mask the source IP address.

NAME
Check that the channel names match.

ALL
Check for matching queue manager names, the communications address, and for matching channel names.

The default is `AdoptNewMCACheck=NAME,ADDRESS,QM`.

Related concepts
“Channel states” on page 54
A channel can be in one of many states at any time. Some states also have substates. From a given state a channel can move into other states.

TCP, LU62, NETBIOS, and SPX

Use these queue manager properties pages, or stanzas in the qm.ini file, to specify network protocol configuration parameters. They override the default attributes for channels.

TCP
Use the TCP queue manager properties page from the IBM WebSphere MQ Explorer, or the TCP stanza in the qm.ini file, to specify Transmission Control Protocol/Internet Protocol (TCP/IP) configuration parameters.

Port=1414|*port_number*
The default port number, in decimal notation, for TCP/IP sessions. The *well known* port number for WebSphere MQ is 1414.

Library1=*DLLName1* (WebSphere MQ for Windows only)
The name of the TCP/IP sockets DLL.

The default is WSOCK32.

KeepAlive=NO|YES
Switch the KeepAlive function on or off. KeepAlive=YES causes TCP/IP to check periodically that the other end of the connection is still available. If it is not, the channel is closed.

ListenerBacklog=number
Override the default number of outstanding requests for the TCP/IP listener.

When receiving on TCP/IP, a maximum number of outstanding connection requests is set. This can be considered to be a *backlog* of requests waiting on the TCP/IP port for the listener to accept the request. The default listener backlog values are shown in Table 33 on page 429.

Table 33. Default outstanding connection requests (TCP)

Platform	Default ListenerBacklog value
Windows Server	100
Windows Workstation	5
Linux	100
Solaris	100
HP-UX	20
AIX V4.2 or later	100

Table 33. Default outstanding connection requests (TCP) (continued)	
Platform	**Default ListenerBacklog value**
AIX V4.1 or earlier	10

Note: Some operating systems support a larger value than the default shown. Use this to avoid reaching the connection limit.

Conversely, some operating systems might limit the size of the TCP backlog, so the effective TCP backlog could be smaller than requested here.

If the backlog reaches the values shown in Table 33 on page 429, the TCP/IP connection is rejected and the channel cannot start. For message channels, this results in the channel going into a RETRY state and retrying the connection at a later time. For client connections, the client receives an MQRC_Q_MGR_NOT_AVAILABLE reason code from MQCONN and retries the connection at a later time.

SvrSndBuffSize=32768|number
The size in bytes of the TCP/IP send buffer used by the server end of a client-connection server-connection channel.

SvrRcvBuffSize=32768|number
The size in bytes of the TCP/IP receive buffer used by the server end of a client-connection server-connection channel.

Connect_Timeout=0|number
The number of seconds before an attempt to connect the socket times out. The default value of zero specifies that there is no connect timeout.

LU62 (WebSphere MQ for Windows only)
Use the LU6.2 queue manager properties page from the IBM WebSphere MQ Explorer, or the LU62 stanza in the qm.ini file, to specify SNA LU 6.2 protocol configuration parameters.

TPName
The TP name to start on the remote site.

Library1=*DLLName 1*
The name of the APPC DLL.

The default value is WCPIC32.

Library2=*DLLName2*
The same as Library1, used if the code is stored in two separate libraries.

The default value is WCPIC32.

NETBIOS (WebSphere MQ for Windows only)
Use the Netbios queue manager properties page from the IBM WebSphere MQ Explorer, or the NETBIOS stanza in the qm.ini file, to specify NetBIOS protocol configuration parameters.

LocalName=*name*
The name by which this machine is known on the LAN.

AdapterNum=0|*adapter_number*
The number of the LAN adapter. The default is adapter 0.

NumSess=1|*number_of_sessions*
The number of sessions to allocate. The default is 1.

NumCmds=1|*number_of_commands*
The number of commands to allocate. The default is 1.

NumNames=1|*number_of_names*
The number of names to allocate. The default is 1.

Library1=*DLLName1*
The name of the NetBIOS DLL.

The default value is NETAPI32.

