WebSphere MQ

Queue Manager Clusters

SC34-6061-02

Note!

Before using this information and the product it supports, be sure to read the general information under "Notices", on page 155.

Third edition (December 2002)

This is the third edition of this book that applies to WebSphere MQ.

This edition applies to the following WebSphere MQ V5.3 products:
- WebSphere MQ for AIX
- WebSphere MQ for HP-UX
- WebSphere MQ for iSeries
- WebSphere MQ for Linux for Intel
- WebSphere MQ for Linux for zSeries
- WebSphere MQ for Solaris
- WebSphere MQ for Windows
- WebSphere MQ for z/OS

Unless otherwise stated, the information also applies to these products:
- MQSeries for OS/2 Warp, V5.1
- MQSeries for Compaq Tru64 UNIX, V5.1
- MQSeries for Compaq OpenVMS Alpha, V5.1
- MQSeries for Compaq NonStop Kernel, V5.1
- MQSeries for Sun Solaris, Intel Platform Edition, V5.1

Contents

Figures

Tables

About this book

This book describes how to create and use *clusters* of WebSphere® MQ queue managers. It explains the concepts and terminology of clustering and shows how you can benefit by taking advantage of clustering. It details changes to the message queue interface (MQI), and summarizes the syntax of new and changed WebSphere MQ commands. It shows a number of examples of tasks you can perform to set up and maintain clusters of queue managers.

The term UNIX® systems is used to denote the following UNIX operating systems:

- AIX®
- Compaq Tru64 UNIX
- HP-UX
- Linux (for Intel and zSeries™)
- Solaris

The term Windows® is used throughout this book to denote the following Microsoft® systems:

- Windows NT®
- Windows 2000
- Windows XP

z/OS™ means any release of z/OS or OS/390® that supports the current version of WebSphere MQ.

Who should read this book

This book is for anyone who needs to understand WebSphere MQ clusters. The following readers are specifically addressed:

- Network planners responsible for designing the overall queue manager network
- Application programmers responsible for designing applications that access queues and queue managers within clusters
- Systems administrators responsible for monitoring the local system and implementing some of the planning details
- System programmers with responsibility for designing and programming the user exits

What you need to know to understand this book

This book describes WebSphere MQ clustering in detail, and includes step-by-step examples that you should be able to follow with only limited background knowledge about WebSphere MQ in general. An understanding of the concepts of message queuing, for example the purpose of queues, queue managers, and channels, would be an advantage.

To understand fully how to make the best use of clusters, it is useful to be familiar with the WebSphere MQ products for the specific platforms you will be using, and the communications protocols that are used on those platforms. It is also helpful to have an understanding of how distributed queue management works. These topics are discussed in the *WebSphere MQ Intercommunication*.

How to use this book

This book contains three parts. The chapters in Part 1, “Getting started with queue manager clusters”, on page 1 are aimed at users who are new to clusters. Read these chapters first to learn what queue manager clusters are and how to use them. Throughout this part of the book, the use of clusters is compared with more traditional distributed queuing techniques. If you are not familiar with distributed queuing, skip the sections that are not of interest to you. You should still be able to follow the guidance and examples given. The chapters in this part are:

- Chapter 1, “Concepts and terminology”, on page 3, which introduces the concepts of queue manager clusters, explains the associated terminology, and highlights the differences between using clusters and using distributed queuing techniques.
- Chapter 2, “Using clusters to ease system administration”, on page 11, which shows the benefits of using clusters and shows when and where you might choose to implement them in your existing network.
- Chapter 3, “First tasks”, on page 19, which describes some of the first steps in setting up and using a cluster. You should be able to accomplish these first tasks without an in-depth understanding of clusters or distributed queuing.

The chapters in Part 2, “Using queue manager clusters”, on page 31 are aimed at more experienced users who want to understand about clusters in detail. Read these chapters to learn how to use clusters to the best advantage. The chapters in this part are:

- Chapter 4, “How queue manager clusters work”, on page 35, which provides more detail about the components of clusters and explains how clustering works.
- Chapter 5, “Using clusters for workload management”, on page 47, which describes how to use clusters to achieve workload balance.
- Chapter 6, “Using WebSphere MQ commands with clusters”, on page 61, which introduces commands that are specific to work with WebSphere MQ clusters.
- Chapter 7, “Managing WebSphere MQ clusters”, on page 71, which provides administrative information about how to design and maintain a cluster.
- Chapter 8, “Keeping clusters secure”, on page 81, which discusses security aspects associated with using clusters.
- Chapter 9, “Advanced tasks”, on page 87, which guides you through a series of more advanced tasks.

The chapters in Part 3, “Reference information”, on page 119 contain reference information about the cluster workload exit. The chapters in this part are:

- Chapter 10, “Cluster workload exit call and data structures”, on page 121.
- Chapter 12, “Constants for the cluster workload exit”, on page 147.

Summary of changes

This section describes changes in this edition of *WebSphere MQ Queue Manager Clusters*. Changes since the previous edition of the book are marked by vertical lines to the left of the changes.

Changes for this edition (SC34-6061-02)

This edition provides additions and clarifications for users of Version 5.1 of MQSeries® for Compaq NonStop Kernel, MQSeries for Compaq OpenVMS Alpha, and MQSeries for Compaq Tru64 UNIX.

Changes for the previous editions (SC34-6061-00 and -01)

The first two editions for WebSphere MQ included the following changes:

- Changes throughout the book to reflect the rebranding of MQSeries to WebSphere MQ.
- Adding the platforms Windows XP, Linux for zSeries, and Linux for Intel.
- A new section on how long queue manager repositories retain information
- The introduction of LU6.2 examples
- The introduction of DHCP examples
- The introduction of the Secure Sockets Layer (SSL)
- Two new options on the REFRESH CLUSTER command REPOS(YES/NO)
- Two new options on the RESET CLUSTER command QUEUES(YES/NO)
- The introduction of * as CLUSTER NAME in REFRESH CLUSTER
- A new problem solving section.
- Information on the new data structure MQXCLWLN

Changes

Part 1. Getting started with queue manager clusters

Getting started

Chapter 1. Concepts and terminology

This chapter introduces the concepts of queue manager clusters and explains some of the terminology. For the benefit of customers familiar with traditional distributed-queuing techniques, it compares the use of clusters with the use of distributed queuing. If you are not familiar with distributed queuing, you should skip the sections that are not of interest to you.

Concepts

Businesses are increasingly becoming aware of the advantages of establishing an intranet or of connecting processors to a LAN. You might also have connected some z/OS processors to form a sysplex, or some AIX processors in the form of an SP2®. Processors linked in these ways benefit from support from each other and have access to a far wider range of programs and data.

In the same way, WebSphere MQ queue managers can be connected to form a *cluster*. This facility is available to queue managers on the following platforms:
- WebSphere MQ for AIX, V5.3
- WebSphere MQ for iSeries™, V5.3
- WebSphere MQ for HP-UX, V5.3
- WebSphere MQ for z/OS, V5.3
- WebSphere MQ for Solaris, V5.3
- WebSphere MQ for Windows, V5.3
- WebSphere MQ for Linux for Intel and Linux for zSeries, V5.3
- MQSeries for Compaq Tru64 UNIX, V5.1
- MQSeries for Compaq OpenVMS Alpha, V5.1
- MQSeries for Compaq NonStop Kernel, V5.1
- MQSeries for Sun Solaris, Intel Platform Edition, V5.1
- MQSeries for OS/2® Warp, V5.1

You can connect the queue managers using any of the communications protocols that are available on your platform. That is, TCP or LU 6.2 on any platform, NetBIOS or SPX on OS/2 or Windows, and UDP on AIX. Connections on more than one protocol can exist within a cluster. Of course, if you try to make a connection to a queue manager using a protocol that it does not support, the channel will not become active.

Comparison with distributed queuing

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 it must have defined:
- A transmission queue
- A channel to the remote queue manager

Figure 1 on page 4 shows the components required for distributed queuing.

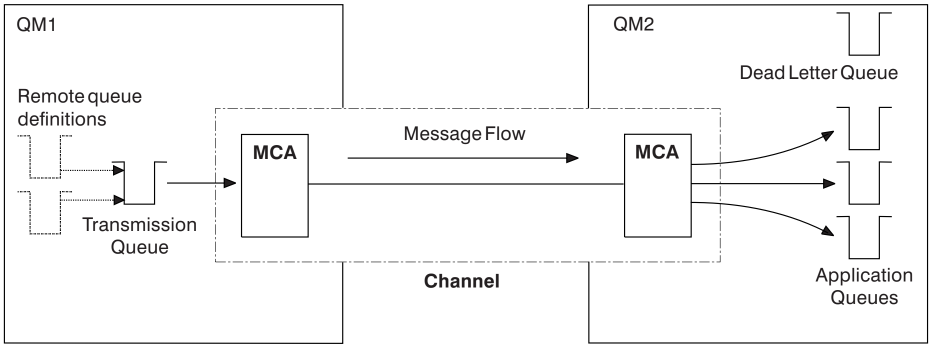

Figure 1. Distributed queuing

Distributed queing is when you group queue managers in a cluster, the queue managers can make the queues that they host available to every other queue manager in the cluster. Any queue manager can send a message to any other queue manager in the same cluster without explicit 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

Overview of cluster components

Figure 2 on page 5 shows the components of a cluster called CLUSTER.

- In this cluster there are three queue managers, QM1, QM2, and QM3.
- QM1 and QM2 host repositories of information about the queue managers in the cluster. They are referred to as *full repository queue managers*. (The repositories are represented in the diagram by the shaded cylinders.)
- QM2 and QM3 host some queues that are accessible to any other queue manager in the cluster. These are called *cluster queues*. (The cluster queues are represented in the diagram by the shaded queues.)

  As with distributed queuing, an application uses the MQPUT call to put a message on a cluster queue at *any* queue manager. An application uses the MQGET call to retrieve messages from a cluster queue on the local queue manager.
- Each queue manager has a definition for the receiving end of a channel called TO.*qmgr* on which it can receive messages. This is a *cluster-receiver channel*. A cluster-receiver channel is similar to a receiver channel used in distributed queuing, but in addition to carrying messages this channel can also carry information about the cluster.
- Each queue manager also has a definition for the sending end of a channel, which connects to the cluster-receiver channel of one of the full repository queue managers. This is a *cluster-sender channel*. In Figure 2 on page 5, QM1 and QM3 have cluster-sender channels connecting to TO.QM2. QM2 has a cluster-sender channel connecting to TO.QM1. A cluster-sender channel is similar to a sender channel used in distributed queuing, but in addition to carrying messages this channel can also carry information about the cluster.

  Once both the cluster-receiver end and the cluster-sender end of a channel have been defined, the channel starts automatically.

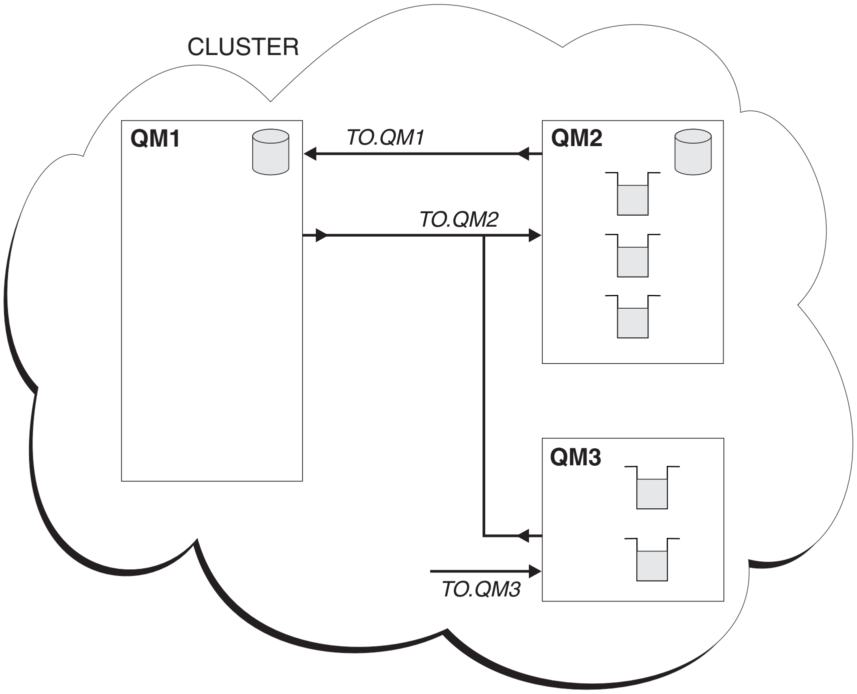

Figure 2. A cluster of queue managers

Terminology

Before proceeding to the next chapter it is useful to understand the following terminology:

Cluster
A cluster is a network of queue managers that are logically associated in some way. The queue managers in a cluster may be physically remote. For example, they might represent the branches of an international chain store and be physically located in different countries. Each cluster within an enterprise should have a unique name.

Cluster queue manager
A cluster queue manager is a queue manager that is a member of a cluster. A queue manager may be a member of more than one cluster. (See "Overlapping clusters" on page 74.) 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 just 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. For more information about queue-sharing groups see the *WebSphere MQ for z/OS Concepts and Planning Guide*.

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).

When you make or alter a definition on a cluster queue manager, the information is sent to the full repository queue manager and the repositories in the cluster are updated accordingly.

Cluster queue

A cluster queue is a queue that is hosted by a cluster queue manager and made available to other queue managers in the cluster. The cluster queue manager makes a local queue definition for the queue, specifying the name of the cluster where the queue is to be found. This definition has the effect of showing the other queue managers in the cluster that the queue is there. 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.

A cluster queue can be a queue that is shared by members of a queue-sharing group in WebSphere MQ for z/OS.

Repository

A repository is a collection of information about the queue managers that are members of a cluster. This information includes queue manager names, their locations, their channels, which queues they host, and so on. The information is stored in the form of messages on a queue called SYSTEM.CLUSTER.REPOSITORY.QUEUE. (This queue is one of the default objects created when you start a WebSphere MQ queue manager, except on WebSphere MQ for z/OS where it is defined as part of queue manager customization.) Typically, two queue managers in a cluster hold a *full repository*. The remaining queue managers all hold a *partial repository*.

Repository queue manager

A repository queue manager is a cluster queue manager that holds a *full* repository. To ensure availability, set up two or more full repository queue managers in each cluster. The full repository queue managers receive information sent by the other queue managers in the cluster and update their repositories accordingly. The full repository queue managers send messages to each other to be sure that they are both kept up to date with new information about the cluster.

Full repository and partial repository

A queue manager that hosts a *complete* set of information about every queue manager in the cluster is referred to as having a *full repository* for the cluster.

The other queue managers in the cluster inquire about the information in the full repositories and build up their own subsets of this information in partial repositories. A queue manager’s 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 will send them the new information. For much of the time a queue manager’s partial repository has all the information it 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, except on WebSphere MQ for z/OS where it is defined as part of queue manager customization.

Cluster-receiver channel

A cluster-receiver (CLUSRCVR) channel definition defines the receiving 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 repository. By defining the cluster-receiver channel, the queue manager shows to 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.

Cluster-sender channel

A cluster-sender (CLUSSDR) channel definition defines the sending end of a channel on which a cluster queue manager can send cluster information to one of the full repositories. The cluster-sender channel is used to notify the repository of any changes to the queue manager's status, for example the addition or removal of a queue. 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 queue manager's CLUSSDR channel definition points to. Once the initial contact has been made, further cluster queue manager objects are defined automatically as necessary so that the queue manager can send cluster information to every full repository, and messages to every queue manager.

Cluster transmission queue

Each cluster queue manager has a cluster transmission queue called SYSTEM.CLUSTER.TRANSMIT.QUEUE. The cluster transmission queue transmits all messages from the queue manager to any other queue manager that is in the same cluster. This queue is one of the default cluster transmission queues, except on WebSphere MQ for z/OS where it is defined as part of queue manager customization.

Binding

You can create a cluster in which more than one queue manager hosts an instance of the same cluster queue. This is discussed in "More than one instance of a queue" on page 47. If you do this, make sure that a sequence of messages are all 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 (see "MQOPEN" on page 55).

Benefits

There are two reasons for using clusters:

1. Reduced system administration.

   As soon as you start to establish even a small cluster you will benefit from simplified system administration. Establishing a network of queue managers in a cluster involves fewer definitions than establishing a network that is to use distributed queuing. With fewer definitions to make, you can set up or change your network more quickly and easily, and reduce the risk of making an error in your definitions.
2. Increased availability and workload balancing.

   Simple clusters give you easier system administration. Moving to more complicated clusters, offers improved scalability of the number of instances of a queue you can define, providing greater availability. Because you can define instances of the same queue on more than one queue manager, the workload can be distributed throughout the queue managers in a cluster.

These two objectives are discussed in detail in Chapter 2, “Using clusters to ease system administration”, on page 11 and Chapter 5, “Using clusters for workload management”, on page 47.

Things to consider

Consider the following before starting to use clusters:

- On z/OS, if you are using CICS®, you must use the WebSphere MQ mover (not the CICS mover) to take part in clustering.
- To get the most out of clusters, all the queue managers in the network must be on a platform that supports clusters. Until all your systems are on platforms that support clusters, you might have queue managers outside a cluster that cannot access your cluster queues without extra manual definitions.
- 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 as usual. (If there is no dead-letter queue, the channel fails and retries, as described in the *WebSphere MQ Intercommunication* book.)
- 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. However, as with distributed queuing, there is a risk that you may 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 may be cumbersome.
- The WebSphere MQ Explorer cannot administer a cluster with repository queue managers on WebSphere MQ for z/OS. You must nominate an additional repository on a system that the WebSphere MQ Explorer can administer.
- The purpose of distribution lists, which are supported on WebSphere MQ for AIX, iSeries, HP-UX, Solaris, Linux, and Windows V5.3 and MQSeries for OS/2 Warp, Compaq Tru64 UNIX, Compaq NonStop Kernel, and Compaq OpenVMS Alpha, V5.1, is to use a single MQPUT command to send the same message to multiple destinations. You can use distribution lists in conjunction with queue

manager clusters. However, in a clustering environment all the messages are expanded at MQPUT time and so the advantage, in terms of network traffic, is not so great as in a non-clustering environment. The advantage of distribution lists, from the administrator's point of view, 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 to 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**.
- If you use the MQOO_BIND_ON_OPEN option on an MQOPEN call to force messages to be sent to a specific destination, and the destination queue manager is not available, the messages are not delivered. Messages are not routed to another queue manager because of the risk of duplication.
- If a queue manager is to host a cluster's 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 were to use DHCP, the IP address would be subject to change because DHCP can allocate a new IP address each time you restart a system. Therefore, it would not be possible to specify the IP address in the CLUSSDR definitions. Even if all your CLUSSDR definitions specified the hostname rather than the IP address, the definitions would still not be reliable. This is because DHCP does not necessarily update the DNS directory entry for the host with the new address.

  Note: Unless you have installed software that guarantees to keep your DNS directory up-to-date, you should not nominate queue managers as full repositories if they are on systems that use DHCP.
- 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 from a local cluster queue, but you can PUT to any queue in a cluster. If you open a queue to use the MQGET command, the queue manager will only use the local queue.

Summary of concepts

If you are familiar with WebSphere MQ and distributed queuing, think of a cluster as a network of queue managers maintained by a conscientious systems administrator. Whenever you create a receiver channel or define a queue, the systems administrator automatically creates corresponding sender channels and remote-queue definitions on the other queue managers.

You do not need to make transmission queue definitions because WebSphere MQ provides a transmission queue on each queue manager. This single transmission queue can be used to carry messages to any other queue manager.

All the queue managers that join a cluster agree to work in this way. They send out information about themselves and about the queues they host, and they receive information about the other members of the cluster.

This information is stored in repositories. Most queue managers retain only the information that they need, that is, information about queues and queue managers with which they need to communicate. Some queue managers retain a full repository of **all** the information about **all** queue managers in the cluster.

A cluster-receiver channel is a communication channel similar to a receiver channel. When you define a cluster-receiver channel, not only is the object created on your queue manager, but also information about the channel and the queue manager that owns it is stored in the repositories. The definition of a cluster-receiver channel is a queue manager's initial introduction to a cluster. Once it has been defined, other queue managers can automatically make corresponding definitions for the cluster-sender end of the channel as needed.

A cluster-sender channel is a communication channel similar to a sender channel. You need a cluster-sender channel only if you want to communicate with another cluster queue manager. When another cluster queue manager wants to communicate with you, your cluster-sender channel is created automatically by reference to the appropriate cluster-receiver channel definition. However, each queue manager must have one manually defined cluster-sender channel, through which it makes its initial contact with the cluster.

Queue managers on platforms that support clusters do not have to be part of a cluster. You can continue to use distributed queuing techniques as well as, or instead of, using clusters.

Chapter 2. Using clusters to ease system administration

This chapter describes how you can use clusters to simplify system administration in your environment. It is intended for users who have not used clusters before and who want to learn how they might benefit from setting up and using a simple cluster. This chapter covers:

- “How can I use clusters?”
- “How does the system administrator benefit?” on page 12
- “What about my applications?” on page 13
- “How do I set up a cluster?” on page 14

For information about how to set up a more complex cluster that benefits from workload management, refer to Chapter 5, “Using clusters for workload management”, on page 47.

How can I use clusters?

Typically a cluster contains queue managers that are logically related in some way and need to share some data or applications. For example you might have one queue manager for each department in your company, managing data and applications specific to that department. You could group all these queue managers into a cluster so that they all feed into the PAYROLL application. Or you might have one queue manager for each branch of your chain store, managing the stock levels and other information for that branch. If you group these queue managers into a cluster, they can all access the same set of SALES and PURCHASES applications, which are held centrally, perhaps on the head-office queue manager.

Once a cluster has been set up, the queue managers within it can communicate with each other without extra channel definitions or remote-queue definitions.

You can convert an existing network of queue managers into a cluster or you can establish a cluster as part of setting up a new network.

A WebSphere MQ client can connect to a queue manager that is part of a cluster, just as it can connect to any other queue manager. See the *WebSphere MQ Clients* book for more information about clients.

Can I use clusters and queue-sharing groups?

On WebSphere MQ for z/OS you can group queue managers into queue-sharing groups. A queue manager in a queue-sharing group can define a local queue that is to be shared by up to 32 queue managers. For more information about queue-sharing groups see the *WebSphere MQ for z/OS Concepts and Planning Guide*.

Shared queues can also be *cluster queues*. Furthermore, the queue managers in a queue-sharing group can also be in one or more clusters. See “Task 10: Adding new queue managers that host a shared queue” on page 108 for an example task showing how to use clusters in combination with queue-sharing groups.