SPX (WebSphere MQ for Windows only)
Use the SPX queue manager properties page from the IBM WebSphere MQ Explorer, or the SPX stanza in the qm.ini file, to specify SPX protocol configuration parameters.

Socket=5E86|*socket_number*
The SPX socket number in hexadecimal notation. The default is X'5E86'.

BoardNum=0|*adapter_number*
The LAN adapter number. The default is adapter 0.

KeepAlive=NO|YES
Switch the KeepAlive function on or off.

KeepAlive=YES causes SPX to check periodically that the other end of the connection is still available. If it is not, the channel is closed.

Library1=*DLLName1*
The name of the SPX DLL.

The default is WSOCK32.DLL.

Library2=*DLLName2*
The same as LibraryName1, used if the code is stored in two separate libraries.

The default is WSOCK32.DLL.

ListenerBacklog=number
Override the default number of outstanding requests for the SPX listener.

When receiving on SPX, a maximum number of outstanding connection requests is set. This can be considered to be a *backlog* of requests waiting on the SPX socket for the listener to accept the request. The default listener backlog values are shown in Table 34 on page 431.

Table 34. Default outstanding connection requests (SPX)

Platform	Default ListenerBacklog value
Windows Server	100
Windows Workstation	5

Note: Some operating systems support a larger value than the default shown. Use this to avoid reaching the connection limit.

Conversely, some operating systems might limit the size of the SPX backlog, so the effective SPX backlog could be smaller than requested here.

If the backlog reaches the values shown in Table 34 on page 431, the SPX connection is rejected and the channel cannot start. For message channels, this results in the channel going into a RETRY state and retrying the connection at a later time. For client connections, the client receives an MQRC_Q_MGR_NOT_AVAILABLE reason code from MQCONN and should retry the connection at a later time.

Exit path

Use the Exits queue manager properties page from the IBM WebSphere MQ Explorer, or the ExitPath stanza in the qm.ini file to specify the path for user exit programs on the queue manager system.

ExitsDefaultPath=*string*
The ExitsDefaultPath attribute specifies the location of:

- 32-bit channel exits for clients
- 32-bit channel exits and data conversion exits for servers
- Unqualified XA switch load files

ExitsDefaultPath64=*string*
The ExitsDefaultPath64 attribute specifies the location of:

- 64-bit channel exits for clients
- 64-bit channel exits and data conversion exits for servers
- Unqualified XA switch load files

API exits

For a server, use the `Exits` queue manager properties page from the IBM WebSphere MQ Explorer, or the `ApiExitLocal` stanza in the `qm.ini` file to identify API exit routines for a queue manager. For a client modify the `ApiExitLocal` stanza in the mqclient.ini file to identify API exit routines for a queue manager.

On Windows systems, you can also use the amqmdain command to change the entries for API exits. (To identify API exit routines for all queue managers, you use the `ApiExitCommon` and `ApiExitTemplate` stanzas, as described in "API exits" on page 418.)

Note, that for the API exit to work correctly, the message from the server must be sent to the client unconverted. After the API exit has processed the message, the message must then be converted on the client. This, therefore, requires that you have installed all conversion exits on the client.

For a complete description of the attributes for these stanzas, see Configuring API exits.

QMErrorLog stanza on UNIX, Linux, and Windows

Use the `Extended` queue manager properties page from the WebSphere MQ Explorer, or the `QMErrorLog` stanza in the qm.ini file to tailor the operation and contents of queue manager error logs.

Attention: You can use WebSphere MQ Explorer to make the changes, only if you are using a local queue manager on the Windows platform.

ErrorLogSize=*maxsize*
Specifies the size of the queue manager error log at which it is copied to the backup. *maxsize* must be in the range 32768 through 2147483648 bytes. If `ErrorLogSize` is not specified, the default value of 2097152 bytes (2 MB) is used.

ExcludeMessage=*msgIds*
Specifies messages that are not to be written to the queue manager error log. If your WebSphere MQ system is heavily used, with many channels stopping and starting, a large number of information messages are sent to the z/OS console and hardcopy log. The WebSphere MQ-IMS bridge and buffer manager might also produce a large number of information messages, so excluding messages prevents you from receiving a large number of messages if you require it. *msqIds* contain a comma-separated list of message id's from the following:

5211 - Maximum property name length exceeded.
5973 - Distributed publish/subscribe subscription inhibited
5974 - Distributed publish/subscribe publication inhibited
6254 - The system could not dynamically load shared library
7234 - Number of messages loaded
9001 - Channel program ended normally
9002 - Channel program started
9202 - Remote host not available
9208 - Error on receive from host
9209 - Connection closed
9228 - Cannot start channel responder
9489 - SVRCONN max instances limit exceeded
9490 - SVRCONN max instances per client limit exceeded
9508 - Cannot connect to queue manager
9524 - Remote queue manager unavailable

9528 - User requested closure of channel
9558 - Remote Channel is not available
9637 - Channel is lacking a certificate
9776 - Channel was blocked by user ID
9777 - Channel was blocked by NOACCESS map
9782 - Connection was blocked by address
9999 - Channel program ended abnormally

SuppressMessage=*msgIds*
Specifies messages that are written to the queue manager error log once only in a specified time interval. If your WebSphere MQ system is heavily used, with many channels stopping and starting, a large number of information messages are sent to the z/OS console and hardcopy log. The WebSphere MQ-IMS bridge and buffer manager might also produce a large number of information messages, so suppressing messages prevents you from receiving a number of repeating messages if you require it. The time interval is specified by `SuppressInterval`. *msqIds* contain a comma-separated list of message id's from the following:

5211 - Maximum property name length exceeded.
5973 - Distributed publish/subscribe subscription inhibited
5974 - Distributed publish/subscribe publication inhibited
6254 - The system could not dynamically load shared library
7234 - Number of messages loaded
9001 - Channel program ended normally
9002 - Channel program started
9202 - Remote host not available
9208 - Error on receive from host
9209 - Connection closed
9228 - Cannot start channel responder
9489 - SVRCONN max instances limit exceeded
9490 - SVRCONN max instances per client limit exceeded
9508 - Cannot connect to queue manager
9524 - Remote queue manager unavailable
9528 - User requested closure of channel
9558 - Remote Channel is not available
9637 - Channel is lacking a certificate
9776 - Channel was blocked by user ID
9777 - Channel was blocked by NOACCESS map
9782 - Connection was blocked by address
9999 - Channel program ended abnormally

If the same message id is specified in both `SuppressMessage` and `ExcludeMessage`, the message is excluded.

SuppressInterval=*length*
Specifies the time interval, in seconds, in which messages specified in `SuppressMessage` are written to the queue manager error log once only. *length* must be in the range 1 through 86400 seconds. If `SuppressInterval` is not specified, the default value of 30 seconds is used.

Queue manager default bind type

Use the `Extended` queue manager properties page from the IBM WebSphere MQ Explorer, or the `Connection` stanza in the qm.ini file to specify the default bind type.

DefaultBindType=SHARED|ISOLATED
If DefaultBindType is set to ISOLATED, applications and the queue manager run in separate processes, and no resources are shared between them.

If DefaultBindType is set to SHARED, applications and the queue manager run in separate processes, but some resources are shared between them.

The default is SHARED.

SSL and TLS stanza of the queue manager configuration file

Use the SSL stanza of the queue manager configuration file to configure SSL or TLS channels on your queue manager.

Online Certificate Status Protocol (OCSP)

A certificate can contain an AuthorityInfoAccess extension. This extension specifies a server to be contacted through Online Certificate Status Protocol (OCSP). To allow SSL or TLS channels on your queue manager to use AuthorityInfoAccess extensions, ensure that the OCSP server named in them is available, is correctly configured, and is accessible over the network. For more information, see Working with revoked certificates.

CrlDistributionPoint (CDP)

A certificate can contain a CrlDistributionPoint extension. This extension contains a URL which identifies both the protocol used to download a certificate revocation list (CRL) and also the server to be contacted.

If you want to allow SSL or TLS channels on your queue manager to use CrlDistributionPoint extensions, ensure that the CDP server named in them is available, correctly configured, and accessible over the network.