How does the system administrator benefit?

Using clusters leads to easier administration of a network. Look at Figure 3, which shows four queue managers each with two queues. Let us consider how many definitions are needed to connect these queue managers using distributed queuing. Then we will see how many definitions are needed to set up the same network as a cluster.

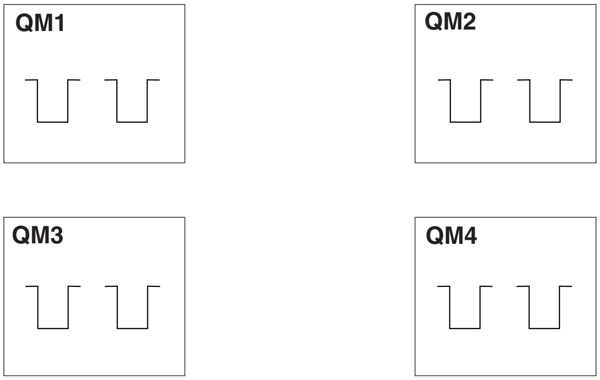

Figure 3. A network of four queue managers

Definitions to set up a network using distributed queuing

To set up the network shown in Figure 3 using distributed queuing, you might have the following definitions:

Table 1. 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
Optionally, on z/OS, a process definition specifying trigger data if channels are to be triggered	3	12

While you might reduce this number of definitions by, for example, using generic receiver-channel definitions, the maximum number of definitions could be as many as 20 on each queue manager, which is a total of 80 for this network.

Definitions to set up a network using clusters

When using clusters, you need:

- Just one CLUSSDR and one CLUSRCVR definition at each queue manager
- No separately defined transmission queues
- No remote-queue definitions

To set up the network shown in Figure 3 on page 12 using clusters you need the following definitions:

Table 2. 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
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 would need 4 definitions on each queue manager — a total of 16 definitions all together. You would also need to alter the queue-manager definitions for two of the queue managers, to make them full repository queue managers for the cluster.

The CLUSSDR and CLUSRCVR channel definitions need be made only once. When the cluster is in place you can add or remove queue managers (other than the repository queue managers) without any disruption to the other queue managers.

Clearly, this amounts to a significant reduction in the number of definitions required to set up a network containing a large number of queue managers.

With fewer definitions to make there is less risk of error:

- Object names will always match, for example the channel name in a sender-receiver pair.
- The transmission queue name specified in a channel definition will always match the correct transmission queue definition or the transmission queue name specified in a remote queue definition.
- A QREMOTE definition will always point to the correct queue at the remote queue manager.

Furthermore, 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.

What about my applications?

You need not alter any of your applications if you are going to set up a simple WebSphere MQ cluster. The applications name the target queue on the MQOPEN call as usual and need not be concerned about the location of the queue manager.

However, if you set up a cluster in which there are multiple definitions for the same queue, as described in Chapter 5, “Using clusters for workload management” , on page 47, you must review your applications and modify them as necessary.

How do I set up a cluster?

After installing the product, you have to create queue managers. Any queue manager you create is capable of working in a cluster.

Having decided that you want to create a cluster of queue managers, you need to consider which queue managers in the cluster are to hold the full repositories of cluster information. You can choose any number of queue managers for this purpose but the recommended number is two. See “Selecting queue managers to hold full repositories” on page 71 for more information.

The smallest possible cluster contains only two queue managers. In this case both queue managers contain full repositories. You need only a small number of definitions to set this up, and yet there is a high degree of autonomy at each queue manager.

Figure 4 shows a cluster of two queue managers. You can set up a cluster like this using WebSphere MQ Script commands (MQSC), or any other type of administration command or utility that is available on your platform. See Chapter 6, “Using WebSphere MQ commands with clusters”, on page 61 for more information.

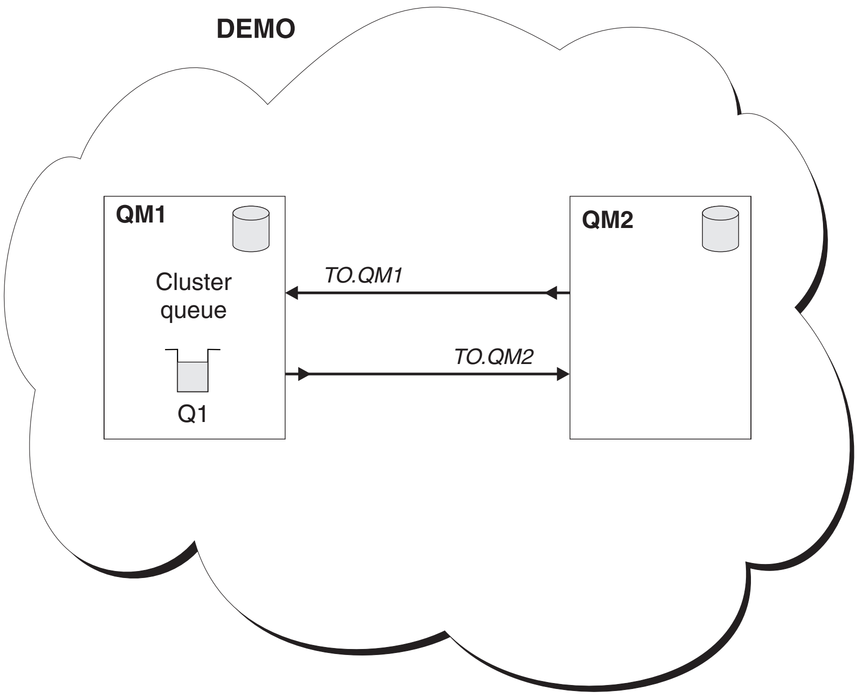

Figure 4. A small cluster of two queue managers

Setting up a cluster like this is described in “Task 1: Setting up a new cluster” on page 19.

Establishing communication in a cluster

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, NetBIOS or SPX on OS/2 or Windows NT, and UDP on AIX. Configuring communication links is described in detail in the

WebSphere MQ Intercommunication book. As part of this configuration, you also need channel initiators and channel listeners just as you do with distributed queuing.

Channel initiator

All cluster queue managers need a channel initiator to monitor the system-defined initiation queue SYSTEM.CHANNEL.INITQ. This is the initiation queue for all transmission queues including the cluster transmission queue.

WebSphere MQ for z/OS
There is one channel initiator for each queue manager and it runs as a separate address space. You start it using the MQSC START CHINIT, which you issue as part of your queue manager startup.

Platforms other than z/OS
When you start a queue manager, a channel initiator is automatically started.

Channel listener

Run a channel listener program on each queue manager. A channel listener program ‘listens’ for incoming network requests and starts the appropriate receiver channel when it is needed.

The implementation of channel listeners is platform specific.

WebSphere MQ for z/OS
Use the channel listener program provided by WebSphere MQ. To start a WebSphere MQ channel listener, use the MQSC command START LISTENER, which you issue as part of your channel initiator startup. For example:

```
START LISTENER PORT(1414) TRPTYPE(TCP)
```

or

```
START LISTENER LUNAME(LONDON.LUNAME) TRPTYPE(LU62)
```

As well as a listener for each queue manager, members of a queue-sharing group can make use of a shared listener. Do not use shared listeners in conjunction with clusters. If you do, queue managers might receive messages pertaining to queues for which they do not have a CLUSRCVR definition.

WebSphere MQ for iSeries
Use the channel listener program provided by WebSphere MQ. To start a WebSphere MQ channel listener use the CL command STRMQMLSR. For example:

```
STRMQMLSR MQMNAME(QM1) PORT(1414)
```

Alternatively, use the MQSC command START LISTENER.

MQSeries for OS/2 Warp
Use either the channel listener program provided by WebSphere MQ, inetd, or the facilities provided by the operating system (for example, Attach manager for LU 6.2 communications).

To start the WebSphere MQ channel listener use the RUNMQLSR command. For example:

```
RUNMQLSR -t tcp -p 1414 -m QM1
```

Alternatively, use the MQSC command START LISTENER.

To use inetd to start channels, configure two files:

1. Edit the file TCPIP\ETC\SERVICES. If you do not have the following line in that file, add it as shown:

   ```
   MQSeries     1414/tcp      # Websphere MQ channel listener
   ```

   where 1414 is the port number required for WebSphere MQ. You can change this, but it must match the port number specified at the sending end.
2. Edit the file TCPIP\ETC\INETD.LST. If you do not have the following line in that file, add it as shown:

   ```
   MQSeries      tcp C:\MQM\BIN\AMQCRSTA -m queue.manager.name
   ```

   where *queue.manager.name* is the name of your queue manager. If you have MQSeries for OS/2 Warp installed on a different drive, replace the `C:` with the correct drive letter.

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
```

Alternatively, use the MQSC command START LISTENER.

WebSphere MQ on UNIX 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
```

Alternatively, use the MQSC command START LISTENER.

To use inetd to start channels, configure two files:

1. Edit the file /etc/services. (To do this you must be logged in as a superuser or root.) If you do not have the following line in that file, add it as shown:

   ```
   MQSeries       1414/tcp      # Websphere MQ channel listener
   ```

   where 1414 is the port number required by WebSphere MQ. You can change this, but it must match the port number specified at the sending end.
2. Edit the file /etc/inetd.conf. If you do not have the following line in that file, add it as shown:

   For AIX:

   ```
   MQSeries stream tcp nowait mqm /usr/mqm/bin/amqcrsta amqcrsta
   -m queue.manager.name
   ```

   For Compaq Tru64 UNIX , HP-UX, Solaris and Linux:

   ```
   MQSeries stream tcp nowait mqm /opt/mqm/bin/amqcrsta amqcrsta
   -m queue.manager.name
   ```

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 Compaq Tru64 UNIX, Solaris or Linux:

1. Find the process ID of the inetd with the command:
   ```
   ps -ef | grep inetd
   ```
2. Run the command:
   ```
   kill -1 inetd processid
   ```

MQSeries for Compaq NonStop Kernel

Use either the channel listener program provided by MQSeries (runmqlsr), or the PATHCOM command.

To start the channel listener using the runmqlsr command, use a command of the form:

```
runmqlsr -t tcp -p 1414 -m QM1
```

For more information, see *MQSeries for Compaq NonStop Kernel System Administration*.

To start the channel listener using PATHCOM, use a command of the form:

```
START SERVER MQS-TCPLISxx
```

MQSeries for Compaq OpenVMS Alpha

For information on setting up communications, see *WebSphere MQ Intercommunication*.

Chapter 3. First tasks

This chapter shows how to perform these tasks:
- "Task 1: Setting up a new cluster"
- "Task 2a: Adding a new queue manager to a cluster" on page 27
- "Task 2b: Adding a new queue manager to a cluster — using DHCP" on page 28

Much of the information you need for these tasks is elsewhere in the WebSphere MQ library. This chapter gives pointers to that information and fills in details relating specifically to work with clusters.

Notes:
1. Throughout the examples in this chapter and Chapter 9, "Advanced tasks", on page 87, the queue managers have illustrative names such as LONDON and NEWYORK. Don't forget that on WebSphere MQ for z/OS, queue-manager names are limited to 4 characters.
2. The names of the queue managers imply that each queue manager is on a separate machine. You could just as easily set up these examples with all the queue managers on the same machine.
3. The examples in these chapters show WebSphere MQ Script Commands (MQSC) as they would be entered by the system administrator at the command console. For information about other ways of entering commands, refer to Chapter 6, "Using WebSphere MQ commands with clusters", on page 61.

Task 1: Setting up a new cluster

Scenario:
- You are setting up a new 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.
- Examples given using either TCP/IP or LU 6.2.

Figure 5 on page 24 shows what this cluster looks like.

Note: On WebSphere MQ for Windows you can use one of the wizards supplied with WebSphere MQ Explorer to create a new cluster similar to the one created by this task.

The steps required to complete task 1

To set up this cluster, follow these steps.

1. Decide on the organization of the cluster and its name

You have 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, but is a good way to start and provides scope for future expansion. When you open new branches of your store, you will be able to add the new queue managers to the cluster easily and without any disruption to the existing network. “Task 2a: Adding a new queue manager to a cluster” on page 27 describes how to do this.

For the time being the only application you are running is the inventory application. The cluster name is INVENTORY.

2. Determine which queue managers should hold full repositories

In any cluster you need to nominate at least one queue manager, or preferably two, to hold full repositories. See “Selecting queue managers to hold full repositories” on page 71 for more information. In this example there are only two queue managers, LONDON and NEWYORK, both of which hold full repositories.

Notes:

1. You can perform the remaining steps in any order.
2. As you proceed through the steps, warning messages might be written to the queue-manager log or the z/OS system console if you have yet to make some expected definitions.

```
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.
```

3. 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, you need to alter the queue-manager definition, using the ALTER QMGR command and specifying the REPOS attribute:

```
ALTER QMGR REPOS(INVENTORY)
```

If you enter

```
     1 : ALTER QMGR REPOS(INVENTORY)
AMQ8005: Websphere MQ queue manager changed.
```

1. C:\..runmqsc jupiter.queue.manager
2. ALTER QMGR REPOS(INVENTORY) (as shown above)

jupiter.queue.manager will be changed to a repository.

Note: If you just runmqsc and enter the ALTER QMGR command, the local queue manager will be changed.

4. Define the CLUSRCVR channels

On every queue manager in a cluster you need to define a cluster-receiver channel on which the queue manager can receive messages. This definition defines the queue manager’s connection name and the CLUSTER keyword shows the queue manager’s availability to receive messages from other queue managers in the cluster. The queue manager’s connection name is stored in the repositories, where other queue managers can refer to it.

Choose one of the following examples

Using transport protocol TCP/IP:

On the LONDON queue manager, define:

```
DEFINE CHANNEL(TO.LONDON) CHLTYPE(CLUSRCVR) TRPTYPE(TCP)
CONNAME(LONDON.CHSTORE.COM) CLUSTER(INVENTORY)
DESCR('TCP Cluster-receiver channel for queue manager LONDON')
```

```
     1 : DEFINE CHANNEL(TO.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'
```

In this example the channel name is TO.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 either as an alphanumeric DNS hostname, or as a dotted-decimal IP address.) Do not allow the CONNAME to specify a generic name.

On the NEWYORK queue manager, define:

```
DEFINE CHANNEL(TO.NEWYORK) CHLTYPE(CLUSRCVR) TRPTYPE(TCP)
CONNAME(NEWYORK.CHSTORE.COM) CLUSTER(INVENTORY)
DESCR('TCP Cluster-receiver channel for queue manager NEWYORK')
```

Using transport protocol LU 6.2: On the LONDON queue manager, define:

```
DEFINE CHANNEL(TO.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(TO.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'
```

In this example the channel name is TO.LONDON, and the connection name (CONNAME) specifies the LU name of the queue manager, which is LONDON.LUNAME. Do not allow the CONNAME to specify a generic LU name.

On the NEWYORK queue manager, define:

```
DEFINE CHANNEL(TO.NEWYORK) CHLTYPE(CLUSRCVR) TRPTYPE(LU62)
CONNAME(NEWYORK.LUNAME) CLUSTER(INVENTORY)
MODENAME('#INTER') TPNAME('MQSERIES')
DESCR('LU62 Cluster-receiver channel for queue manager NEWYORK')
```

5. Define the CLUSSDR channels

On every queue manager in a cluster you need to define one cluster-sender channel on which the queue manager can send messages to one of the full repository queue managers. 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. Note that the channel names given on the CLUSSDR definitions must match those on the corresponding CLUSRCVR definitions.

Choose one of the following examples.

Using transport protocol TCP/IP:

On the LONDON queue manager, define:

```
DEFINE CHANNEL(TO.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(TO.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.
```

On the NEWYORK queue manager, define:

```
DEFINE CHANNEL(TO.LONDON) CHLTYPE(CLUSSDR) TRPTYPE(TCP)
CONNAME(LONDON.CHSTORE.COM) CLUSTER(INVENTORY)
DESCR('TCP Cluster-sender channel from NEWYORK to repository at LONDON')
```

Using transport protocol LU 6.2:

On the LONDON queue manager, either define:

```
DEFINE CHANNEL(TO.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')
```

or

```
DEFINE CHANNEL(TO.NEWYORK) CHLTYPE(CLUSSDR) TRPTYPE(LU62)
CONNAME(CPIC) CLUSTER(INVENTORY)
DESCR('LU62 Cluster-sender channel from LONDON to repository at NEWYORK')
```

```
     1 : DEFINE CHANNEL(TO.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.
```

or

```
     2 : DEFINE CHANNEL(TO.NEWYORK) CHLTYPE(CLUSSDR) TRPTYPE(LU62)
         CONNAME(CPIC) CLUSTER(INVENTORY)
         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.
```

On the NEWYORK queue manager, define:

```
DEFINE CHANNEL(TO.LONDON) CHLTYPE(CLUSSDR) TRPTYPE(LU62)
CONNAME(LONDON.LUNAME) CLUSTER(INVENTORY)
DESCR('LU62 Cluster-sender channel from NEWYORK to repository at LONDON')
```

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.

6. 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.

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 and, on all platforms, start a listener program on each queue manager. The listener program listens for incoming network requests and starts the cluster-receiver channel when it is needed. See "Establishing communication in a cluster" on page 14 for more information.

The cluster achieved by task 1

The cluster set up by this task looks like this:

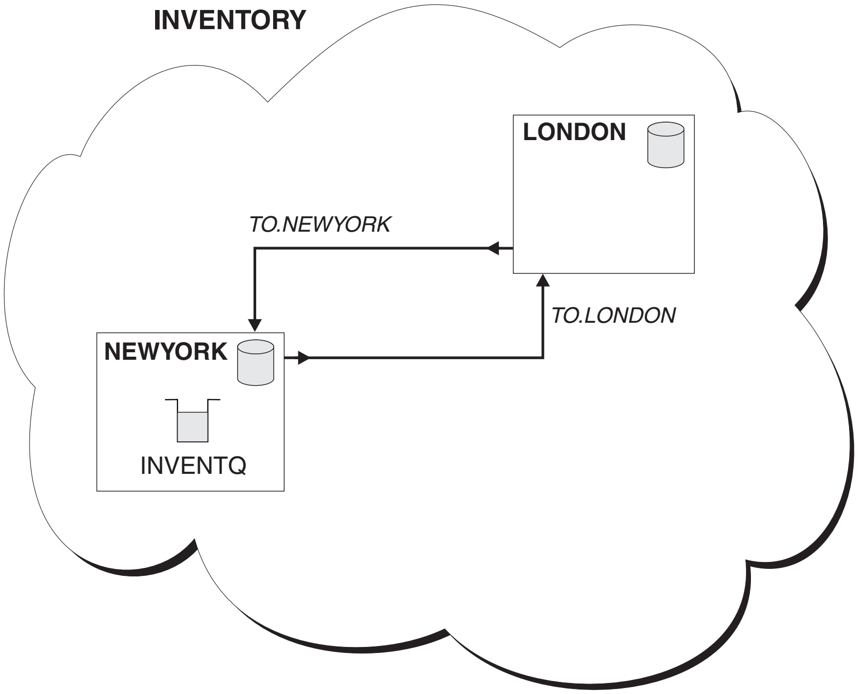

Figure 5. The INVENTORY cluster with two queue managers

Clearly, this is a very 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.

Converting an existing network into a cluster

If you are converting an existing network into a cluster like this, in step 6, you need to alter the existing queue definition. You also need to delete the remote queue definition at LONDON for the INVENTQ queue. See “Task 7: Converting an existing network into a cluster” on page 97 for an example of this.

Verifying task 1

Issue some DISPLAY commands to verify the cluster that you have set up. The responses you see should be similar to those shown in the examples that follow.

From the NEWYORK queue manager, issue the command:

```
dis clusqmgr(*)
```

```
     1 : dis clusqmgr(*)
AMQ8441: Display Cluster Queue Manager details.
   CLUSQMGR(NEWYORK)             CLUSTER(INVENTORY)
   CHANNEL(TO.NEWYORK)
AMQ8441: Display Cluster Queue Manager details.
   CLUSQMGR(LONDON)              CLUSTER(INVENTORY)
   CHANNEL(TO.LONDON)
```

Now issue the corresponding DISPLAY CHANNEL STATUS command:

```
dis chstatus(*)
```

```
     1 : dis chstatus(*)
AMQ8417: Display Channel Status details.
   CHANNEL(TO.NEWYORK)           XMITQ( )
   CONNAME(9.20.40.24)           CURRENT
   CHLTYPE(CLUSRCVR)             STATUS(RUNNING)
AMQ8417: Display Channel Status details.
   CHANNEL(TO.LONDON)            XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)
   CONNAME(9.20.51.25)           CURRENT
   CHLTYPE(CLUSSDR)              STATUS(RUNNING)
```

For more details on troubleshooting see “Troubleshooting” on page 110.

Using the cluster set up in task 1

Because the INVENTQ queue has been advertised to the cluster there is no need for remote-queue definitions. Applications running on NEWYORK and applications running on LONDON can put messages to the INVENTQ queue. They can receive responses to their messages by providing a reply-to queue and specifying its name when they put messages.

Using sample programs: Test your setup by sending some messages between the two queue managers, using amqsput. In the following example LONDON puts a message to the INVENTQ at NEWYORK:

1. On LONDON issue the command:
   - amqsput INVENTQ LONDON
2. Type some messages
3. On NEW YORK issue the command:
   - amqsget INVENTQ NEWYORK
4. You should now see the messages you entered on LONDON

Using your own programs: Test your setup by sending some messages between the two queue managers. In the following example LONDON puts a message to the INVENTQ at NEWYORK and receives a reply on its queue LONDON_reply.

1. Define a local queue called LONDON_reply
2. Set the MQOPEN options to MQOO_OUTPUT

3. Issue the MQOPEN call to open the queue INVENTQ
4. Set the ReplyToQ name in the message descriptor to LONDON_reply
5. Issue the MQPUT call to put the message
6. Commit the message

On NEWYORK:

1. Set the MQOPEN options to MQOO_BROWSE
2. Issue the MQOPEN call to open the queue INVENTQ
3. Issue the MQGET call to get the message from INVENTQ
4. Retrieve the ReplyToQ name from the message descriptor
5. Put the ReplyToQ name in the ObjectName field of the object descriptor
6. Set the MQOPEN options to MQOO_OUTPUT
7. Issue the MQOPEN call to open LONDON_reply at queue manager LONDON
8. Issue the MQPUT call to put the message to LONDON_reply

On LONDON:

1. Set the MQOPEN options to MQOO_BROWSE
2. Issue the MQOPEN call to open the queue LONDON_reply
3. Issue the MQGET call to get the message from LONDON_reply

Note: The definition for the local queue LONDON_reply does not need the CLUSTER attribute. NEWYORK replies to this queue by explicitly specifying the queue manager name. Another way of doing is to use a temporary dynamic queue. See the *WebSphere MQ Application Programming Guide* for more information.