The SSL Stanza

Use the SSL stanza in the `qm.ini` file to configure how SSL or TLS channels on your queue manager attempts to use the following facilities, and how they react if problems occur when using them.

In each of the following cases, if the value supplied is not one of the valid values listed, then the default value is taken. No error messages are written mentioning that an invalid value is specified.

CDPCheckExtensions=YES|NO

CDPCheckExtensions specifies whether SSL or TLS channels on this queue manager try to check CDP servers that are named in CrlDistributionPoint certificate extensions.

- YES: SSL or TLS channels try to check CDP servers to determine whether a digital certificate is revoked.
- NO: SSL or TLS channels do not try to check CDP servers. This value is the default.

OCSPAuthentication=REQUIRED|WARN|OPTIONAL

OCSPAuthentication specifies the action to be taken when a revocation status cannot be determined from an OCSP server.

If OCSP checking is enabled, an SSL or TLS channel program attempts to contact an OCSP server.

If the channel program is unable to contact any OCSP servers, or if no server can provide the revocation status of the certificate, then the value of the OCSPAuthentication parameter is used.

- REQUIRED: Failure to determine the revocation status causes the connection to be closed with an error. This value is the default.
- WARN: Failure to determine the revocation status causes a warning message to be written in the queue manager error log, but the connection is allowed to proceed.
- OPTIONAL: Failure to determine the revocation status allows the connection to proceed silently. No warnings or errors are given.

OCSPCheckExtensions=YES|NO

OCSPCheckExtensions specifies whether SSL and TLS channels on this queue manager try to check OCSP servers that are named in AuthorityInfoAccess certificate extensions.

- YES: SSL and TLS channels try to check OCSP servers to determine whether a digital certificate is revoked. This value is the default.
- NO: SSL and TLS channels do not try to check OCSP servers.

SSLHTTPProxyName=*string*

The string is either the host name or network address of the HTTP Proxy server that is to be used by GSKit for OCSP checks. This address can be followed by an optional port number, enclosed in parentheses. If you do not specify the port number, the default HTTP port, 80, is used. On the HP-UX PA-RISC and Sun Solaris SPARC platforms, and for 32-bit clients on AIX, the network address can only be an IPv4 address; on other platforms it can be an IPv4 or IPv6 address.

This attribute might be necessary if, for example, a firewall prevents access to the URL of the OCSP responder.

Exit properties

Use the Cluster queue manager properties page from the IBM WebSphere MQ Explorer, or the ExitPropertiesLocal stanza in the qm.ini file, to specify information about exit properties on a queue manager. Alternatively, you can set it using the **amqmdain** command.

By default, this setting is inherited from the CLWLMode attribute in the ExitProperties stanza of the machine-wide configuration (described in "Exit properties" on page 415). Change this setting only if you want to configure this queue manager in a different way. This value can be overridden for individual queue managers using the cluster workload mode attribute on the Cluster queue manager properties page.

CLWLMode=SAFE|FAST

The cluster workload (CLWL) exit allows you to specify which cluster queue in the cluster to open in response to an MQI call (for example, MQOPEN, MQPUT). The CLWL exit runs either in FAST mode or SAFE mode depending on the value you specify on the CLWLMode attribute. If you omit the CLWLMode attribute, the cluster workload exit runs in SAFE mode.

SAFE

Run the CLWL exit in a separate process from the queue manager. This is the default.

If a problem arises with the user-written CLWL exit when running in SAFE mode, the following happens:

- The CLWL server process (amqzlwa0) fails.
- The queue manager restarts the CLWL server process.
- The error is reported to you in the error log. If an MQI call is in progress, you receive notification in the form of a return code.

The integrity of the queue manager is preserved.

Note: Running the CLWL exit in a separate process can affect performance.

FAST

Run the cluster exit inline in the queue manager process.

Specifying this option improves performance by avoiding the process switching costs associated with running in SAFE mode, but does so at the expense of queue manager integrity. You should only run the CLWL exit in FAST mode if you are convinced that there are **no** problems with your CLWL exit, and you are particularly concerned about performance.

If a problem arises when the CLWL exit is running in FAST mode, the queue manager will fail and you run the risk of the integrity of the queue manager being compromised.

Subpool

This stanza is created by WebSphere MQ. Do not change it.

The stanza Subpool, and attribute ShortSubpoolName within that stanza, are written automatically by WebSphere MQ when you create a queue manager. WebSphere MQ chooses a value for ShortSubpoolName. Do not alter this value.

The name corresponds to a directory and symbolic link created inside the /var/mqm/sockets directory, which WebSphere MQ uses for internal communications between its running processes.

Configuring HP Integrity NonStop Server

Use this information to help you to configure your IBM WebSphere MQ client for HP Integrity NonStop Server installation.

For details about configuring a client by using a configuration file, see "Configuring a client using a configuration file" on page 123.

For details about configuring a client by using environment variables, see "Using WebSphere MQ environment variables" on page 140.

If you are performing IBM WebSphere MQ client for HP Integrity NonStop Server operations under TMF/Gateway, see the subtopics for information about how to configure the TMF/Gateway. Included are an overview of the Gateway process, configuring the Gateway to run under Pathway, and configuring the client initialization file to enable your IBM WebSphere MQ client for HP Integrity NonStop Server to reach the TMF Gateway.

This section also contains IBM WebSphere MQ client for HP Integrity NonStop Server specific information about granting permissions to channels.

Gateway process overview

The HP NonStop Transaction Management Facility (TMF) provides services to enable a gateway process to register as a resource manager. The IBM WebSphere MQ provided TMF/Gateway process runs under Pathway.

IBM WebSphere MQ registers a single gateway process for each queue manager that is coordinated by TMF, therefore you must configure a separate TMF/Gateway for each queue manager that is to participate in TMF coordinated units of work. This registration is so that each queue manager is an independent resource manager, and for administrative purposes, registering each queue manager once with HP NonStop TMF results in an easy to understand mapping.

For multiple installations of IBM WebSphere MQ, you must nominate a single gateway process from one of these installations for each queue manager to be coordinated by TMF.

The interface to the gateway process supports any client at the same version or earlier.

For more information about administering the gateway process, see Administering HP Integrity NonStop Server.

Configuring Gateway to run under Pathway

TMF/Gateway is the interface between the HP NonStop Transaction Management Facility (TMF) and IBM WebSphere MQ that enables TMF to be the transaction coordinator for IBM WebSphere MQ transactions.

The IBM WebSphere MQ provided TMF/Gateway converts transactions from TMF coordination into eXtended Architecture (XA) transaction coordination to communicate with the remote queue manager.

You must have one TMF/Gateway per queue manager that requires coordination, and client configuration is required so that the client can connect to the correct Gateway.

The TMF/Gateway can use all the mechanisms available to the client to communicate with a queue manager. Configure the TMF/Gateway in the way you would for your other applications.

The TMF/Gateway is not a HP Integrity NonStop Server process pair and is designed to run in a Pathway environment. The TMF/Gateway creates permanent resources within TMF, which it reuses on subsequent runs, therefore the TMF/Gateway must always be run under the same user authority.

Defining the serverclass

TMF/Gateway is hosted as a serverclass within a Pathway environment. To define the serverclass, you must set the following server attributes:

PROCESSTYPE=OSS
Specifies the type of servers in the serverclass. The Gateway process is a multi-threaded OSS program. This attribute is mandatory, and must be set to OSS.

MAXSERVERS=1
Specifies the maximum number of server processes in this serverclass that can run at the same time. There can be only a single Gateway process for any queue manager. This attribute is mandatory and must be set to 1.

NUMSTATIC=1
Specifies the maximum number of static servers within this serverclass. The Gateway process must be run as a static server. This attribute is mandatory and must be set to 1.

TMF=ON
Specifies whether servers in this serverclass can lock and update data files that are audited by the TMF subsystem. The Gateway process participates in the TMF transactions of IBM WebSphere MQ client applications therefore this attribute must be set to ON.