Converting an existing network into a cluster

If you are converting an existing network into a cluster like this, in step 7, you need to alter the existing queue definition. You also need to delete the remote queue definition at LONDON for the INVENTQ queue. See “Task 7: Converting an existing network into a cluster” on page 97 for an example of this.

Task 2a: Adding a new queue manager to a cluster

Scenario:

- The INVENTORY cluster has been set up as described in "Task 1: Setting up a new cluster" on page 19. 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 will send 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 steps required to complete task 2a

Follow these steps:

1. Determine which full repository PARIS should refer to first

Every queue manager in a cluster must refer to one or other of the full repositories in order to gather information about the cluster and so build up its own partial repository. Choose either of the repositories, because 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(TO.PARIS) CHLTYPE(CLUSRCVR) TRPTYPE(TCP)
CONNAME(PARIS.CHSTORE.COM) CLUSTER(INVENTORY)
DESCR('Cluster-receiver channel for queue manager PARIS')
```

This advertises the queue manager's availability 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 TO.PARIS. These will be 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 TO.LONDON to the queue manager whose network address is LONDON.CHSTORE.COM.

```
DEFINE CHANNEL(TO.LONDON) CHLTYPE(CLUSSDR) TRPTYPE(TCP)
CONNAME(LONDON.CHSTORE.COM) CLUSTER(INVENTORY)
DESCR('Cluster-sender channel from PARIS to repository at LONDON')
```

Adding a queue manager

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 and, 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. See "Establishing communication in a cluster" on page 14 for more information.

The cluster achieved by task 2a

The cluster set up by this task looks like this:

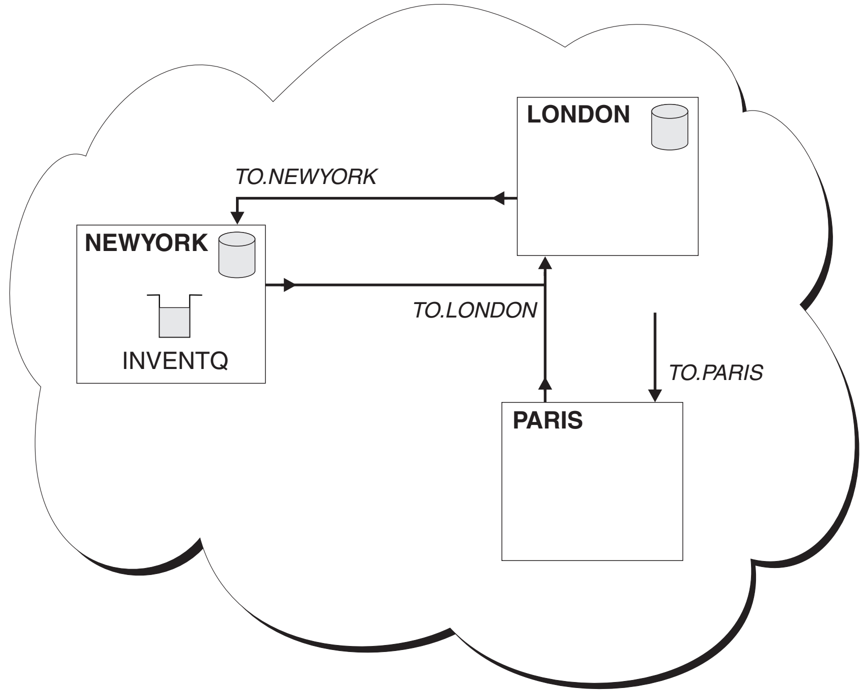

Figure 6. The INVENTORY cluster with three queue managers

By making only two definitions, a CLUSRCVR definition and a CLUSSDR definition, we have added the queue manager PARIS to the cluster.

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 TO.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.

Task 2b: Adding a new queue manager to a cluster — using DHCP

The following example makes use of the following:

- the ability to omit the CONNAME value on a CLUSRCVR definition.
- the ability to use +QMNAME+ on a CLUSSDR definition.

Neither of these are possible on z/OS.

Scenario:

- The INVENTORY cluster has been set up as described in "Task 1: Setting up a new cluster" on page 19. 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 will send 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 may change on system restart.
- The channels between the PARIS and LONDON systems are named according to a defined naming convention, which involves 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, or the name of the queue manager on the LONDON repository is subject to change.

The steps required to complete task 2b

Follow these steps:

1. Determine which full repository PARIS should refer to first

This is the same as for task 2a.

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(TO.PARIS) CHLTYPE(CLUSRCVR)
TRPTYPE(TCP) CLUSTER(INVENTORY)
DESCR('Cluster-receiver channel for queue manager PARIS')
```

This advertises the queue manager's availability to receive messages from other queue managers in the cluster INVENTORY. There is no need to specify the CONNAME, you can request WebSphere MQ to find out the connection name from the system either by omitting CONNAME, or by specifying `CONNAME(' ')`. WebSphere MQ generates the CONNAME value using the current IP address of the system. There is no need to make definitions on other queue managers for a sending end to the cluster-receiver channel TO.PARIS. These will be 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 `TO.+QMNAME+` to the queue manager whose network address is LONDON.CHSTORE.COM.

```
DEFINE CHANNEL(TO.+QMNAME+) CHLTYPE(CLUSSDR) TRPTYPE(TCP)
CONNAME(LONDON.CHSTORE.COM) CLUSTER(INVENTORY)
DESCR('Cluster-sender channel from PARIS to repository at LONDON')
```

The cluster achieved by task 2b

The cluster set up by this task is the same as for task 2a.

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 WebSphere MQ resolves the +QMNAME+ to the queue manager name (LONDON). WebSphere MQ then matches the definition for a channel called TO.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 TO.+QMNAME+ is replaced by an internally-generated CLUSSDR definition for TO.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.

Notes:

1. The channel created with the +QMNAME+ name becomes inactive immediately. It is never used to transmit data.
2. Channel exits may 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 TO.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.

Part 2. Using queue manager clusters

Using clusters

Chapter 4. How queue manager clusters work

This chapter provides more detailed information about clusters and how they work. It discusses:

- "Components of a cluster"
- "What makes clustering work?" on page 38
- "Using aliases and remote-queue definitions with clusters" on page 39

Components of a cluster

Let us now consider how the components of a cluster work together and look at some more of the components and features of WebSphere MQ clusters.

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, and by requesting that thereafter they be 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. When it does this, 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 out any information about itself, for example when it creates a new queue definition, this information is sent to the chosen repository and also 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.

Queues

A queue manager that hosts cluster queues must advertise its queues to the cluster. It does this using the DEFINE QLOCAL command with the CLUSTER option, for example:

```
DEFINE QLOCAL(Q1) CLUSTER(SALES)
```

Once a queue has been advertised, any queue manager in the cluster can put messages to it. To put a message, the queue manager has to 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 its cluster transmission queue.

Cluster transmission queue

Each cluster queue manager has a cluster transmission queue called SYSTEM.CLUSTER.TRANSMIT.QUEUE. A definition for this queue (and others required for clustering) is created by default on every queue manager except on z/OS where it can be defined using the supplied sample CSQ4INSX.

A queue manager that is part of a cluster can send messages on the cluster transmission queue to any other queue manager that is in the same cluster.

Note: Applications must not write directly to the cluster transmission queue. They write to named queues that resolve to the cluster transmission queue. Similarly, you must not specify the cluster transmission queue as a named transmission queue in a remote queue definition, or specify it as the queue manager's default transmission queue.

Queue managers can also communicate with other queue managers that are not part of a cluster. To do this, a queue manager must define channels and a transmission queue to the other queue manager, in the same way as in a distributed-queuing environment.

During name resolution, the cluster transmission queue takes precedence over the default transmission queue. When a queue manager that is not part of a cluster puts a message to a remote queue, the default action, if there is no transmission queue with the same name as the destination queue manager, is to use the default transmission queue. When the sending queue manager **is** part of a cluster, the default action, if there is no transmission queue with the same name as the destination queue manager, is to use SYSTEM.CLUSTER.TRANSMIT.QUEUE, except when the destination queue is not part of the cluster. In short, if the normal resolution takes place, the normal transmission queue is used if the queue is resolved using the full repository, SYSTEM.CLUSTER.TRANSMIT.QUEUE is used.

Cluster channels

The *WebSphere MQ Intercommunication* book describes how message channels are used in distributed queuing. 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.

A cluster-receiver channel definition defines a channel on which a queue manager can receive messages. A queue manager's CLUSRCVR definition enables other queue managers to auto-define their corresponding CLUSSDR channel definitions to that queue manager. First each queue manager must manually define a cluster-sender channel. This 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 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 make the CLUSSDR definitions on full repository queue managers manually and not leave them to be auto-defined.

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) and then 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, you can look at it using the DISPLAY QUEUE command with the CLUSINFO attribute. For example:

```
DISPLAY QUEUE(*) CLUSINFO
```

See the *WebSphere MQ Script (MQSC) Command Reference* book for more information about the DISPLAY QUEUE command.

Auto-definition of channels

When you use distributed queuing, a queue manager must have a definition for a sender channel before it can send a message to a remote destination.

When a queue manager has joined a cluster by making its initial CLUSSDR and CLUSRCVR definitions, it does not need to make any other definitions for channels to other queue managers in the cluster. WebSphere MQ automatically makes cluster-sender channel definitions when they are needed. 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 those specified in 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 those on the corresponding cluster-receiver definition. Always be aware of this, 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 command. To see the auto-defined channels use the command DISPLAY CLUSQMGR(qmname). You can also use the command DISPLAY CHSTATUS(channelname) to display the status of the auto-defined CLUSSDR channel corresponding to the CLUSRCVR channel definition you created. For more information about these commands, refer to the *WebSphere MQ Script (MQSC) Command Reference* book.

You can enable 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, for example, do this in a cluster environment to:

- Tailor communications definitions, that is, SNA LU 6.2 names
- Add or remove other exits, for example, security exits

See the *WebSphere MQ Intercommunication* book for information about the channel auto-definition exit and how to use it.

What makes clustering work?

Defining a cluster-sender channel has the effect of introducing a queue manager to one of the full repository queue managers. The full repository queue manager updates the information in its full repository accordingly. Then it automatically creates a cluster-sender channel back to the queue manager, and sends the queue manager information about the cluster. Thus a queue manager learns about a cluster and a cluster learns about a queue manager.

Look again at Figure 2 on page 5. Suppose that queue manager QM1 wants to send some messages to the queues at QM2. It knows what queues are available at QM2, because QM2 has defined a cluster-sender channel to it and so introduced itself. QM1 has defined a cluster-sender channel to QM2, on which it can send messages.

QM3 has introduced itself to QM2. Because QM1 also holds a full repository, QM2 has passed on all the information about QM3 to QM1. Therefore, QM1 knows what queues are available at QM3, and what cluster-receiver channel QM3 has defined. If QM1 wants to send some messages to queues at QM3, it automatically creates a cluster-sender channel connecting to the cluster-receiver channel at QM3.

Figure 7 on page 39 shows the same cluster, with the two cluster-sender channels that have been created automatically. (These are represented by the two dashed lines that join with the cluster-receiver channel TO.QM3.) It also shows the cluster transmission queue, SYSTEM.CLUSTER.TRANSMIT.QUEUE, which QM1 uses to send its messages. All queue managers in the cluster have a cluster transmission queue, from which they can send messages to any other queue manager in the same cluster.

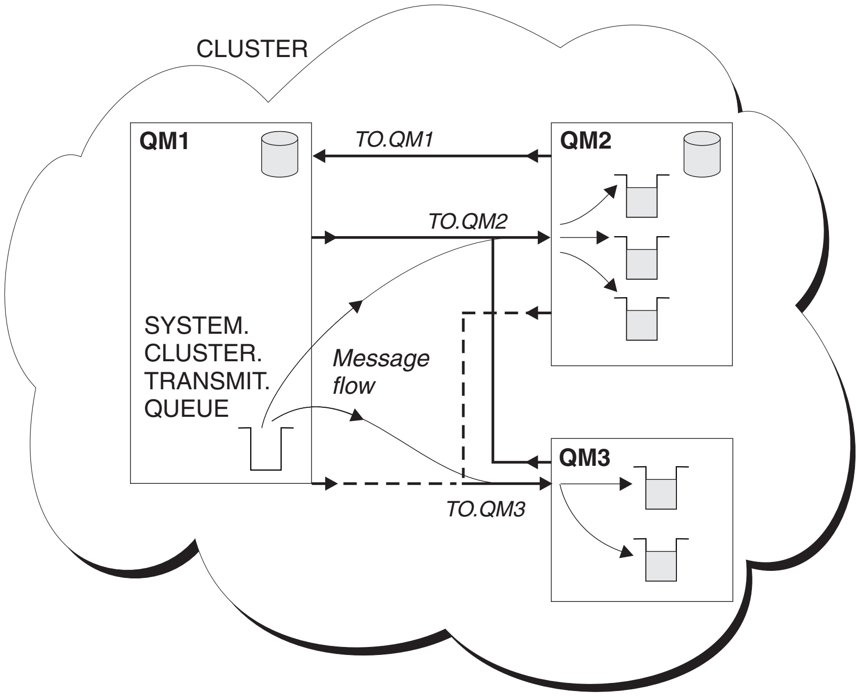

Figure 7. A cluster of queue managers, showing auto-defined channels

The auto-definition of cluster-sender channels—defined automatically when needed—is crucial to the function and efficiency of clusters. However, so that you can see only what **you** need to define to make clusters work, the diagrams throughout this book show only channels (or the receiving ends of channels) for which you make manual definitions.

Using aliases and remote-queue definitions with clusters

There are three types of alias; queue-manager aliases, reply-to queue aliases, and queue aliases. These apply in a clustered environment just as well as in a distributed-queuing environment. This section describes how to use aliases with clusters. When reading this section, you might need to refer to the *WebSphere MQ Application Programming Guide*.

Queue-manager aliases

The concept of queue-manager aliasing is described in detail in the *WebSphere MQ Intercommunication* book.

Queue-manager aliases, which are created using a remote-queue definition with a blank RNAME, have four 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. This can be a cluster queue manager. For example, a queue manager might have the queue-manager alias definition:

```
DEFINE QREMOTE(YORK) RNAME(' ') RQMNAME(CLUSQM)
```

This defines YORK as a queue-manager name that 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, and changes the transmission header to say CLUSQM instead of YORK.

Note: This does not mean that all queue managers in the cluster resolve the name YORK to CLUSQM. 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 the queue manager with the alias definition and the alias is resolved.

Altering or specifying the transmission queue when sending messages

This method of aliasing can be used to join a cluster to a non-cluster system. For example, for queue managers in the cluster ITALY to communicate with the queue manager called PALERMO, which is outside the cluster, one of the queue managers in the cluster must act as a gateway. From this queue manager, issue the command:

```
DEFINE QREMOTE(ROME) RNAME(' ') RQMNAME(PALERMO) XMITQ(X) CLUSTER(ITALY)
```

This is a queue-manager alias definition, which 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. Any message put to a queue opened with the queue-manager name set to ROME in the open handle, is sent to the gateway queue manager, where the queue manager alias definition was made. Once there, it is put on the transmission queue X and moved by conventional, 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 looks in the transmission header to see the name of the destination queue and queue manager. If it has a queue-manager alias definition with the same name as the queue manager referenced, it substitutes the RQMNAME from its 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 a queue manager as a gateway into the cluster.

This enables workload balancing for messages coming from outside the cluster.

Suppose you have a queue called EDINBURGH on more than one queue manager in the cluster, and 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, which is called a *gateway* queue manager. To take advantage of the default workload balancing mechanism, the gateway queue manager **must not** contain an instance of the EDINBURGH queue.

For an example of this see "Putting from a queue manager outside the cluster - alternative" on page 43

Reply-to queue aliases

A reply-to queue alias definition is used to specify alternative names for reply information. Reply-to queue alias definitions can be used in conjunction with clusters just the same as in a distributed queuing environment. For example:

- Queue manager VENICE sends a message to queue manager PISA using the MQPUT call and specifying the following in the message descriptor:
  ```
  ReplyToQ='QUEUE'
  ReplyToQMgr=''
  ```
- So that replies sent to QUEUE can be received on OTHERQ at PISA, create a reply-to queue alias definition on VENICE:
  ```
  DEFINE QREMOTE(QUEUE) RNAME(OTHERQ) RQMNAME(PISA)
  ```

  This form of remote-queue definition creates a reply-to alias. This alias is effective only on the system on which it was created.

RQMNAME and QREMOTE can specify the same names, even if RQMNAME is itself a cluster queue manager.

See the *WebSphere MQ Intercommunication* book for more information.

Queue aliases

A QALIAS definition is used to create an ALIAS by which a queue is to be known. You might do this if, for example, you want to start using a different queue but you do not want to change your applications. You might also do this if for some reason you do not want your applications to know the real name of the queue to which they are putting messages, or because you have a naming convention that differs from the one where the queue is defined. Another reason might be security; your applications might not be authorized to access the queue by its real name but only by its alias.

You create a QALIAS definition on a queue manager using the DEFINE QALIAS command. For example, the command:

```
DEFINE QALIAS(PUBLIC) TARGQ(LOCAL) CLUSTER(C)
```

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 the command:

```
DEFINE QALIAS(PRIVATE) TARGQ(PUBLIC)
```

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.

Examples of using aliases within clusters

Figure 8 and Figure 9 on page 44 show 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)
```

To communicate with a queue manager outside the cluster, one or more queue managers inside the cluster must act as a gateway. The gateway in this example is QM1.

Putting from a queue manager outside a cluster

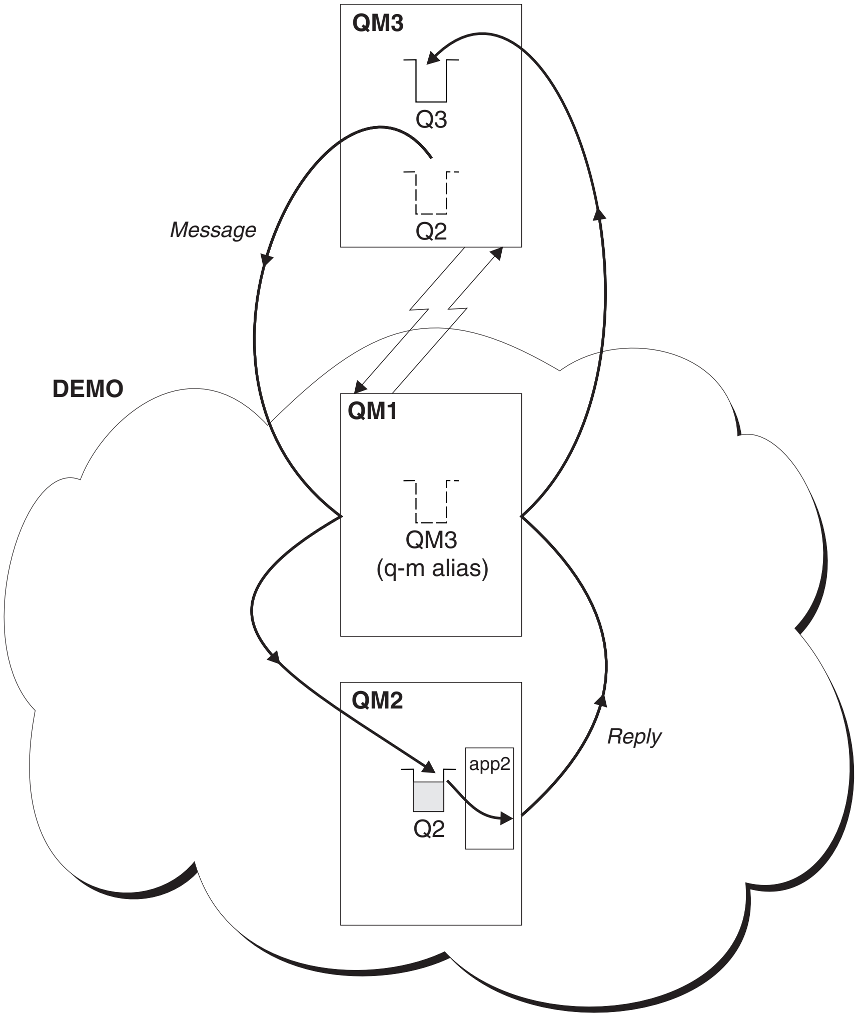

Figure 8. Putting from a queue manager outside the cluster

Let us consider how the queue manager that is outside the cluster can put a message to the queue Q2 at QM2, which is inside the cluster.

The queue manager outside the cluster (QM3 in Figure 8 on page 42) must have a QREMOTE definition for each queue in the cluster that it wants to put messages to. For example:

```
DEFINE QREMOTE(Q2) RNAME(Q2) RQMNAME(QM2) XMITQ(QM1)
```

You can see this remote queue on QM3 in Figure 8 on page 42.

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 8 on page 42.

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.

Replying to a queue manager outside the cluster

To form a return path for replies, the gateway (QM1) advertises a queue-manager alias for the queue manager outside the cluster. It advertises this alias to the whole cluster by adding the cluster attribute to its queue-manager alias definition. (Remember that a queue-manager alias definition is like a remote queue definition, but with a blank RNAME.) For example:

```
DEFINE QREMOTE(QM3) RNAME(' ') RQMNAME(QM3) CLUSTER(DEMO)
```

Again, because QM3 is not part of a cluster, it must communicate using distributed queuing techniques. Therefore, QM1 must also have a sender channel and a transmission queue to QM3. QM3 needs a corresponding receiver channel. The channels and transmission queues are not shown explicitly in Figure 8 on page 42.

When the application (app2) on QM2 issues an MQPUT call to send a reply to Q3 at QM3, the reply is sent to the gateway, which uses its queue-manager alias to resolve the destination-queue and queue-manager name.

Note: You may define more than one route out of a cluster.

Putting from a queue manager outside the cluster - alternative

There is another way of putting from a queue manager outside a cluster.

On the gateway queue manager define a queue-manager alias called, for example, ANY.CLUSTER:

```
DEFINE QREMOTE(ANY.CLUSTER) RNAME(' ') RQMNAME(' ')
```

This maps any response to the queue manager ANY.CLUSTER to null, which means the QREMOTE definition in the queue manager outside the cluster can use the queue manager name ANY.CLUSTER instead of having to use the exact queue manager name. Therefore, on the queue manager outside the cluster, the definition:

```
DEFINE QREMOTE(Q2) RNAME(Q2) RQMNAME(ANY.CLUSTER) XMITQ(QM1)
```

would cause messages to go to QM1 initially, and from there to be routed to any queue manager in the cluster that hosts the cluster queue Q2.

Putting to a queue manager outside the cluster

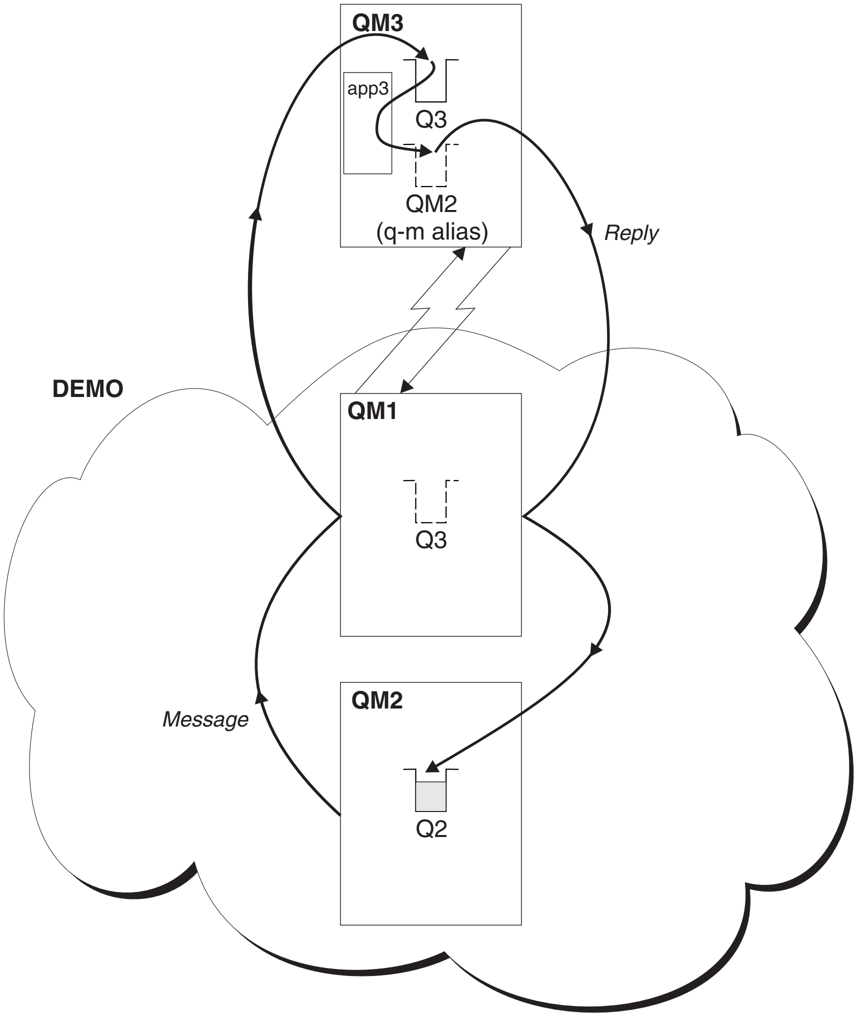

Figure 9. Putting to a queue manager outside the cluster

Now let us consider how to put a message from QM2, which is inside the cluster, to the queue Q3 at QM3, which is outside the cluster.

The gateway, in this example QM1, has a QREMOTE definition 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. These are not shown in Figure 9.

To put a message, an application on QM2 issues an MQPUT call specifying the target queue name (Q3) and specifying the name of the queue to which replies are to be sent (Q2). The message is sent to QM1, which uses its remote-queue definition to resolve the queue name to Q3 at QM3.

Note: You may define more than one route out of a cluster.

Replying from a queue manager outside the cluster

So that QM3 can send replies to the queue managers inside the cluster, it must have a queue-manager alias for each queue manager in the cluster with which it needs to communicate. This queue-manager alias must specify the name of the gateway through which messages are to be routed, that is, the name of the transmission queue to the gateway queue manager. In this example, QM3 needs a queue manager alias definition for QM2:

```
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).

Putting across clusters

Instead of grouping all your queue managers together in one big cluster, you can have many smaller clusters with one or more queue managers in each 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 74.) You can use aliases to change the names of queues and queue managers to avoid name conflicts or to comply with local naming conventions.

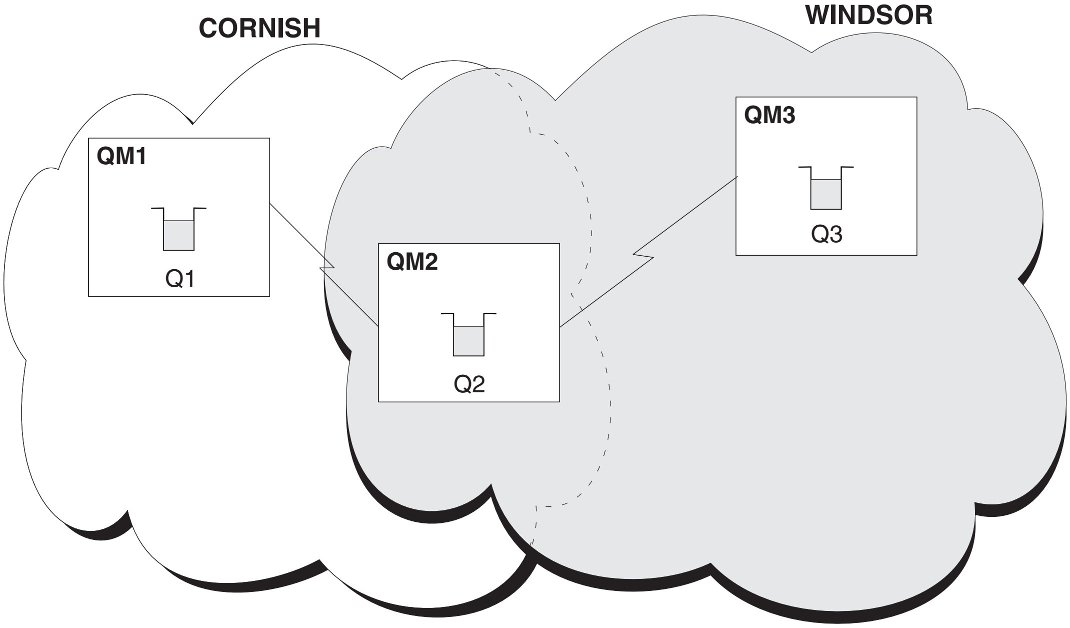

Figure 10. Bridging across clusters

Figure 10 shows two clusters with a bridge between them. (There could be more than one bridge.) QM1 has defined a cluster queue Q1, as follows:

```
DEFINE QLOCAL(Q1) CLUSTER(CORNISH)
```

QM3 has defined a cluster queue Q3, as follows:

```
DEFINE QLOCAL(Q3) CLUSTER(WINDSOR)
```

Aliasing and remote-queue definitions

QM2 has created a namelist called CORNISHWINDSOR, containing the names of both clusters:

```
DEFINE NAMELIST(CORNISHWINDSOR)
 DESCR('CornishWindsor namelist')
 NAMES(CORNISH, WINDSOR)
```

QM2 has also defined a cluster queue Q2, as follows:

```
DEFINE QLOCAL(Q2) CLUSNL(CORNISHWINDSOR)
```

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 10 on page 45, on QM2, you need:

```
DEFINE QALIAS(MYQ3) TARGQ(Q3) CLUSTER(CORNISH) DEFBIND(NOTFIXED)
```

This means that an application connected to a queue manager in CORNISH (for example QM1), can put a message to a queue, which it refers to as MYQ3, and this message is routed to Q3 at QM3.

When you open a queue you need to set DEFBIND to either (NOTFIXED) or (QDEF) because if it is left as the default (OPEN) the queue manager will resolve the alias definition to the bridge queue manager that hosts it, and the bridge will not forward the message on.

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)
```

This means that 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.

Chapter 5. Using clusters for workload management

This chapter describes the advanced method of using WebSphere MQ clusters. It describes how you can set up a cluster that has more than one definition for the same queue, and can therefore benefit from increased availability and workload balancing in your network. This chapter discusses workload management and the implications of using clusters in this way.

More than one instance of a queue

As well as setting up clusters to reduce system administration (as described in Chapter 2, “Using clusters to ease system administration”, on page 11), 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, able to run the same applications and have local definitions of the same queues. For example, in a z/OS parallel sysplex the cloned applications might access and update data in a shared DB2® database or a shared Virtual Storage Access Method (VSAM) database. Because you can have several instances of an application each receiving messages and running independently of each other, the workload can be spread between your queue managers.

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. This means that applications need not explicitly name the queue manager when sending messages. A workload management algorithm determines which queue manager should handle the message.

Figure 11 on page 48 shows a cluster in which there is more than one definition for the queue Q3. When an application at QM1 puts a message to Q3, it does not necessarily know which instance of Q3 will process its message. (Note, however, that when an application on QM2 or an application on QM4 puts a message to Q3 the local instance of the queue is used.)

If a local queue within the cluster becomes unavailable while a message is in transit, the message is forwarded to another instance of the queue (but only if the queue was opened (MQOPEN) with the BIND_NOT_FIXED open option). If there are no other instances of the queue available, the message is placed on the dead letter queue. If a remote queue or an alias queue within the cluster becomes unavailable while a message is in transit, the message is placed on the dead letter queue.

Multiple queue definitions

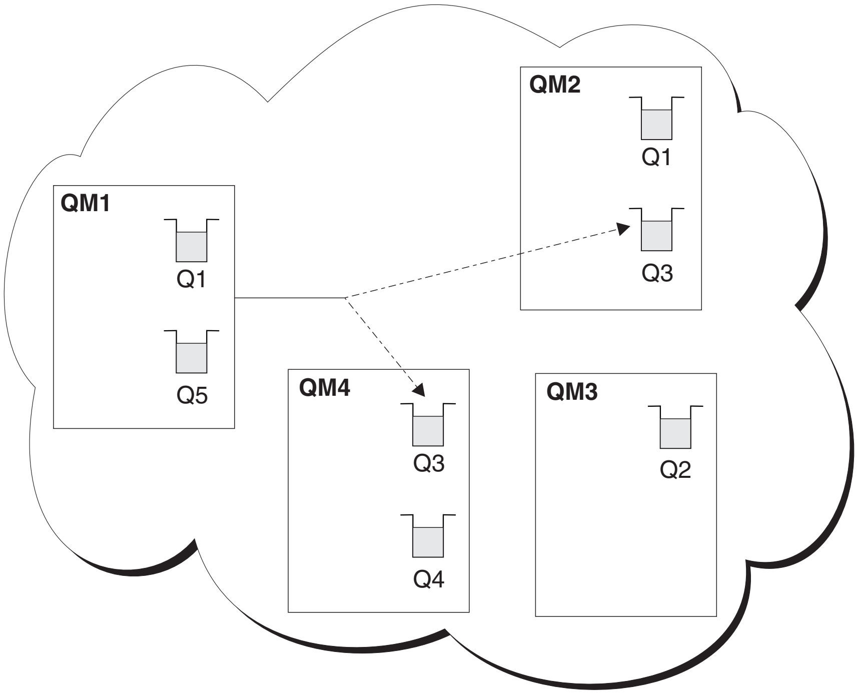

Figure 11. A cluster with multiple instances of the same queue

If the destination queue manager goes out of service while there is still a message on the transmission queue for it, the system attempts to reroute the message. However, 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.

Notes:

1. Before setting up a cluster that has multiple instances of the same queue, ensure that your messages do not have dependencies on each other, for example needing to be processed in a specific sequence or by the same queue manager. See “Programming considerations” on page 53.
2. Make the definitions for different instances of the same queue identical. Otherwise you will get different results from different MQINQ calls.

“Task 3: Adding a new queue manager that hosts a queue” on page 87 shows how to set up a cluster with more than one instance of a queue.

In WebSphere MQ for z/OS, queue managers that are in queue-sharing groups can host cluster queues as shared queues. These cluster queues are therefore available to all other queue managers in the same queue-sharing group. For example, in Figure 11 either or both of the queue managers QM2 and QM4 can be a shared-queue manager. They can be in the same or a different queue-sharing group. Each has a definition for the queue Q3. If a message is put to the shared queue Q3, on the shared-queue manager QM4, any of the queue managers in the same queue-sharing group can read the message. Because each queue-sharing group can contain up to 32 queue managers, each with access to the same data, this significantly increases the throughput of your messages.

Workload balancing

When you have clusters containing more than one instance of the same queue, WebSphere MQ uses a workload management algorithm to determine the best queue manager to route a message to. The workload management algorithm selects the local queue manager as the destination whenever possible. If there is no instance of the queue on the local queue manager, the algorithm determines which destinations are suitable. Suitability is based on the state of the channel (including any priority you might have assigned to the channel), and also the availability of the queue manager and queue. The algorithm uses a round-robin approach to finalize its choice between the suitable queue managers.

Cluster workload user exit

In most cases the workload management algorithm will be sufficient for your needs. However, so that you can provide your own user-exit program to tailor workload management, WebSphere MQ includes a user exit, the cluster workload exit.

If you have some particular information about your network or messages that you could use to influence workload balancing, you might decide to write a cluster workload exit program or use one supplied by a third party. For example, you might know which are the high-capacity channels or the cheap network routes, or you might decide that you want to route messages depending upon their content.

The cluster workload exit is called when a cluster queue is opened using the MQOPEN or MQPUT1 call, and when a message is put to a queue using the MQPUT call. The target queue manager selected at MQOPEN time is fixed if MQOO_BIND_ON_OPEN is specified (see “MQOPEN” on page 55). In this case, even if the target queue manager fails, the exit is not run again. In cases where the target queue manager is not fixed, if the target queue manager chosen at the time of an MQPUT call is unavailable, or fails while the message is still on the transmission queue, the exit is called again to select a new target queue manager.

You name cluster workload exits in the queue-manager definition by specifying the CLWLEXIT attribute on the ALTER QMGR command. For example:

```
ALTER QMGR CLWLEXIT(myexit)
```

On platforms other than z/OS, the queue manager will load the new CLWLEXIT the next time the queue manager is started.

If the queue-manager definition does not contain a workload-exit program name, the workload exit is not called.

Cluster workload exits are called with an exit parameter structure (MQWXP), a message definition structure (MQMD), a message length parameter, and a copy of the message (or part of the message). See Chapter 10, “Cluster workload exit call and data structures”, on page 121 for reference information about the cluster workload exit and the associated data structures.

For more information on using your own cluster workload exit see Chapter 11, “Using your own cluster workload exits”, on page 145

Writing and compiling cluster workload exit programs

The following guidance applies specifically to cluster workload exit programs. Read it in conjunction with the general application-programming guidance given in the *WebSphere MQ Application Programming Guide*.

WebSphere MQ for z/OS

Cluster workload exits are invoked as if by a z/OS LINK in:

- Non-authorized problem program state
- Primary address space control mode
- Non-cross-memory mode
- Non-access register mode
- 31-bit addressing mode
- Storage key 8
- Program Key Mask 8
- TCB key 8

Put the link-edited modules in the data set specified by the CSQXLIB DD statement of the queue manager address space procedure. The names of the load modules are specified as the workload exit names in the queue-manager definition.

When writing workload exits for WebSphere MQ for z/OS, the following rules apply:

- You must write exits must be written in assembler or C. If you use C, it must conform to the C systems programming environment for system exits, described in the *z/OS C/C++ Programming Guide*, SC09-2362.
- Exits are loaded from the non-authorized libraries defined by a CSQXLIB DD statement. Providing CSQXLIB has DISP=SHR, exits can be updated while the queue manager is running, with the new version used when the next MQCONN thread the queue manager starts.
- Exits must be reentrant, and capable of running anywhere in virtual storage.
- Exits must reset the environment on return to that at entry.
- Exits must free any storage obtained, or ensure that it will be freed by a subsequent exit invocation.
- No MQI calls are allowed.
- Exits must not use any system services that could cause a wait, because this would severely degrade the performance of the queue manager. In general, therefore, avoid SVCs, PCs, and I/O.
- Exits must not issue ESTAEs or SPIEs, apart from within any subtasks they attach.

Note that there are no absolute restrictions on what you can do in an exit. However, most SVCs involve waits, so avoid them, except for the following:

- GETMAIN/FREEMAIN
- LOAD/DELETE

Do not use ESTAEs and ESPIEs because their error handling might interfere with the error handling performed by WebSphere MQ. This means that WebSphere MQ might not be able to recover from an error, or that your exit program might not receive all the error information.

The system parameter EXITLIM, which is described in the *WebSphere MQ for z/OS System Setup Guide*, limits the amount of time an exit may run for. The default value for EXITLIM is 30 seconds. If you see the return code

MQRC_CLUSTER_EXIT_ERROR (2266 X'8DA') your exit may be looping. If you think the exit needs more than 30 seconds to complete, increase the value of EXITLIM.

MQSeries for Compaq OpenVMS Alpha

When linking a workload exit on Compaq OpenVMS Alpha, specify the following in the linker options file:

```
sys$share:mqm/share
sys$share:mqutl/share
SYMBOL_VECTOR=(clwlFunction=PROCEDURE,MQStart=PROCEDURE)
```

You need a system-wide executive logical name to reference the exit image. For example, if the exit name is SYS$SHARE:AMQSWLM.EXE, define the following logical name:

```
$DEFINE/SYSTEM/EXEC AMQSWLM SYS$SHARE:AMQSWLM
```

Do not specify the .EXE file extension in the logical name definition.

For this logical name to be defined during system startup, define it in SYS$MANAGER:MQS_SYSTARTUP.COM.

Platforms other than z/OS

Cluster workload exits must not use MQI calls. In other respects, the rules for writing and compiling cluster workload exit programs are similar to the rules that apply to channel exit programs. These are described in detail in the *WebSphere MQ Intercommunication* book.

For information about building your application, see the *WebSphere MQ Application Programming Guide* and the *WebSphere MQ Intercommunication* book. However you need to resolve the MQXCLWLN function by linking with the mqutl library on Windows NT or the libmqutl library on the UNIX platforms.

Sample cluster workload exit

WebSphere MQ includes a sample cluster workload exit program. You can copy this sample and use it as a basis for your own programs.

WebSphere MQ for z/OS

The sample cluster workload exit program is supplied in Assembler and in C. The Assembler version is called CSQ4BAF1 and can be found in the library `thlqual`.SCSQASMS. The C version is called CSQ4BCF1 and can be found in the library `thlqual`.SCSQC37S. (`thlqual` is the target library high-level qualifier for WebSphere MQ data sets in your installation.)

On platforms other than z/OS

The sample cluster workload exit program is supplied in C and is called amqswlm0.c. It can be found in:

Table 3. Sample cluster workload exit program location (not z/OS)

AIX	/usr/mqm/samp
Compaq Tru64 UNIX, HP-UX, and Sun Solaris	/opt/mqm/samp
OS/2 Warp	C:\mqm\tools\c\samples

Table 3. Sample cluster workload exit program location (not z/OS) (continued)

Windows	C:\Program Files\Websphere MQ\Tools\c\Samples (Where C is the drive on which you have installed the product.)
iSeries	The qmqm library
Compaq OpenVMS Alpha	sys&sysroot:[syshlp.examples.mqseries.bin]
Compaq NonStop Kernel	$volume.ZMQSSMPL (Where $volume is the drive on which you have installed MQSeries.)

This sample exit routes all messages to a particular queue manager, unless that queue manager becomes unavailable. It reacts to the failure of the queue manager by routing messages to another queue manager.

You indicate which queue manager you want messages to be sent to by supplying the name of its cluster-receiver channel in the CLWLDATA attribute on the queue-manager definition. For example:

```
ALTER QMGR CLWLDATA('TO.myqmgr')
```

To enable the exit, supply its full path and name in the CLWLEXIT attribute:

On UNIX systems:

```
ALTER QMGR CLWLEXIT('path/amqswlm(clwlFunction)')
```

On OS/2 Warp and Windows NT:

```
ALTER QMGR CLWLEXIT('path\amqswlm(clwlFunction)')
```

On z/OS:

```
ALTER QMGR CLWLEXIT(CSQ4BxF1)
```

where `x` is either ‘A’ or ‘C’, depending on the programming language of the version you are using.

On OS/400®:

Enter the MQSC command:

```
ALTER QMGR CLWLEXIT('AMQSWLM   library   ')
```

(both the program name and the library name occupy 10 characters and are blank-padded to the right if necessary). Alternatively, use the CL command:

```
CHGMQM MQMNAME(qmgrname) CLWLEXIT('library/AMQSWLM')
```

Now, instead of using the supplied workload management algorithm, WebSphere MQ calls this exit to route all messages to your chosen queue manager.

On Compaq OpenVMS Alpha:

```
ALTER QMGR CLWLEXIT('sys$share:AMQSWLM(clwlFunction)')
```

On Compaq NonStop Kernel:

```
ALTER QMGR CLWLEXIT('$volume.subvol.amqswlm(clwlFunction)')
```

Programming considerations

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 do not 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 as described in "Reviewing applications for message affinities".

Also, be aware of the following:

1. If an application opens a target queue so that it can read messages from it or set its attributes, the MQOPEN call operates only on the local version of the queue.
2. If an application opens a target queue so that it can write messages to it, the MQOPEN call chooses between all available instances of the queue. Any local version of the queue is chosen in preference to other instances. This might limit the ability of your applications to exploit clustering.

Reviewing applications for message affinities

Before starting to use clusters with multiple definitions of the same queue, examine your applications to see whether there are any that have message affinities, that is, they exchange related messages. With clusters, a message can be routed to *any* queue manager that hosts a copy of the correct queue, affecting the logic of applications with 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. It might be important that all the questions are sent to the same queue manager and that all the answers are sent back to the other queue manager. In this situation, it is important that the workload management routine does not send the messages to any queue manager that just happens to host a copy of the correct queue.

Similarly, you might have applications that require messages to be processed in sequence, for example a file transfer application or database replication application that sends batches of messages that must be retrieved in sequence.

Note: The use of segmented messages can also cause an affinity problem.

Handling message affinities

If you have applications with message affinities, try, to remove the affinities before starting to use clusters.

Removing message affinities improves the availability of applications. If an application that has message affinities sends a batch of messages to a queue manager and the queue manager fails after receiving only part of the batch, the sending queue manager must wait for it to recover 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 may 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 other possible solutions to the problem. For example, you can:

Name a specific destination on the MQOPEN call

One solution is to specify the remote-queue name and the queue manager name on each MQOPEN call. If you do this, all messages put to the queue using that object handle go to the same queue manager, which might be the local queue manager.

The disadvantages to this technique are:

- No workload management is carried out. This prevents you from taking advantage of the benefits of cluster workload management.
- 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

A variation on the first solution is to 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 this queue manager name and specify it on all subsequent messages.

The advantage of this method over the previous one is that some workload balancing is carried out to deliver the first message.

The disadvantage of this method is that the first queue manager must wait for a response to its first message and must be prepared to find and use the ReplyToQMgr information before sending subsequent messages. As with the previous method, if there is more than one route to the queue manager, the sequence of the messages might not be preserved.

Use the MQOO_BIND_ON_OPEN option on the MQOPEN call

Another solution is to force all your messages to be put to the same destination, do this using the MQOO_BIND_ON_OPEN option on the MQOPEN call. 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 will be selected when the queue is opened and this selection will be 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 MQOO_BIND option on the MQOPEN call to default to MQOO_BIND_AS_Q_DEF. See “Queue definition commands” on page 64 for more information about using the DEFBIND attribute in queue definitions.

Use the cluster workload exit

Instead of modifying your applications you can circumvent the message affinities problem by writing a cluster workload exit program. This would not be easy and is not a recommended solution. This 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.

MQI and clusters

The following WebSphere MQ application programming interface (API) calls have options to help use clusters:
- MQOPEN
- MQPUT and MQPUT1
- MQINQ
- MQSET

The options on these calls that relate specifically to clustering are described here. This information should be read in conjunction with the *WebSphere MQ Application Programming Reference* book, which describes each call in full.

The remainder of this chapter contains general-use programming interface information.

MQOPEN

An option on the MQOPEN call, the MQOO_BIND_ON_OPEN option, allows you to specify that, when there are multiple instances of the same queue within a cluster, the target queue manager needs to be fixed. That is, all messages put to the queue specifying the object handle returned from the MQOPEN call must be directed to the **same** queue manager using the same route.

Use this option when you have messages with affinities. For example, if you need a batch of messages all to be processed by the same queue manager, specify MQOO_BIND_ON_OPEN when you open the queue. WebSphere MQ fixes the queue manager and the route to be taken by all messages put to that queue.

If you do not want to force all your messages to be written to the same destination, specify MQOO_BIND_NOT_FIXED on the MQOPEN call. This selects a destination at MQPUT time, that is, on a message-by-message basis.

Note: Do not specify MQOO_BIND_NOT_FIXED and MQMF_SEGMENTATION_ALLOWED at the same time. If you do, the segments of your message might all be delivered to different queue managers, scattered throughout the cluster.

If you do not specify either MQOO_BIND_ON_OPEN or MQOO_BIND_NOT_FIXED, the default option is MQOO_BIND_AS_Q_DEF. Using MQOO_BIND_AS_Q_DEF causes the binding that is used for the queue handle to be taken from the DefBind queue attribute. See “Queue definition commands” on page 64.

You can also choose a destination queue manager, by specifying its name in the object descriptor on the MQOPEN call. In this way, you can select any queue manager, including the local one.

If you specify one or more of the options MQOO_BROWSE, MQOO_INPUT_*, or MQOO_SET on the MQOPEN call, there must be a local instance of the cluster queue for the open to succeed. If you specify one or more of the options MQOO_OUTPUT, MQOO_BIND_*, or MQOO_INQUIRE on the MQOPEN call, and none of the options MQOO_BROWSE, MQOO_INPUT_*, or MQOO_SET (which always select the local instance) the instance opened is either:

- The instance on the local queue manager, if there is one, or
- An instance elsewhere in the cluster, if there is no local queue-manager instance

For full details about the MQOPEN call and how to use it, see the *WebSphere MQ Application Programming Reference* book.

Resolved queue manager name

When a queue manager name is resolved at MQOPEN time, the resolved name is returned to the application. If the application tries to use this name on a subsequent MQOPEN call, it might find that is it not authorized to access the name.

MQPUT and MQPUT1

If MQOO_BIND_NOT_FIXED is specified on an MQOPEN call, each subsequent MQPUT call invokes the workload management routine to determine which queue manager to send the message to. The destination and route to be taken are selected on a message-by-message basis. The destination and route might change after the message has been put if conditions in the network change. The MQPUT1 call always operates as though MQOO_BIND_NOT_FIXED were in effect, that is, it always invokes the workload management routine.

When the workload management routine has selected a queue manager, the local queue manager completes the put operation. If the target queue manager is a member of the same cluster as the local queue manager, the local queue manager puts the message on the cluster transmission queue, SYSTEM.CLUSTER.TRANSMIT.QUEUE, for transmission to the destination. If the target queue manager is outside the cluster, and the local queue manager has a transmission queue with the same name as the target queue manager, it puts the message on that transmission queue.

If MQOO_BIND_ON_OPEN is specified on the MQOPEN call, MQPUT calls do not need to invoke the workload management routine because the destination and route have already been selected.

MQINQ

Before you can inquire on a queue, you must open it using the MQOPEN call and specifying MQOO_INQUIRE.

If you have clusters in which there are multiple instances of the same queue, the attributes that can be inquired depend on whether there is a local instance of the cluster queue, and on how the queue is opened.

If, on the MQOPEN call, in addition to specifying MQOO_INQUIRE, you also specify one of the options MQOO_BROWSE, MQOO_INPUT_*, or MQOO_SET, there must be a local instance of the cluster queue for the open to succeed. In this case you can inquire on all the attributes that are valid for local queues.

If, on the MQOPEN call, you specify only MQOO_INQUIRE, or MQOO_INQUIRE and MQOO_OUTPUT (but specify none of the options MQOO_BROWSE, MQOO_INPUT_*, or MQOO_SET, which always cause the local instance of a cluster queue to be selected), the instance opened is either:
- The instance on the local queue manager, if there is one, or
- An instance elsewhere in the cluster, if there is no local queue-manager instance

MQI and clusters

If the queue that is opened is not a local queue, only the attributes listed below can be inquired. The QType attribute has the value MQQT_CLUSTER in this case.
- DefBind
- DefPersistence
- DefPriority
- InhibitPut
- QDesc
- QName
- QType

To inquire on the DefBind attribute of a cluster queue, use the MQINQ call with the selector MQIA_DEF_BIND. The value returned is either MQBND_BIND_ON_OPEN or MQBND_BIND_NOT_FIXED. To inquire on the CLUSTER and CLUSNL attributes of the local instance of a queue, use the MQINQ call with the selector MQCA_CLUSTER_NAME or the selector MQCA_CLUSTER_NAMELIST respectively.

Note: If you open a cluster queue with no fixed binding (that is, specifying MQOO_BIND_NOT_FIXED on the MQOPEN call, or specifying MQOO_BIND_AS_Q_DEF when the DefBind attribute of the queue has the value MQBND_BIND_NOT_FIXED), successive MQINQ calls might inquire different instances of the cluster queue.

MQSET

If you open a cluster queue to set its attributes (specifying the MQOO_SET option), there must be a local instance of the cluster queue for the open to succeed. You cannot use the MQSET call to set the attributes of a queue elsewhere in the cluster. However, if you open an alias queue or a remote queue defined with the cluster attribute, you can use the MQSET call to set attributes of the alias queue or remote queue even if the target queue or remote queue it resolves to is a cluster queue.

Return codes

These are the return codes specific to clusters.

MQRC_CLUSTER_EXIT_ERROR (2266 X'8DA')

Occurs when an MQOPEN, MQPUT, or MQPUT1 call is issued to open a cluster queue or put a message on it, and the cluster workload exit defined by the queue-manager’s ClusterWorkloadExit attribute fails unexpectedly or does not respond in time.

On WebSphere MQ for z/OS a message is written to the system log giving more information about this error.

Subsequent MQOPEN, MQPUT, and MQPUT1 calls for this queue handle are processed as though the ClusterWorkloadExit attribute were blank.

MQRC_CLUSTER_EXIT_LOAD_ERROR (2267 X'8DB')

On z/OS, if the cluster workload exit cannot be loaded, a message is written to the system log and processing continues as though the ClusterWorkloadExit attribute is blank.On platforms other than z/OS, when an MQCONN or MQCONNX call is issued to connect to a queue manager, but the call fails because the cluster workload exit defined by the queue-manager's ClusterWorkloadExit attribute cannot be loaded.

MQRC_CLUSTER_PUT_INHIBITED (2268 X'8DC')

Occurs when an MQOPEN call with the MQOO_OUTPUT and MQOO_BIND_ON_OPEN options in effect is issued for a cluster queue, but all the instances of the queue in the cluster are currently put-inhibited, that is, all the queue instances have the InhibitPut attribute set to MQQA_PUT_INHIBITED. Because there are no queue instances available to receive messages, the MQOPEN call fails.

This reason code occurs only when both of the following are true:

- There is no local instance of the queue. (If there is a local instance, the MQOPEN call succeeds, even if the local instance is put-inhibited.)
- There is no cluster workload exit for the queue, or there is a cluster workload exit but it does not choose a queue instance. (If the cluster workload exit chooses a queue instance, the MQOPEN call succeeds, even if that instance is put-inhibited.)

If the MQOO_BIND_NOT_FIXED option is specified on the MQOPEN call, the call can succeed even if all the queues in the cluster are put-inhibited. However, a subsequent MQPUT call may fail if all the queues are still put-inhibited at the time of that call.

MQRC_CLUSTER_RESOLUTION_ERROR (2189 X'88D')

Occurs when an MQOPEN, MQPUT, or MQPUT1 call is issued to open a cluster queue or put a message on it, and the queue definition cannot be resolved correctly because a response is required from the full repository queue manager but none is available.

MQRC_CLUSTER_RESOURCE_ERROR (2269 X'8DD')

Occurs when an MQOPEN, MQPUT, or MQPUT1 call is issued for a cluster queue, but an error occurs while trying to use a resource required for clustering.

MQRC_NO_DESTINATIONS_AVAILABLE (2270 X'8DE')

Occurs when an MQPUT or MQPUT1 call is issued to put a message on a cluster queue, but at the time of the call there are no longer any instances of the queue in the cluster. The PUT fails and the message is not sent.

This situation can occur when MQOO_BIND_NOT_FIXED is specified on the MQOPEN call that opens the queue, or MQPUT1 is used to put the message.

MQRC_STOPPED_BY_CLUSTER_EXIT (2188 X'88C')

Occurs when an MQOPEN, MQPUT, or MQPUT1 call is issued to open or put a message on a cluster queue, but the cluster workload exit rejects the call.

Return codes

Chapter 6. Using WebSphere MQ commands with clusters

This chapter gives an overview of all the WebSphere MQ Script commands (MQSC), attributes, and parameters that apply to the use of clusters. Read this information in conjunction with the *WebSphere MQ Script (MQSC) Command Reference* book, which provides details about all the MQSC commands, their syntax, attributes, and parameters.

Note that the attributes used in commands shown in the *WebSphere MQ Script (MQSC) Command Reference* book differ from the full-length attribute names shown in the *WebSphere MQ Application Programming Reference* book. See Table 4 and Table 5 on page 121 for some examples.

For each MQSC command described here, on platforms other than z/OS, there is an equivalent Programmable Command Format (PCF) command. For details about PCFs, refer to the *WebSphere MQ Programmable Command Formats and Administration Interface* book.

Throughout this book, MQSC commands are shown as they would be entered by the system administrator at the command console. Remember that you do not have to issue the commands in this way. There are a number of other methods, depending on your platform. For example:

- On WebSphere MQ for iSeries you can use CL commands or you can store MQSC commands in a file and use the STRMQMMQSC CL command. See the *WebSphere MQ for iSeries V5.3 System Administration Guide* for more information.
- On z/OS you can use the COMMAND function of the CSQUTIL utility, the operations and control panels or you can use the z/OS console. These are described in the *WebSphere MQ for z/OS System Administration Guide*.
- On all other platforms you can store the commands in a file and use runmqsc, as described in the *WebSphere MQ System Administration Guide* .

For a complete description of the different methods of issuing MQSC commands, refer to the *WebSphere MQ Script (MQSC) Command Reference* book.

The WebSphere MQ Explorer cannot administer a cluster with repository queue managers on WebSphere MQ for z/OS. You must nominate an additional repository on a system that the WebSphere MQ Explorer can administer.

On WebSphere MQ for Windows you can also use WebSphere MQ Explorer to work with clusters. For example you can view cluster queues and inquire about the status of cluster-sender and cluster-receiver channels. WebSphere MQ Explorer includes three wizards, which you can use to guide you through the following tasks:

- Creating a new cluster
- Joining an independent queue manager to a cluster
- Joining a cluster queue manager to another cluster

See the *WebSphere MQ System Administration Guide* for more information about using WebSphere MQ Explorer.

MQSC command attributes

A number of MQSC commands have cluster attributes. These are introduced here, under the following headings:
- Queue-manager definition commands
- Channel definition commands
- Queue definition commands

In the commands, a cluster name, specified using the CLUSTER attribute, can be up to 48 characters long. Cluster names must conform to the rules described in the *WebSphere MQ Script (MQSC) Command Reference* book.

A list of cluster names, specified using the CLUSNL attribute, can contain up to 256 names. To create a cluster namelist, use the command DEFINE NAMELIST described in the *WebSphere MQ Script (MQSC) Command Reference* book.

Queue-manager definition commands

To specify that a queue manager holds a full repository for a cluster, use the ALTER QMGR command specifying the attribute REPOS(*clustername*). To specify a list of several cluster names, define a cluster namelist and then use the attribute REPOSNL(*namelist*) on the ALTER QMGR command:

```
DEFINE NAMELIST(CLUSTERLIST)
       DESCR('List of clusters whose repositories I host')
       NAMES(CLUS1, CLUS2, CLUS3)

ALTER QMGR REPOSNL(CLUSTERLIST)
```

Use the CLWLEXIT(*name*) attribute to specify the name of a user exit to be called when a message is put to a cluster queue. Use the CLWLDATA(*data*) attribute to specify data to be passed to the cluster workload user exit. Use the CLWLLEN(*length*) attribute to specify the maximum amount of message data to be passed to the cluster workload user exit.

The attributes on the ALTER QMGR command also apply to the DISPLAY QMGR command.

For full details of the attributes and syntax of the ALTER QMGR command and the DISPLAY QMGR command, refer to the *WebSphere MQ Script (MQSC) Command Reference* book.

The equivalent PCFs are MQCMD_CHANGE_Q_MGR and MQCMD_INQUIRE_Q_MGR. These are described in the *WebSphere MQ Programmable Command Formats and Administration Interface* book.

Channel definition commands

The DEFINE CHANNEL, ALTER CHANNEL, and DISPLAY CHANNEL commands have two specific CHLTYPE parameters for clusters: CLUSRCVR and CLUSSDR. To define a cluster-receiver channel you use the DEFINE CHANNEL command, specifying CHLTYPE(CLUSRCVR). Many of the other attributes needed on a cluster-receiver channel definition are the same as those that apply to a receiver-channel or a sender-channel definition. To define a cluster-sender channel you use the DEFINE CHANNEL command, specifying CHLTYPE(CLUSSDR), and many of the same attributes as you use to define a sender channel.

It is no longer necessary to specify the full repository queue manager's name when you define a cluster-sender channel. So long as you know the naming convention used for channels in your cluster you can make a CLUSSDR definition using the +QMNAME+ construction (+QMNAME+ is not supported on z/OS). After connection WebSphere MQ changes the name of the channel and substitutes the correct full repository queue manager name in place of +QMNAME+. The resulting channel name is truncated to 20 characters.

Clearly this works only if your convention for naming channels includes the name of the queue manager. For example, if you have a full repository queue manager called QM1 with a cluster-receiver channel called TO.QM1.ALPHA, another queue manager can define a cluster-sender channel to this queue manager, but specify the channel name as TO.+QMNAME+.ALPHA.

If you use the same naming convention for all your channels, be aware that only one +QMNAME+ definition can exist at one time.

The attributes on the DEFINE CHANNEL and ALTER CHANNEL commands that are specific to clusters are:
- CLUSTER
- CLUSNL
- NETPRTY

The CLUSTER and CLUSNL attributes apply only if you specify CHLTYPE(CLUSRCVR) or CHLTYPE(CLUSSDR). The NETPRTY attribute applies only to cluster-receiver channels.

Use the CLUSTER attribute to specify the name of the cluster with which this channel is associated. Alternatively, use the CLUSNL attribute to specify a namelist of cluster names.

Use the NETPRTY attribute to specify a priority for the channel. This helps the workload management routines. If there is more than one possible route to a destination, the workload management routine selects the one with the highest priority.

These attributes on the DEFINE CHANNEL command and ALTER CHANNEL command also apply to the DISPLAY CHANNEL command.

The CONNAME specified on a cluster-receiver channel definition is used throughout the cluster to identify the network address of the queue manager. Take care to select a value for the CONNAME parameter that resolves throughout your WebSphere MQ cluster. Do not use a generic name. Remember that the value specified on the cluster-receiver channel takes precedence over any value specified in a corresponding cluster-sender channel.

The following paragraphs discuss the facility to omit the CONNAME value on a CLUSRCVR definition. This is not possible on z/OS.

When the network protocol you are using is TCP/IP it is not necessary to specify the network address of your queue manager when you define a cluster-receiver channel. You can issue the DEFINE CHANNEL command without supplying a value for CONNAME. WebSphere MQ generates a CONNAME for you, assuming the default port and using the current IP address of the system. The generated CONNAME is always in the dotted-decimal form, rather than in the form of an alphanumeric DNS host name.

Command attributes

This facility is useful when you have machines using Dynamic Host Configuration Protocol (DHCP). With WebSphere MQ if you do not supply a value for the CONNAME on a CLUSRCVR channel, you do not need to make any changes to your definitions if DHCP allocates you a new IP address.

If you specify a blank for the CONNAME on the CLUSRCVR definition, WebSphere MQ generates a CONNAME from the IP address of the system. Only the generated CONNAME is stored in the repositories. Other queue managers in the cluster do not know that the CONNAME was originally blank.

If you issue the DISPLAY CLUSQMGR command you will see the generated CONNAME. However, if you issue the DISPLAY CHANNEL command from the local queue manager, you will see that the CONNAME is blank.

If the queue manager is stopped and restarted with a different IP address, because of DHCP, WebSphere MQ regenerates the CONNAME and updates the repositories accordingly.

Note: Auto-defined cluster-sender channels take their attributes from those specified in 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 those on the corresponding cluster-receiver definition. Beware that you can, for example, define a CLUSRCVR without specifying a port number in the CONNAME parameter, whilst manually defining 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 would be used and the channel would fail.

Note that the DISPLAY CHANNEL command does not display auto-defined channels. However, you can use the DISPLAY CLUSQMGR command, introduced in “DISPLAY CLUSQMGR” on page 66, to examine the attributes of auto-defined cluster-sender channels.

Use the DISPLAY CHSTATUS command to display the status of a cluster-sender or cluster-receiver channel. This command gives the status of both manually defined channels and auto-defined channels.

For full details of the attributes and syntax of the DEFINE CHANNEL, ALTER CHANNEL, DISPLAY CHANNEL, and DISPLAY CHSTATUS commands, refer to the *WebSphere MQ Script (MQSC) Command Reference* book.

The equivalent PCFs are MQCMD_CHANGE_CHANNEL, MQCMD_COPY_CHANNEL, MQCMD_CREATE_CHANNEL, and MQCMD_INQUIRE_CHANNEL. For information about these PCFs, refer to the *WebSphere MQ Programmable Command Formats and Administration Interface* book.

Queue definition commands

The cluster attributes on the DEFINE QLOCAL, DEFINE QREMOTE, and DEFINE QALIAS commands, and the three equivalent ALTER commands, are:

CLUSTER
To specify the name of the cluster to which the queue belongs.

CLUSNL
To specify a namelist of cluster names.

DEFBIND
To specify the binding to be used when an application specifies MQOO_BIND_AS_Q_DEF on the OPEN call. The default for this attribute is DEFBIND(OPEN), which binds the queue handle to a specific instance of the cluster queue when the queue is opened. The alternative is to specify DEFBIND(NOTFIXED) so that the queue handle is not bound to any particular instance of the cluster queue. When you specify DEFBIND on a queue definition, the queue is defined with one of the attributes, MQBND_BIND_ON_OPEN or MQBND_BIND_NOT_FIXED.

We recommend that you set the DEFBIND attribute to the same value on all instances of the same cluster queue.

The attributes on the DEFINE QLOCAL, DEFINE QREMOTE, and DEFINE QALIAS commands also apply to the DISPLAY QUEUE command. To display information about cluster queues, specify a queue type of QCLUSTER or the keyword CLUSINFO on the DISPLAY QUEUE command, or use the command DISPLAY QCLUSTER.

The DISPLAY QUEUE or DISPLAY QCLUSTER command returns the name of the queue manager that hosts the queue (or the names of all queue managers if there is more than one instance of the queue). It also returns the system name for each queue manager that hosts the queue, the queue type represented, and the date and time at which the definition became available to the local queue manager. This information is returned using the CLUSQMGR, QMID, CLUSQT, CLUSDATE, and CLUSTIME attributes respectively.

The system name for the queue manager (QMID), is a unique, system-generated name for the queue manager.

You can define a cluster queue that is also a shared queue. For example you can define:

```
DEFINE QLOCAL(MYQUEUE) CLUSTER(MYCLUSTER) QSGDISP(SHARED) CFSTRUCT(STRUCTURE)
```

For full details of the parameters and syntax of the QUEUE definition commands, refer to the *WebSphere MQ Script (MQSC) Command Reference* book.

The equivalent PCFs are MQCMD_CHANGE_Q, MQCMD_COPY_Q, MQCMD_CREATE_Q, and MQCMD_INQUIRE_Q. For information about these PCFs, refer to the *WebSphere MQ Programmable Command Formats and Administration Interface* book.

WebSphere MQ commands for work with clusters

This section introduces MQSC commands that apply specifically to work with WebSphere MQ clusters:
- DISPLAY CLUSQMGR
- SUSPEND QMGR
- RESUME QMGR
- REFRESH CLUSTER
- RESET CLUSTER

The PCF equivalents to these commands are:
- MQCMD_INQUIRE_CLUSTER_Q_MGR
- MQCMD_SUSPEND_Q_MGR_CLUSTER
- MQCMD_RESUME_Q_MGR_CLUSTER
- MQCMD_REFRESH_CLUSTER
- MQCMD_RESET_CLUSTER

These are described in the *WebSphere MQ Programmable Command Formats and Administration Interface* book.

DISPLAY CLUSQMGR

Use the DISPLAY CLUSQMGR command to display cluster information about queue managers in a cluster. If you issue this command from a queue manager with a full repository, the information returned pertains to every queue manager in the cluster. If you issue this command from a queue manager that does not have a full repository, the information returned pertains only to the queue managers in which it has an interest. That is, every queue manager to which it has tried to send a message and every queue manager that holds a full repository.

The information includes most channel attributes that apply to cluster-sender and cluster-receiver channels, such as:

DEFTYPE	How the queue manager was defined. DEFTYPE can be one of the following: **CLUSSDR** Defined explicitly as a cluster-sender channel **CLUSSDRA** Defined by auto-definition as a cluster-sender channel **CLUSSDRB** Defined as a cluster-sender channel, both explicitly and by auto-definition **CLUSRCVR** Defined as a cluster-receiver channel
QMTYPE	Whether it holds a full repository or only a partial repository.
CLUSDATE	The date at which the definition became available to the local queue manager.
CLUSTIME	The time at which the definition became available to the local queue manager.
STATUS	The current status of the cluster-sender channel for this queue manager.
SUSPEND	Whether the queue manager is suspended.
CLUSTER	What clusters the queue manager is in.
CHANNEL	The cluster-receiver channel name for the queue manager.

SUSPEND QMGR and RESUME QMGR

Use the SUSPEND QMGR command and RESUME QMGR command to remove a queue manager from a cluster temporarily, for example for maintenance, and then to reinstate it. Use of these commands is discussed in "Maintaining a queue manager" on page 76.

REFRESH CLUSTER

You are unlikely to need to use this command, except in exceptional circumstances. Issue the REFRESH CLUSTER command from a queue manager to discard all locally held information about a cluster.

There are two forms of this command using the REPOS parameter.

Using REFRESH CLUSTER(clustername) REPOS(NO) provides the default behavior. The queue manager will retain knowledge of all cluster queue manager and cluster queues marked as locally defined and all cluster queue managers that are marked as full repositories. In addition, if the queue manager is a full repository for the cluster it will also retain knowledge of the other cluster queue managers in the cluster. Everything else will be removed from the local copy of the repository and rebuilt from the other full repositories in the cluster. Cluster channels will not be stopped if REPOS(NO) is used, a full repository will use its CLUSSDR channels to inform the rest of the cluster that it has completed its refresh.

Using REFRESH CLUSTER(clustername) REPOS(YES) specifies that in addition to the default behavior, objects representing full repository cluster queue managers are also refreshed. This option may not be used if the queue manager is itself a full repository. If it is a full repository, you must first alter it so that it is not a full repository for the cluster in question. The full repository location will be recovered from the manually defined CLUSSDR definitions. After the refresh with REPOS(YES) has been issued the queue manager can be altered so that it is once again a full repository, if required.

You can issue REFRESH CLUSTER(*). This refreshes the queue manager in all of the clusters it is a member of. If used with REPOS(YES) this has the additional effect of forcing the queue manager to restart its search for full repositories from the information in the local CLUSSDR definitions, even if the CLUSSDR connects the queue manager to several clusters.

For information on resolving problems with the REFRESH CLUSTER command see “Resolving Problems” on page 115.

RESET CLUSTER

You are unlikely to need to use this command, except in exceptional circumstances. Use the RESET CLUSTER command to forcibly remove a queue manager from a cluster. You can do this from a full repository queue manager by issuing either the command:

```
RESET CLUSTER(clustername) QMNAME(qmname) ACTION(FORCEREMOVE)  QUEUES(NO)
```

or the command:

```
RESET CLUSTER(clustername) QMID(qmid) ACTION(FORCEREMOVE)  QUEUES(NO)
```

You cannot specify both QMNAME and QMID.

Specifying QUEUES(NO) on a RESET CLUSTER command is the default. Specifying QUEUES(YES) means that reference to cluster queue or queues owned by the queue manager being force removed are removed from the cluster in addition to the cluster queue manager itself. The cluster queues are removed even if the cluster queue manager is not visible in the cluster, perhaps because it was previously force removed without the QUEUES option.

Commands for work with clusters

You might use the RESET CLUSTER command if, for example, a queue manager has been deleted but still has cluster-receiver channels defined to the cluster. Instead of waiting for WebSphere MQ to remove these definitions (which it does automatically) you can issue the RESET CLUSTER command to tidy up sooner. All other queue managers in the cluster are then informed that the queue manager is no longer available.

In an emergency where a queue manager is temporarily damaged, you might want to inform the rest of the cluster before the other queue managers try to send it messages. RESET CLUSTER can be used to remove the damaged queue manager. Later when the damaged queue manager is working again, you can use the REFRESH CLUSTER command to reverse the effect of RESET CLUSTER and put it back in the cluster again.

Using the RESET CLUSTER command is the only way to delete auto-defined cluster-sender channels. You are unlikely to need this command in normal circumstances, but your IBM® Support Center might advise you to issue the command to tidy up the cluster information held by cluster queue managers. Do not use this command as a short cut to removing a queue manager from a cluster. The correct way to do this is described in “Task 5: Removing a queue manager from a cluster” on page 93.

You can issue the RESET CLUSTER command only from full repository queue managers.

If you use QMNAME, and there is more than one queue manager in the cluster with that name, the command is not actioned. Use QMID instead of QMNAME to ensure the RESET CLUSTER command is actioned.

Because repositories retain information for only 90 days, after that time a queue manager that was forcibly removed can reconnect to a cluster. It does this automatically (unless it has been deleted). If you want to prevent a queue manager from rejoining a cluster, you need to take appropriate security measures. See “Preventing queue managers joining a cluster” on page 83.

All cluster commands (except DISPLAY CLUSQMGR) work asynchronously. Commands that change object attributes involving clustering will update the object and send a request to the repository processor . Commands for working with clusters will be checked for syntax, and a request will be sent to the repository processor.

The requests sent to the repository processor are processed asynchronously, along with cluster requests received from other members of the cluster. In some cases, processing may take a considerable time if they have to be propagated around the whole cluster to determine if they are successful or not.

On z/OS

In both cases, message CSQM130I will be sent to the command issuer indicating that a request has been sent. This message is followed by message CSQ9022I to indicate that the command has completed successfully, in that a request has been sent. It does not indicate that the cluster request has been completed successfully.

Any errors are reported to the z/OS console on the system where the channel initiator is running, they are not sent to the command issuer.

This is not like channel operation commands, which operate synchronously, but in two stages.

Chapter 7. Managing WebSphere MQ clusters

This chapter provides information for the system administrator, including:
- "Designing clusters"
- "Cluster-administration considerations" on page 76

Designing clusters

This chapter will look at how to design a new cluster.

Selecting queue managers to hold full repositories

In each cluster you must select at least one, preferably two, or possibly more of the queue managers to hold full repositories. A cluster can work quite adequately with only one full repository but using two improves availability. You interconnect the full repository queue managers by defining cluster-sender channels between them.

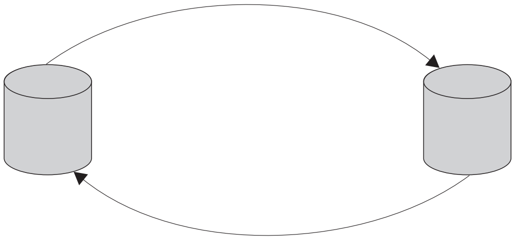

Figure 12. A typical 2-repository topology

Figure 12 shows a typical 2-full repository topology. This is the topology used in the cluster shown in Figure 2 on page 5.

- The most important consideration is that the queue managers chosen to hold full repositories need to be reliable and well managed. For example, it would be far better to choose queue managers on a stable z/OS system than queue managers on a portable personal computer that is frequently disconnected from the network.
- You might also consider the location of the queue managers and choose ones that are in a central position geographically or perhaps ones that are located on the same system as a number of other queue managers in the cluster.
- Another consideration might be whether a queue manager already holds the full repositories for other clusters. Having made the decision once, and made the necessary definitions to set up a queue manager as a full repository for one cluster, you might well choose to rely on the same queue manager to hold the full repositories for other clusters of which it is a member.

When a queue manager sends out information about itself or requests information about another queue manager, the information or request is sent to two full repositories. A full repository named on a CLUSSDR definition handles the request whenever possible, but if the chosen full repository is not available another full repository is used. When the first full repository becomes available again it collects the latest new and changed information from the others so that they keep in step.

Design considerations

In very large clusters, containing thousands of queue managers, you might want to have more than two full repositories. Then you might have one of the following topologies. (These are only example topologies)

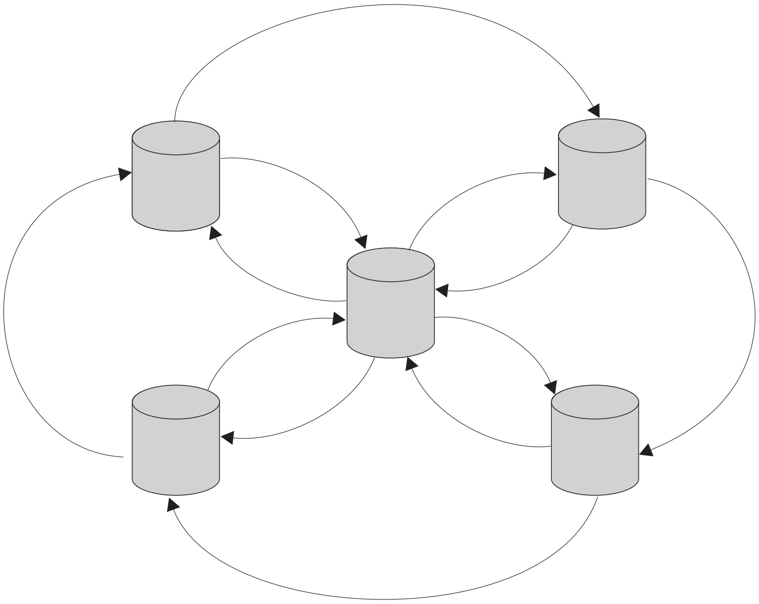

Figure 13. A hub and spoke arrangement of full repositories

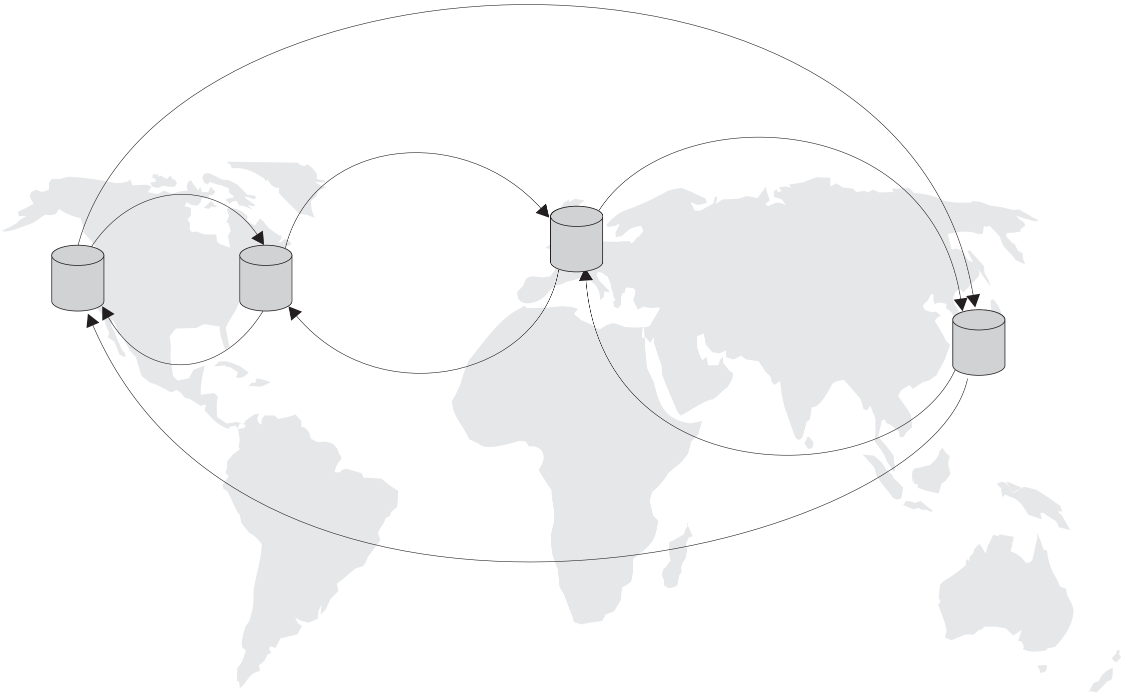

Figure 14. A complex full repository topology

If all the full repository queue managers go out of service at the same time, queue managers continue to work using the information they have in their partial repositories. Clearly they are limited to using the information that they have. New information and requests for updates cannot be processed. When the full repository queue managers reconnect to the network, messages are exchanged to bring all repositories (both full and partial) back up-to-date.

The full repositories republish the publications they receive through the manually-defined CLUSSDR channels, which must point to other full repositories in the cluster. You **must** make sure that a publication received by any full repository ultimately reaches all the other full repositories. You do this by manually defining CLUSSDR channels between the full repositories. The more interconnection of full repositories you have, the more robust the cluster.

Having only two full repositories is sufficient for all but very exceptional circumstances.

Organizing a cluster

Having selected the queue managers to hold full repositories, you need to decide which queue managers should 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, but it always tries to use the one to which it has a CLUSSDR channel definition first. It is not significant which full repository you choose. However, you should consider the topology of your configuration, and perhaps the physical or geographical location of the queue managers as shown in Figure 12 through Figure 14.

Design considerations

WebSphere MQ Explorer tries to contact a full repository queue manager in your cluster in order to build its displays. If you are using a z/OS system as a full repository, the Explorer will not be able to contact it because z/OS queue managers cannot run the command server to respond to the Explorer's PCF commands. To ensure that a particular full repository queue manager is not used by the WebSphere MQ Explorer, include the string %NOREPOS% in the description field of its cluster-receiver channel definition. When the explorer chooses which full repository to contact, it ignores those whose channel description contains %NOREPOS%, and treats them as though they did not hold a full repository for the cluster.

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 do this in a cluster that has a large number of full repositories, spread over a wide area, to control which full repositories your information is sent to.

Naming Convention

When setting up a new cluster, consider a naming convention for the queue managers. Every queue manager must have a different name, but it might help you to remember which queue managers are grouped where if you give them a set of similar names.

Every cluster-receiver channel must also have a unique name. One possibility is to use the queue-manager name preceded by the preposition TO. For example `TO.QM1`, `TO.QM2`, and so on. If you have more than one channel to the same queue manager, each with different priorities or using different protocols you might extend this convention to use names such as `TO.QM1.S1`, `TO.QM1.N3`, and `TO.QM1.T4`. `A1` might be the first SNA channel, `N3` might be the NetBIOS channel with a network priority of 3, and so on.

The find qualifier might describe the class of service the channel provides. See "Defining classes of service" on page 75 for more details.

Remember that all cluster-sender channels have the same name as their corresponding cluster-receiver channel.

Do not use generic connection names on your cluster-receiver definitions. In WebSphere MQ for z/OS you can define VTAM generic resources or Dynamic Domain Name Server (DDNS) generic names, but do not do this if you are using clusters. If you define a CLUSRCVR with a generic CONNAME there is no guarantee that your CLUSSDR channels will point to the queue managers you intend. Your initial CLUSSDR might end up pointing to any queue manager in the queue-sharing group, not necessarily one that hosts a full repository. Furthermore, if a channel goes to retry status, it might reconnect to a different queue manager with the same generic name and the flow of your messages will be disrupted.

Overlapping clusters

You can create clusters that overlap, as described in "Putting across clusters" on page 45. There are a number of reasons you might do this, for example:

- To allow different organizations to have their own administration.
- To allow independent applications to be administered separately.
- To create classes of service.
- To create test and production environments.

In Figure 10 the queue manager QM2 is a member of both the clusters illustrated. 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. A namelist can contain a list of names, for example, cluster names. You can create a namelist naming the clusters, and then specify this namelist on the ALTER QMGR command for QM2 to make it a full repository queue manager for both clusters. See “Task 8: Adding a new, interconnected cluster” on page 101 for some examples of how to use namelists.

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 will not be possible to separate them again**. It is also a good idea to give the clusters and channels different names so that they are more easily distinguished when you look at the output from 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 knows 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 15. WebSphere MQ will move 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.

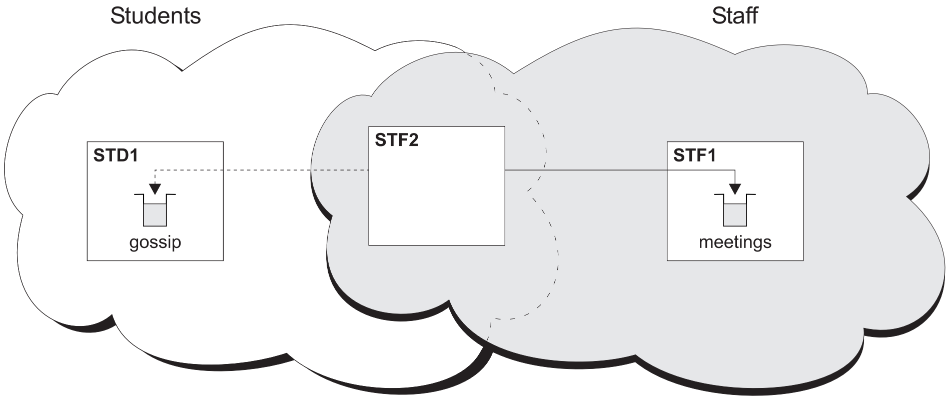

Figure 15. Classes of service

Objects

The following objects are needed when using WebSphere MQ clusters. They are included in the set of default objects defined when you create a queue manager except on z/OS, where they can be found in the customization samples.

Do **not** alter the default queue definitions. You can alter the default channel definitions in the same way as any other channel definition, using MQSC or PCF commands.

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.COMMAND.QUEUE
Each queue manager in a cluster has a local queue called SYSTEM.CLUSTER.COMMAND.QUEUE. This queue is used to carry messages to the full repository. The queue manager uses this queue to send any new or changed information about itself to the full repository queue manager and to send any requests for information about other queue managers. This queue is normally empty.

SYSTEM.CLUSTER.TRANSMIT.QUEUE
Each queue manager has a definition for a local queue called SYSTEM.CLUSTER.TRANSMIT.QUEUE. This is the transmission queue for all messages to all queues and queue managers that are within clusters.

SYSTEM.DEF.CLUSSDR
Each cluster has a default CLUSSDR channel definition called SYSTEM.DEF.CLUSSDR. This 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.

SYSTEM.DEF.CLUSRCVR
Each cluster has a default CLUSRCVR channel definition called SYSTEM.DEF.CLUSRCVR. This 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.

Cluster-administration considerations

Let us now look at some considerations affecting the system administrator.

Maintaining a queue manager

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.

To suspend a queue manager, issue the SUSPEND QMGR command, for example:

```
SUSPEND QMGR CLUSTER(SALES)
```

This sends a notification to the queue managers in the cluster SALES advising them 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. While the queue manager is suspended the workload management routines avoid sending messages to it, other than messages that **have** to be handled by that queue manager. Messages that **have** to be handled by that queue manager include messages sent by the local queue manager. The workload management routines choose the local queue manager whenever possible, even if it is suspended.

When the maintenance is complete the queue manager can resume its position in the cluster. It should issue the command RESUME QMGR, for example:

```
RESUME QMGR CLUSTER(SALES)
```

This notifies to 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.

You can enforce the suspension of a queue manager by using the FORCE option on the SUSPEND QMGR command, for example:

```
SUSPEND QMGR CLUSTER(SALES) MODE(FORCE)
```

This forcibly stops all inbound channels from other queue managers in the cluster. If you do not specify MODE(FORCE), the default MODE(QUIESCE) applies.

Refreshing a queue manager

A queue manager can make a fresh start in a cluster. This is unlikely to be necessary in normal circumstances but you might be asked to do this by your IBM Support Center. You can issue the REFRESH CLUSTER command from a queue manager to remove all cluster queue-manager objects and all cluster queue objects relating to queue managers other than the local one, from the local full repository. The command also removes any auto-defined channels that do not have messages on the cluster transmission queue and that are not attached to a full repository queue manager. Effectively, the REFRESH CLUSTER command allows a queue manager to be *cold started* with respect to its full repository content. (WebSphere MQ ensures that no data is lost from your queues.)

Recovering a queue manager

To recover a queue manager in a cluster, restore the queue manager from a linear log. (See the *WebSphere MQ System Administration Guide* for details)

If you have to restore from a point-in-time backup, issue the REFRESH CLUSTER command on the restored queue manager for all clusters in which the queue manager participates.

There is no need to issue the REFRESH CLUSTER command on any other queue manager.

Maintaining the cluster transmission queue

The availability and performance of the cluster transmission queue are essential to the performance of clusters. Make sure that it does not become full, and take care not to accidentally issue an ALTER command to set it either get-disabled or put-disabled. Also make sure that the medium the cluster transmission queue is stored on (for example z/OS page sets) does not become full. For performance reasons, on z/OS set the INDXTYPE of the cluster transmission queue to CORRELID.

What happens when a queue manager fails?

If a message-batch is sent to a particular queue manager and that queue manager becomes unavailable this is what happens:

- With the exception of non-persistent messages on a fast channel (which might be lost) the undelivered batch of messages is backed out to the cluster transmission queue on the sending queue manager.

- If the backed-out batch of messages is not in doubt and the messages are not bound to the particular queue manager, the workload management routine is called. The workload management routine selects a suitable alternative queue manager and the messages are sent there.
- Messages that have already been delivered to the queue manager, or are in doubt, or have no suitable alternative, wait until the original queue manager becomes available again.

The restart can be automated using Automatic Restart Management (ARM) on z/OS, HACMP on AIX, or any other restart mechanism available on your platform.

What happens when a repository fails?

Cluster information is carried to repositories (whether full or partial) on a local queue called SYSTEM.CLUSTER.COMMAND.QUEUE. If this queue fills up, perhaps because the queue manager has stopped working, the cluster-information messages are routed to the dead-letter queue. If you observe that this is happening, from the messages on your queue-manager log or z/OS system console, you need to run an application to retrieve the messages from the dead-letter queue and reroute them to the correct destination.

If errors occur on a repository queue manager, messages tell you what error has occurred and how long the queue manager will wait before trying to restart. On WebSphere MQ for z/OS the SYSTEM.CLUSTER.COMMAND.QUEUE is get-disabled. When you have identified and resolved the error, get-enable the SYSTEM.CLUSTER.COMMAND.QUEUE so that the queue manager can restart successfully.

In the unlikely event of a queue manager's repository running out of storage, storage allocation errors appear on your queue-manager log or z/OS system console. If this happens, stop and then restart the queue manager. When the queue manager is restarted, more storage is automatically allocated to hold all the repository information.

What happens if I put-disable a cluster queue?

When a cluster queue is put-disabled, this situation is reflected in the full repository of each queue manager that is interested in that queue. The workload management algorithm tries to send messages to destinations that are put-enabled. If there are no put-enabled destinations and no local instance of a queue, an MQOPEN call that specified MQOO_BIND_ON_OPEN returns a return code of MQRC_CLUSTER_PUT_INHIBITED to the application. If MQOO_BIND_NOT_FIXED is specified, or there is a local instance of the queue, an MQOPEN call succeeds but subsequent MQPUT calls fail with return code MQRC_PUT_INHIBITED.

You can write a user exit program to modify the workload management routines so that messages can be routed to a destination that is put-disabled. If a message arrives at a destination that is put-disabled (because it was in flight at the time the queue became disabled or because a workload exit chose the destination explicitly), the workload management routine at the queue manager can choose another appropriate destination if there is one, or place the message on the dead-letter queue, or if there is no dead-letter queue, return the message to the originator.

How long do the queue manager repositories retain information?

When a queue manager sends out some information about itself, for example to advertise the creation of a new queue, the full and partial repository queue managers store the information for 30 days. 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 it sees an expiry time of the remaining period. 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 may 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 will become 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 WebSphere MQ checks the request. If it has been referenced during the 27 days, it is remade automatically. If not, it is left to expire and is remade by the queue manager if it is needed again. This prevents a build up of requests for information from dormant queue managers.

Cluster channels

Although using clusters relieves you of the need to define channels (because WebSphere MQ defines them for you), the same channel technology used in distributed queuing is used for communication between queue managers in a cluster. 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

These topics are all discussed in the *WebSphere MQ Intercommunication* book and the advice given there is generally applicable to cluster channels, but you might want to give some special consideration to the following:

1. When you are defining cluster-sender channels and cluster-receiver channels choose a value for HBINT or KAINT that will detect a network or queue manager failure in a useful amount of time but not burden the network with too many heartbeat or keep alive flows. Bear in mind that choosing a short time, for example, less than about 10 seconds, will give false failures if your network sometimes slows down and introduces delays of this length.
2. Set the BATCHHB value if you want to reduce the window for causing a marooned message because it has got ″in doubt″ on a failed channel. This is more likely to occur if the message traffic along the channel is sporadic with long periods of time between bursts of messages, and during which a network failure is likely. This is sometimes a situation that is artificially induced when testing fail over of cluster queue managers, and may not be relevant on the production systems.
3. If the cluster-sender end of a channel fails and subsequently tries to restart before the heartbeat or keep alive has detected the failure, the restart is rejected if the cluster-receiver end of the channel has remained active. To avoid this,

you can arrange for the cluster-receiver channel to be terminated and restarted, when a cluster-sender channel attempts to restart.

On WebSphere MQ for z/OS
Control this using the ADOPTMCA and ADOPTCHK parameters of CSQ6CHIP. See the *WebSphere MQ for z/OS System Setup Guide* for more information.

On platforms other than z/OS
Control this using the AdoptNewMCA, AdoptNewMCATimeout, and AdoptNewMCACheck attributes in the qm.ini file or the Windows NT Registry. See the *WebSphere MQ System Administration Guide* Guide for more information.

Chapter 8. Keeping clusters secure

This chapter discusses the following security considerations:

- "Stopping unauthorized queue managers sending messages to your queue manager"
- "Stopping unauthorized queue managers putting messages on your queues"
- "Stopping your queue manager putting messages to remote queues" on page 82
- "Preventing queue managers joining a cluster" on page 83
- "Forcing unwanted queue managers to leave a cluster" on page 84
- "Using SSL" on page 84

Stopping unauthorized queue managers sending messages to your queue manager

To prevent selected queue managers from sending messages to your queue manager, define a channel security exit program on the CLUSRCVR channel definition. Write a program that authenticates queue managers trying to send messages on your cluster-receiver channel and denies them access if they are not authorized. Channel security exit programs are called at MCA initiation and termination. See the *WebSphere MQ Intercommunication* book for more information.

Clustering has no effect on the way security exits work. You can restrict access to a queue manager in the same way as you would in a distributed queuing environment.

By using SSL on your cluster—receiver channel you can force other queue managers in the cluster to authenticate themselves and keep out any that cannot.

Stopping unauthorized queue managers putting messages on your queues

To prevent certain queue managers from putting messages on a queue, use the security facilities available on your platform. For example:

- RACF® or other external security managers on WebSphere MQ for z/OS
- The Object Authority Manager (OAM) on WebSphere MQ for iSeries, WebSphere MQ on UNIX systems, and WebSphere MQ for Windows, and on MQSeries for Compaq Tru64 UNIX, V5.1, MQSeries for Compaq OpenVMS Alpha, V5.1, and MQSeries for Compaq NonStop Kernel, V5.1
- User-written procedures on MQSeries for OS/2 Warp

In addition, you can use the PUT authority (PUTAUT) attribute on the CLUSRCVR channel definition. The PUTAUT attribute allows you to specify what user IDs are to be used to establish authority to put a message to a queue. The options on the PUTAUT attribute are:

DEF	Use the default user ID. On z/OS this might involve using both the user ID received from the network and that derived from MCAUSER.
CTX	Use the user ID in the context information associated with the message. On

z/OS this might involve using either the user ID received from the network, or that derived from MCAUSER, or both. Use this option if the link is trusted and authenticated.

ONLYMCA (z/OS only)
As for DEF, but any user ID received from the network will not be used. Use this option if the link is not trusted and you want to allow only a specific set of actions on it, which are defined for the MCAUSER.

ALTMCA (z/OS only)
As for CTX, but any user ID received from the network will not be used.

For more information about using the PUTAUT attribute on a channel definition, see the *WebSphere MQ Intercommunication* book or see the *WebSphere MQ Script (MQSC) Command Reference* book.

Note: As with any other transmission queue, applications cannot put messages directly to SYSTEM.CLUSTER.TRANSMIT.QUEUE without special authorization.

Stopping your queue manager putting messages to remote queues

WebSphere MQ for z/OS
Use RACF to prevent your queue manager putting messages to a remote queue. With RACF you can set up permissions for a named queue regardless of whether that queue exists on your system. The authorization required is MQOO_OUTPUT.

Platforms other than z/OS
On these platforms you cannot restrict access to individual queues that do not exist on your queue manager. However, you can restrict access to **all** the queues in a cluster. For example, on queue manager CORK, to grant the user MYUSER access to the queues in a cluster, issue the following setmqaut commands:

```
setmqaut -m CORK -t qmgr -p MYUSER +connect
setmqaut -m CORK -n SYSTEM.CLUSTER.TRANSMIT.QUEUE
        -t queue -p MYUSER +put
```

On iSeries, the equivalent CL commands are:

```
GRTMQMAUT OBJ(CORK) OBJYTPE(*MQM) USER(MYUSER) AUT(*CONNECT)
GRTMQMAUT OBJ(SYSTEM.CLUSTER.TRANSMIT.QUEUE) OBJYTPE(*Q) +
        USER(MYUSER) AUT(*PUT) MQMNAME(CORK)
```

Setting access in this way allows the user MYUSER to put messages to any queue in the cluster.

It is possible to avoid the need to give general access to all cluster resources and +Put access to the transmit queue. You do this by defining alias or remote queue definitions on your machine which resolve to queues in the cluster, and giving the appropriate authority for access to these instead of the cluster transmit queue. For example, suppose there is a queue called Q1 in the clusters to which your queue manager CORK belongs. If you

```
DEFINE QALIAS(Q1) TARGQ(Q1) DEFBIND(NOTFIXED)
```

and then

```
setmqaut -m CORK -t qmgr -p GUEST +connect
setmqaut -m CORK -t queue -n Q1 -p GUEST -all +put
```

The user GUEST would only be able to send messages to the cluster queue Q1.

Note that it is not possible to use the same technique for a queue manager alias, because this requires access to the underlying SYSTEM.CLUSTER.TRANSMIT.QUEUE queue.

Preventing queue managers joining a cluster

If you want to ensure that only certain authorized queue managers attempt to join a cluster, you must either use a security exit program on the cluster-receiver channel, or write an exit program to prevent unauthorized queue managers from writing to SYSTEM.CLUSTER.COMMAND.QUEUE. Do not restrict access to SYSTEM.CLUSTER.COMMAND.QUEUE such that no queue manager can write to it, or you would prevent any queue manager from joining the cluster.

It is difficult to stop a queue manager that is a member of a cluster from defining a queue. Therefore, there is a danger that a rogue queue manager can join a cluster, learn what queues are in it, define its own instance of one of those queues, and so receive messages that it should not be authorized to receive.

To prevent a queue manager receiving messages that it should not, you can write:

- A channel exit program on each cluster-sender channel, which uses the connection name to determine the suitability of the destination queue manager to be sent the messages. By using SSL on the cluster-receiver of the full repository queue managers you can control which other queue managers can join the cluster.
- A cluster workload exit program, which uses the destination records to determine the suitability of the destination queue and queue manager to be sent the messages
- A channel auto-definition exit program, which uses the connection name to determine the suitability of defining channels to the destination queue manager

Using security exits on cluster channels

When a cluster-sender channel is first started, it uses attributes defined manually by a system administrator. When the channel is stopped and restarted, it picks up the attributes from the corresponding cluster-receiver channel definition. The original cluster-sender channel definition is overwritten with the new attributes, including the SecurityExit attribute. Note the following:

1. You must define a security exit on both the cluster-sender end and the cluster-receiver end of a channel, in order for it to be effective. Even though the security exit name is sent over from the cluster-receiver definition, the initial connection must be made with a security-exit handshake.
2. In addition to the normal security-message handshake, the security exit must validate the PartnerName in the MQCXP structure. The exit must allow the channel to start only if the partner queue manager is authorized.
3. Design the security exit on the cluster-receiver definition to be *receiver initiated*. If you design it as *sender initiated*, an unauthorized queue manager without a security exit can join the cluster because no security checks are performed. Not until the channel is stopped and restarted can the SCYEXIT name be sent over from the cluster-receiver definition and full security checks made. Refer to the *WebSphere MQ Intercommunication* book for information about sender-initiated and receiver-initiated security exits.
4. To view the cluster-sender channel definition that is currently in use, use the command:

```
DISPLAY CLUSQMGR(queue manager) ALL
```

This displays the attributes that have been sent across from the cluster-receiver definition. To view the original definition, use the command:

```
DISPLAY CHANNEL(channel name) ALL
```

5. If the queue managers are on different platforms, you might need to define a channel auto-definition exit on the cluster-sender queue manager to set the SecurityExit attribute. This is because the format of the exit name in the SecurityExit attribute is different for different platforms. For example, on z/OS the format is SCYEXIT('SECEXIT'), whereas on Windows it is SCYEXIT('C:/path/SECEXIT(function)'). Therefore, although the initial handshake can be accomplished successfully, when the attribute is passed from the cluster-receiver definition on one platform to the cluster-sender definition on another platform, it will have the wrong format. This results in an error saying that the user exit is not valid. To avoid this error, write a channel auto-definition exit to define the correctly-formatted security exit name on the remote queue manager.
6. On z/OS the security-exit load module must be in the data set specified in the CSQXLIB DD statement of the channel-initiator address-space procedure. On Windows the security-exit and channel auto-definition exit DLLs must be in the path specified in the SCYEXIT attribute of the channel definition or the CHADEXIT attribute of the queue manager definition respectively, or in the Registry.

Forcing unwanted queue managers to leave a cluster

You can force an unwanted queue manager to leave a cluster. You might need to do this to tidy up, if for example, a queue manager is deleted but its cluster-receiver channels are still defined to the cluster.

Only full repository queue managers are authorized to eject a queue manager from a cluster. For example, to eject the queue manager OSLO from the cluster NORWAY, the full repository queue manager issues the command:

```
RESET CLUSTER(NORWAY) QMNAME(OSLO) ACTION(FORCEREMOVE)
```

or

```
RESET CLUSTER(NORWAY) QMID(qmid) ACTION(FORCEREMOVE)
```

The queue manager that is force removed does not change, it believes it is still in the cluster. All other queue managers will believe it has gone.

For more information on RESET CLUSTER see "RESET CLUSTER" on page 67.

Using SSL

Note: SSL is available on the WebSphere MQ products only.

In a WebSphere MQ cluster a particular CLUSRCVR channel definition is frequently propagated to many other queue managers where it is transformed into an auto-defined CLUSSDR. Subsequently the auto-defined CLUSSDR is used to start a channel to the CLUSRCVR. If the CLUSRCVR is configured for SSL connectivity the following considerations apply:

- All queue managers that want to communicate with this CLUSRCVR must have access to SSL support. This SSL provision must support the CipherSpec for the channel.
- The different queue managers to which the auto-defined CLUSSDRs have been propagated will each have a different distinguished name associated. If distinguished name peer checking is to be used on the CLUSRCVR it must be set up so all of the distinguished names that can be received are successfully matched.

  For example, let us assume that all of the queue managers that will host CLUSSDRs which will connect to a particular CLUSRCVR, have certificates associated. Let us also assume that the distinguished names in all of these certificates define the country as UK, organization as IBM, the organization unit as WebSphere MQ Development, and all have common names in the form DEVT.QMxxx, where xxx is numeric.

  In this case an SSLPEER value of `C=UK, O=IBM, OU=WebSphere MQ Development, CN=DEVT.QM*` on the CLUSRCVR will allow all the required CLUSSDRs to connect successfully, but will prevent unwanted CLUSSDRs from connecting.
- If custom CipherSpec strings are used, be aware that the string formats are different on different platforms. An example of this is the CipherSpec string RC4_SHA_US has a value of 05 on z/OS and 0x6801, 128, 0x8004 on Windows. So if custom SSLCIPH parameters are used on a CLUSRCVR, all resulting auto-defined CLUSSDRs should reside on platforms on which the underlying SSL support implements this CipherSpec and on which it can be specified with the custom value. If you cannot select a value for the SSLCIPH parameter that will be understood throughout your WebSphere MQ cluster you will need a channel auto definition exit to change it into something the platforms being used will understand. Use the textual CipherSpec strings where possible (for example RC4_MD5_US).

When upgrading the queue managers in a cluster to WebSphere MQ V5.3, users are advised to upgrade their full repository queue managers first. The new SSL parameters are flowed through the cluster. If not all parts of the cluster are running WebSphere MQ V5.3, these parameters are discarded by the first queue manager not at V5.3 they encounter. The following example describes what happens if you do not upgrade your full repository queue managers first.

- Say you have a queue on a V5.3 partial repository queue manager and you want to access it from another V5.3 partial repository queue manager. You specify a CLUSRCVR on the first partial repository and give it SSL parameters. This flows to the second partial repository through a full repository.
- If the full repository is on a V5.3 queue manager, the SSL parameters are passed through it and a successful SSL connection is established between the two partial repositories. If the full repository is not on a V5.3 queue manager, it discards the SSL parameters without error and forwards the depleted auto-defined CLUSSDR object to the second partial repository. This is then used to attempt a connection to the SSL CLUSRCVR which will fail.

An SSLCRLNL parameter applies to an individual queue manager and is not propagated to other queue managers within a cluster.

Chapter 9. Advanced tasks

This chapter shows some advanced tasks that extend the cluster created in "Task 1: Setting up a new cluster" on page 19 and "Task 2a: Adding a new queue manager to a cluster" on page 27.

These tasks are:
- "Task 3: Adding a new queue manager that hosts a queue"
- "Task 4: Removing a cluster queue from a queue manager" on page 91
- "Task 5: Removing a queue manager from a cluster" on page 93
- "Task 6: Moving a full repository to another queue manager" on page 95

The chapter then goes on to demonstrate four further tasks:
- "Task 7: Converting an existing network into a cluster" on page 97
- "Task 8: Adding a new, interconnected cluster" on page 101
- "Task 9: Removing a cluster network" on page 106
- "Task 10: Adding new queue managers that host a shared queue" on page 108

You can perform these tasks, and the two described in Chapter 3, "First tasks", on page 19, without stopping your existing cluster queue managers or disrupting your existing network in any way.

Much of the information you need to achieve these tasks is documented elsewhere in the WebSphere MQ library. This chapter gives pointers to that information and fills in details relating specifically to work with clusters.

Notes:

1. Throughout the examples in this chapter and Chapter 3, "First tasks", on page 19, the queue managers have illustrative names such as LONDON and NEWYORK. Don't forget that on WebSphere MQ for z/OS, queue-manager names are limited to 4 characters.
2. The names of the queue managers imply that each queue manager is on a separate machine. You could just as easily set up these examples with all the queue managers on the same machine.
3. The examples in these chapters show WebSphere MQ Script Commands (MQSC) as they would be entered by the system administrator at the command console. For information about other ways of entering commands, refer to Chapter 6, "Using WebSphere MQ commands with clusters", on page 61.

Task 3: Adding a new queue manager that hosts a queue

Scenario:

- The INVENTORY cluster has been set up as described in "Task 2a: Adding a new queue manager to a cluster" on page 27. 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 “Task 6: Moving a full repository to another queue manager” on page 95.

The steps required to complete task 3

Follow these steps:

1. Determine which full repository TORONTO should refer to first

Every queue manager in a cluster must refer to one or other of the full repositories to gather information about the cluster and so build 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 will communicate 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:

```
DEFINE CHANNEL(TO.TORONTO) CHLTYPE(CLUSRCVR) TRPTYPE(TCP)
CONNAME(TORONTO.CHSTORE.COM) CLUSTER(INVENTORY)
DESCR('Cluster-receiver channel for TORONTO')
```

This advertises the queue manager’s availability to receive messages from other queue managers in the cluster, INVENTORY.

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 we have chosen NEWYORK, so TORONTO needs the following definition:

```
DEFINE CHANNEL(TO.NEWYORK) CHLTYPE(CLUSSDR) TRPTYPE(TCP)
CONNAME(NEWYORK.CHSTORE.COM) CLUSTER(INVENTORY)
DESCR('Cluster-sender channel from TORONTO to repository at NEWYORK')
```

4. 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. See “Reviewing applications for message affinities” on page 53 for more information.

5. Install the inventory application on the system in Toronto

See the *WebSphere MQ Application Programming Guide* for information about how to do this.

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)
```

Now that you have completed all the definitions, if you have not already done so you should start the channel initiator on WebSphere MQ for z/OS and, on all platforms, start a listener program on queue manager TORONTO. The listener program listens for incoming network requests and starts the cluster-receiver channel when it is needed. See "Establishing communication in a cluster" on page 14 for more information.

The cluster achieved by task 3

The cluster set up by this task looks like this:

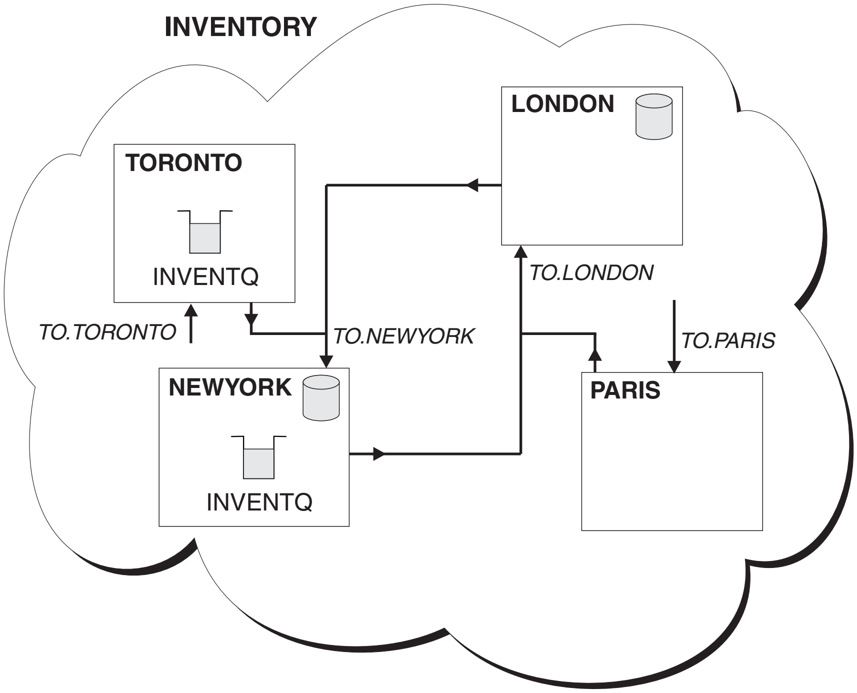

Figure 16. The INVENTORY cluster with four queue managers

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 make any alterations to the 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.

Assuming that the inventory application is designed appropriately and that there is sufficient processing capacity on the systems in New York and Toronto, the inventory application will continue to function if either the NEWYORK or the TORONTO queue manager becomes unavailable.

Extensions to this task

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 use the facility to allow even distribution of your workload, or you may decide to control the distribution yourself by using a *data partitioning* technique.

For example, suppose that you decide to add a customer-account query and update application running in LONDON and NEWYORK. Account information can only be held in one place, but you could arrange for half the records, for example for account numbers 00000 to 49999, to be held in LONDON, and the other half, in the range 50000 to 99999, to be held in NEWYORK. Write a cluster workload exit program to examine the account field in all messages, and route the messages to the appropriate queue manager.

Task 4: Removing a cluster queue from a queue manager

Scenario:

- The INVENTORY cluster has been set up as described in "Task 3: Adding a new queue manager that hosts a queue" on page 87. 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.

The steps required to complete task 4

Perform the following tasks:

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. Do this from queue manager TORONTO, using the ALTER QLOCAL command and specifying a blank cluster name, like this:

```
ALTER QLOCAL(INVENTQ) CLUSTER(' ')
```

2. 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 will go to the dead-letter queue. You need to stop all applications from putting messages explicitly to the queue on this queue manager.

3. Monitor the queue until it is empty

Monitor the queue using the DISPLAY QUEUE command and specifying the attributes IPPROCS, OPPROCS, and CURDEPTH, or use the WRKMQMQSTS command on iSeries. When the number of input processes, the number of output processes, and the current depth of the queue are all zero, you can be sure that the queue is empty.

4. Monitor the channel to ensure there are no in-doubt messages

To be sure that there are no in-doubt messages on the channel TO.TORONTO, monitor the cluster-sender channel called TO.TORONTO on each of the other queue managers. To do this, issue the DISPLAY CHSTATUS command specifying the INDOUBT parameter from each queue manager:

```
DISPLAY CHSTATUS(TO.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.

5. 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:

Removing a cluster queue from a queue manager

```
DELETE QLOCAL(INVENTQ)
```

The cluster achieved by task 4

The cluster set up by this task is similar to that set up by the previous task, except that the INVENTQ queue is no longer available at queue manager TORONTO.

When you take the queue out of service (step 1), the TORONTO queue manager sends a message to the two full repository queue managers notifying 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.

Now when a queue manager wants to put a message to the INVENTQ, it sees, from its updated partial repository, that the INVENTQ is available only at NEWYORK, and so sends its message there.

You can now remove the inventory application from the system in Toronto, to avoid duplication and save space on the system.

Extensions to this task

In this task description there is only one queue to remove and only one cluster to remove it from.

Suppose that there were 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. To do this, TORONTO defines a namelist of the cluster names:

```
DEFINE NAMELIST(TOROLIST)
       DESCR('List of clusters TORONTO is in')
       NAMES(INVENTORY, PAYROLL, SALES, PURCHASES)
```

and specifies this namelist on each queue definition, like this:

```
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 new namelist, containing the remaining list of cluster names, and then alter the queue definition to use the new namelist. To remove the PAYROLLQ from the INVENTORY cluster:

1. Create a new 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)
   ```

Task 5: Removing a queue manager from a cluster

Scenario:

- The INVENTORY cluster has been set up as described in "Task 3: Adding a new queue manager that hosts a queue" on page 87 and modified as described in "Task 4: Removing a cluster queue from a queue manager" on page 91.
- For business reasons you no longer want to carry out any inventory work at Toronto and so you want to remove the TORONTO queue manager from the cluster.

The steps required to complete task 5

Perform the following tasks at the TORONTO queue manager.

1. Suspend queue manager TORONTO

Issue the SUSPEND QMGR command to suspend availability of the queue manager to the INVENTORY cluster:

```
SUSPEND QMGR CLUSTER(INVENTORY)
```

When you issue this command, other queue managers are advised that they should refrain from sending messages to TORONTO.

2. Remove the CLUSRCVR channel definition

Remove the CLUSRCVR definition from the cluster:

```
ALTER CHANNEL(TO.TORONTO) CHLTYPE(CLUSRCVR) CLUSTER(' ')
```

This command causes the full repository queue managers to remove all information about that channel from their full repositories, so that queue managers will no longer try to send messages to it.

At this point messages should have stopped arriving in the TORONTO queue manager.

3. Stop the CLUSRCVR channel at TORONTO

Issue the STOP CHANNEL command to stop the cluster-receiver channel:

```
STOP CHANNEL(TO.TORONTO)
```

Once the channel is stopped, no more messages can be sent to TORONTO.

Later, to tidy up, you will probably want to delete the channel:

```
DELETE CHANNEL(TO.TORONTO)
```

4. Delete the CLUSSDR channel definition

The CLUSSDR channel definition points to the full repository at queue manager NEWYORK. Stop this channel as follows:

```
STOP CHANNEL(TO.NEWYORK)
```

and then delete it:

```
DELETE CHANNEL(TO.NEWYORK)
```

The cluster achieved by task 5

The cluster set up by this task looks like this:

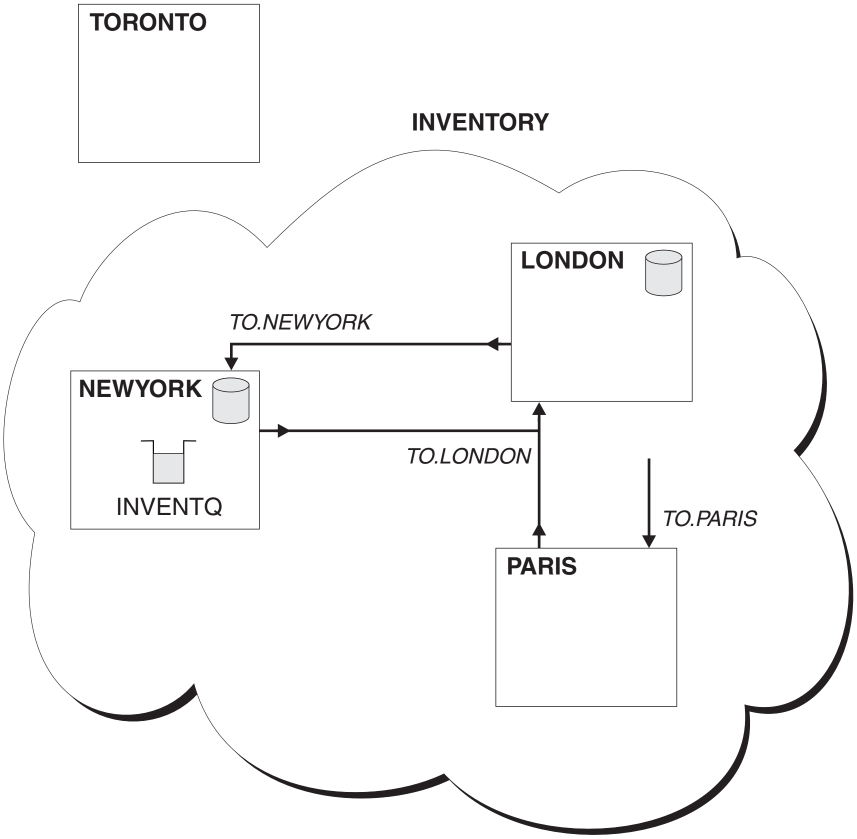

Figure 17. The INVENTORY cluster, with TORONTO outside the cluster

The queue manager TORONTO is no longer part of the cluster. However, it can still function as an independent queue manager.

This modification to the cluster was accomplished without you having to make any alterations to the queue managers NEWYORK, LONDON, and PARIS.

Note: If your queue manager is a full repository queue manager, before you can remove it from a cluster perform the additional step of altering the queue manager definition to set the REPOS and REPOSNL attributes to blank. This sends a notification to other queue managers advising them that they must stop sending cluster information to this queue manager.

Task 6: Moving a full repository to another queue manager

Scenario:

- The INVENTORY cluster has been set up as described in “Task 3: Adding a new queue manager that hosts a queue” on page 87 and modified as described in “Task 4: Removing a cluster queue from a queue manager” on page 91 and “Task 5: Removing a queue manager from a cluster” on page 93.
- 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.

The steps required to complete task 6

Perform the following tasks.

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. Now that 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 held.

```
DEFINE CHANNEL(TO.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(TO.PARIS) CHLTYPE(CLUSSDR) TRPTYPE(TCP)
CONNAME(PARIS.CHSTORE.COM) CLUSTER(INVENTORY)
DESCR('Cluster-sender channel from NEWYORK to repository at PARIS')
```

When you do this, the PARIS queue manager immediately learns all about the cluster from NEWYORK and builds up its own full repository.

4. 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.

5. Remove or change any outstanding definitions

When you are sure that the new arrangement of your cluster is working as expected, you can remove or change any outstanding definitions that are no longer up-to-date. It is not essential that you do this, but you might choose to in order to tidy up.

- On the PARIS queue manager, delete the cluster-sender channel to LONDON.

  ```
  DELETE CHANNEL(TO.LONDON)
  ```

- On the NEWYORK queue manager, delete the cluster-sender channel to LONDON.

```
DELETE CHANNEL(TO.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)
```

The cluster achieved by task 6

The cluster set up by this task looks like this:

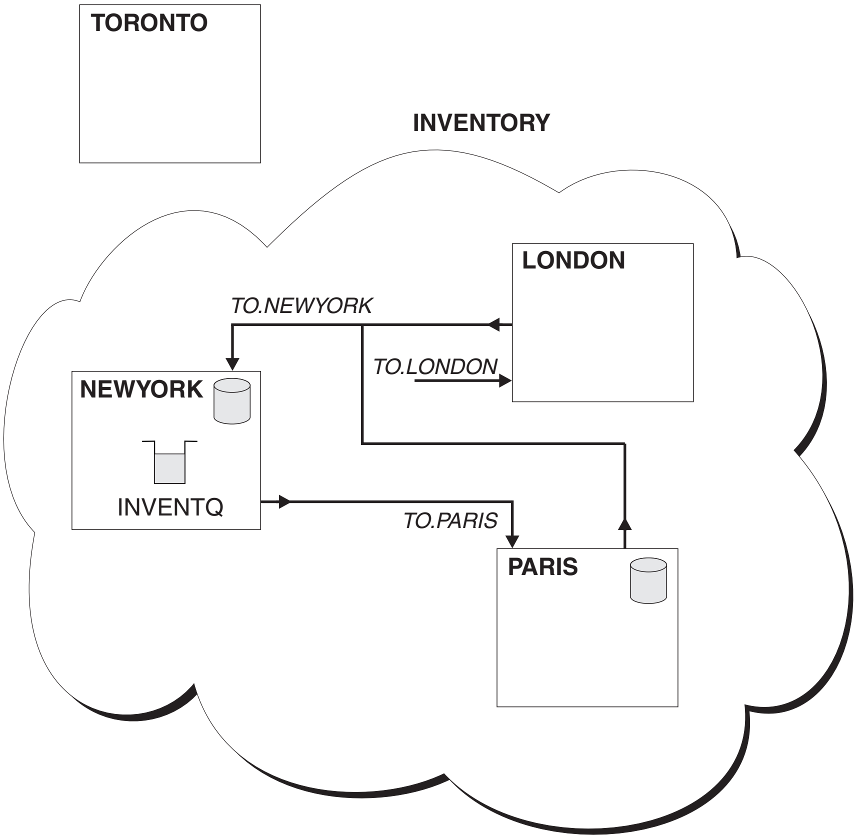

Figure 18. The INVENTORY cluster with the full repository moved to PARIS

Task 7: Converting an existing network into a cluster

"Task 1: Setting up a new cluster" on page 19 through "Task 6: Moving a full repository to another queue manager" on page 95 set up and then extend a new cluster. The remaining two tasks explore a different approach: that of converting an existing network of queue managers into a cluster.

Scenario:

- A 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 19.

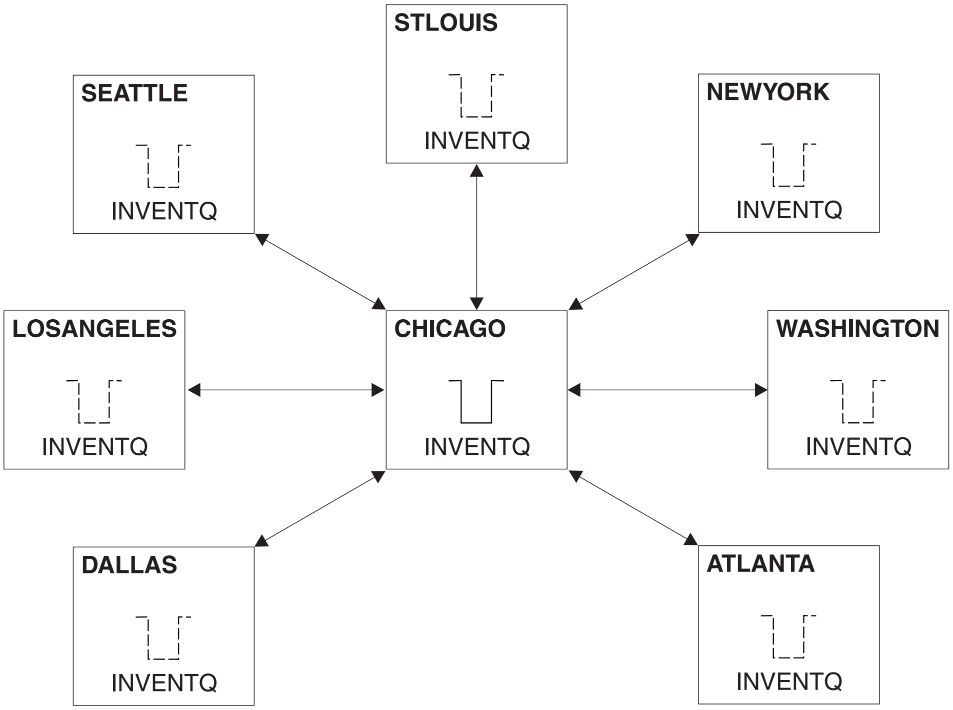

Figure 19. A hub and spoke network

- 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.
- 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 will continue to run as before.
- All the branches have network connectivity to the two central queue managers.
- The network protocol is TCP.

The steps required to complete task 7

Perform the following tasks.

Note: You do not need to convert your entire network all at once. This task could be completed in stages.

1. Review the inventory application for message affinities

Before proceeding ensure that the application can handle message affinities. See "Reviewing applications for message affinities" on page 53 for more information.

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 chainstore 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(CHAINSTORE)
```

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 of these you define first.

Make a CLUSRCVR definition to advertise each queue manager, its network address, and so on, to the cluster. For example, on queue manager ATLANTA:

```
DEFINE CHANNEL(TO.ATLANTA) CHLTYPE(CLUSRCVR) TRPTYPE(TCP)
CONNAME(ATLANTA.CHSTORE.COM) CLUSTER(CHAINSTORE)
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(TO.CHICAGO2) CHLTYPE(CLUSSDR) TRPTYPE(TCP)
CONNAME(CHICAGO2.CHSTORE.COM) CLUSTER(CHAINSTORE)
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. Refer to the *WebSphere MQ Application Programming Guide* and install the inventory application on 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(CHAINSTORE)
```

On CHICAGO2, make a definition for the same queue:

```
DEFINE QLOCAL(INVENTQ) CLUSTER(CHAINSTORE)
```

(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. See the *WebSphere MQ for z/OS System Administration Guide* for details.)

When you make these definitions, a message is sent to the full repositories at CHICAGO and CHICAGO2 and the information in them is updated. From then on, when a queue manager wants to put a message to the INVENTQ, it will find out from the full repositories that there is a choice of destinations for messages sent to that queue.

7. Delete all remote-queue definitions for the INVENTQ

Now that you have linked all your queue managers together in the CHAINSTORE cluster, and have defined the INVENTQ to the cluster, the queue managers no longer need remote-queue definitions for the INVENTQ. At every queue manager, issue the command:

```
DELETE QREMOTE(INVENTQ)
```

Until you do this, the remote-queue definitions will continue to be used and you will not get the benefit of using clusters.

8. Implement the cluster workload exit (optional step)

Because there is more than one destination for messages sent to the INVENTQ, the workload management algorithm will determine which destination each message will be sent to.

If you want to implement your own workload management routine, write a cluster workload exit program. See “Workload balancing” on page 49 for more information.

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 and, on all platforms, start a listener program on each queue manager. The listener program listens for incoming network requests and starts the cluster-receiver channel when it is needed. See “Establishing communication in a cluster” on page 14 for more information.

The cluster achieved by task 7

The cluster set up by this task looks like this:

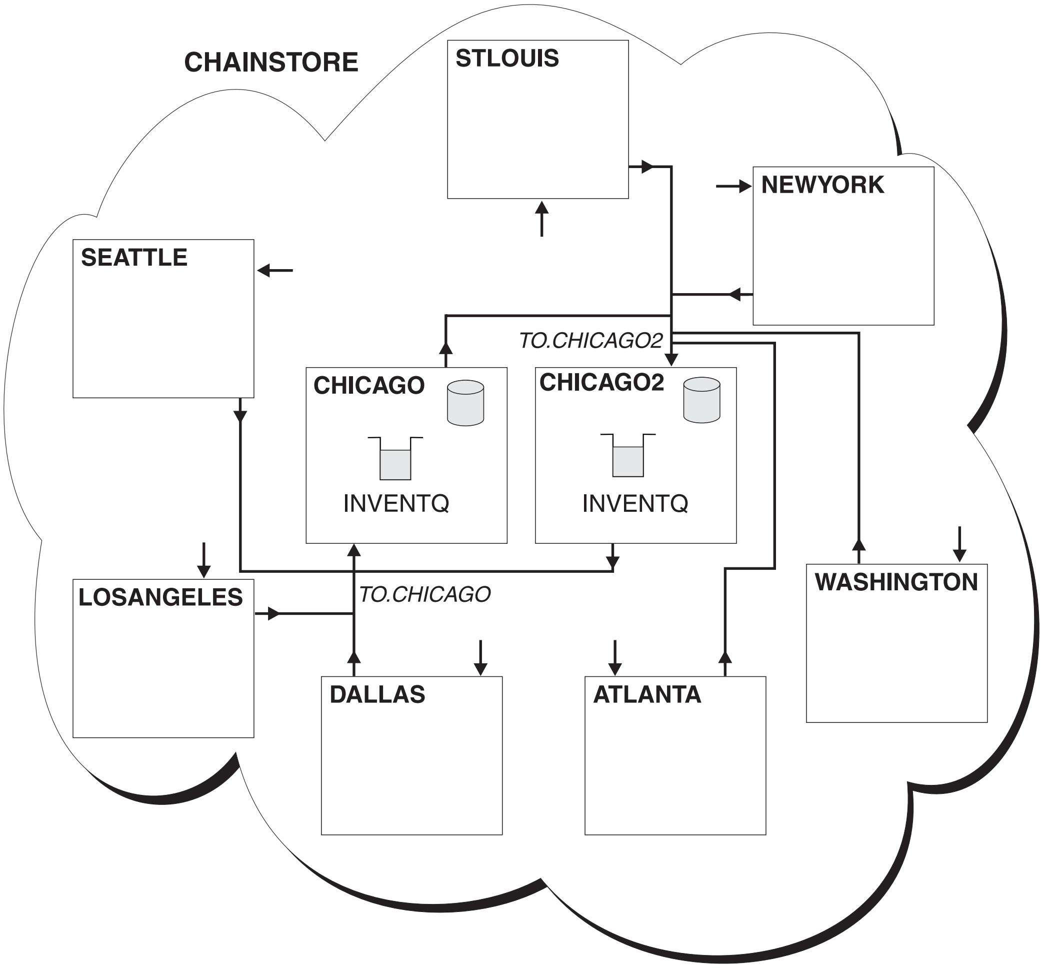

Figure 20. A cluster with a hub and spokes

Remember that this diagram shows only the channels that you have to define manually. Cluster-sender channels are defined automatically when needed so that ultimately all queue managers can receive cluster information from the two full repository queue managers and also messages from the two applications.

Once again, this is a very small example, little more than a proof of concept. In your enterprise it is unlikely that you will have a cluster of this size with only one queue.

You can easily add more queues to the cluster environment by adding the CLUSTER parameter to your queue definitions, and then removing all corresponding remote-queue definitions from the other queue managers.

Task 8: Adding a new, interconnected cluster

Scenario:

- A WebSphere MQ cluster has been set up as described in "Task 7: Converting an existing network into a cluster" on page 97.
- A new cluster called MAILORDER is to be implemented. This cluster will comprise four of the queue managers that are in the CHAINSTORE cluster; CHICAGO, CHIGACO2, SEATTLE, and ATLANTA, and two additional queue managers; HARTFORD and OMAHA. The MAILORDER application will run on the system at Omaha, connected to queue manager OMAHA. It will be driven by the other queue managers in the cluster putting messages on the MORDERQ queue.
- The full repositories for the MAILORDER cluster will be maintained on the two queue managers CHICAGO and CHICAGO2.
- The network protocol is TCP.

The steps required to complete task 8

Perform the following tasks.

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 CHAINSTORE 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(CHAINSTORE, 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 CHAINSTORE. Change that definition to show that the queue managers hold full repositories for all clusters listed in the CHAINMAIL namelist. To do this, at both CHICAGO and CHICAGO2, specify:

```
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 CHAINSTORE. You need to change this to show that the channels are available to all clusters listed in the CHAINMAIL namelist. To do this, at CHICAGO, enter the command:

```
ALTER CHANNEL(TO.CHICAGO) CHLTYPE(CLUSRCVR)
      CLUSTER(' ') CLUSNL(CHAINMAIL)
```

At CHICAGO2, enter the command:

```
ALTER CHANNEL(TO.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(TO.CHICAGO2) CHLTYPE(CLUSSDR)
      CLUSTER(' ') CLUSNL(CHAINMAIL)
```

At CHICAGO2, enter the command:

```
ALTER CHANNEL(TO.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(CHAINSTORE, 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 CHAINSTORE. Change this to show that the channels are available to all clusters listed in the CHAINMAIL namelist. To do this, at SEATTLE, enter the command:

```
ALTER CHANNEL(TO.SEATTLE) CHLTYPE(CLUSRCVR)
      CLUSTER(' ') CLUSNL(CHAINMAIL)
```

At ATLANTA, enter the command:

```
ALTER CHANNEL(TO.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(TO.CHICAGO) CHLTYPE(CLUSSDR)
      CLUSTER(' ') CLUSNL(CHAINMAIL)
```

At ATLANTA, enter the command:

```
ALTER CHANNEL(TO.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 doesn't matter in which sequence you do this. At HARTFORD, enter:

```
DEFINE CHANNEL(TO.HARTFORD) CHLTYPE(CLUSRCVR) TRPTYPE(TCP)
CONNAME(HARTFORD.CHSTORE.COM) CLUSTER(MAILORDER)
DESCR('Cluster-receiver channel for HARTFORD')

DEFINE CHANNEL(TO.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(TO.OMAHA) CHLTYPE(CLUSRCVR) TRPTYPE(TCP)
CONNAME(OMAHA.CHSTORE.COM) CLUSTER(MAILORDER)
DESCR('Cluster-receiver channel for OMAHA')

DEFINE CHANNEL(TO.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. To do this, enter:

```
DEFINE QLOCAL(MORDERQ) CLUSTER(MAILORDER)
```

The cluster achieved by task 8

The cluster set up by this task is shown in Figure 21 on page 104.

Now we have two overlapping clusters. The full repositories for both clusters are held at CHICAGO and CHICAGO2. The mailorder 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 CHAINSTORE 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 CHAINSTORE and on OMAHA in cluster MAILORDER, there is a queue called ACCOUNTQ. If you overlap the clusters and then an application on a queue manager that is a member of both clusters, for example SEATTLE, puts a message to the queue ACCOUNTQ, the message can go to either instance of the ACCOUNTQ.

Before starting this task the system administrators of the two clusters must check for queue-name clashes and be sure that they understand the consequences. You might need to rename a queue, or perhaps set up queue aliases before you can proceed.

Interconnecting a new cluster