PROGRAM=<OSS installation path>/opt/mqm/bin/runmqtmf
For IBM WebSphere MQ client for IBM WebSphere MQ, this attribute must be `runmqtmf`. This attribute must be the absolute OSS path name. Case is significant.

ARGLIST=-*m<QMgr name>*[,-c<channel name>][,-p<port>][,-h<host name>][,-n<max threads>]
These attributes provide parameters to the Gateway process, where:

- `QMgrName` is the name of the queue manager for this Gateway process. If you are using a queue sharing group (or other port distribution technology), this parameter must be targeted to a specific queue manager. This parameter is mandatory.
- `channel name` is the name of the server channel on the queue manager to be used by the Gateway process. This parameter is optional.
- `port` is the TCP/IP port for the queue manager. This parameter is optional.
- `host name` is the host name for the queue manager. This parameter is optional.
- `max threads` is the maximum number of worker threads that are created by the Gateway process. This parameter can be a value of 10 or greater. The lowest value that is used is 10 even if a value lower than 10 is specified. If no value is provided, the Gateway process creates up to a maximum of 50 threads.

Use the `-c`, `-p`, and `-h` attributes as an alternative method of providing connection information to the Gateway, in addition to that described in "Configuring the TMF/Gateway using environment variables" on page 438. If you specify one or more, but not all of the `-c`, `-p`, and `-h` attributes, then those attributes that you do not specify default to the following values:

- `channel name` defaults to `SYSTEM.DEF.SVRCONN`
- `host name` defaults to `localhost`
- `port` defaults to 1414

If any of the parameters you supply are invalid, the TMF/Gateway issues diagnostic message AMQ5379 to the error log and terminates.

OWNER=ID
The user ID under which the Gateway runs and that must be granted connect authority to the queue manager.

SECURITY="value"
Specifies the users, in relation to the Owner attribute, who can access the Gateway from a IBM WebSphere MQ client application.

LINKDEPTH and MAXLINKS must be configured with values appropriate for the expected number of IBM WebSphere MQ client applications that might want to concurrently communicate with the Gateway. If these values are set too low, you might see occurrences of the error message AMQ5399 issued from client applications.

For more information about these server attributes, see the *HP NonStop TS/MP 2.5 System Management Manual*.

Configuring the TMF/Gateway using environment variables

One of the most commonly used methods to define the TMF/Gateway is to set the MQSERVER environment variable, for example:

```
ENV MQSERVER=<channel name>/<transport>/<host name>(<listener port>)
```

ENV at the beginning of the command is Pathway notation.

Configuring the client initialization file

If you are using the HP NonStop Transaction Management Facility (TMF), you must have an IBM WebSphere MQ client initialization file to enable your IBM WebSphere MQ client for the HP Integrity NonStop Server to reach the TMF Gateway.

An IBM WebSphere MQ client initialization file for HP Integrity NonStop Server can be held in a number of locations, for more information, see “Location of the client configuration file” on page 124.

For details of the contents of the configuration file, together with an example, see “Configuring a client using a configuration file” on page 123. Use the TMF stanza to specify the TMF queue manager and server details, for more information, see “TMF and TMF/Gateway stanzas” on page 140.

An example of the entries for a IBM WebSphere MQ client for HP Integrity NonStop Server is:

```
TMF:
  PathMon=$PSD1P

TmfGateway:
  QManager=MQ5B
  Server=MQ-MQ5B

TmfGateway:
  QManager=MQ5C
  Server=MQ-MQ5C
```

For more information about configuring a client using environment variables, see “Using WebSphere MQ environment variables” on page 140.

Granting permissions to channels

Granting permissions to channels on IBM WebSphere MQ client for HP Integrity NonStop Server is identical to other operating systems, however you must know the identification of the owner that the gateway is running under.

You can then use the identification of the owner of the gateway to grant appropriate permissions. The important difference is that granting permissions to queue manager channels is not under the authority of any application.

Use the **setmqaut** command to both to grant an authorization, that is, give a IBM WebSphere MQ principal or user group permission to perform an operation, and to revoke an authorization, that is, remove the permission to perform an operation.

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 IBM 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®, 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. 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.

Part Number:

(1P) P/N: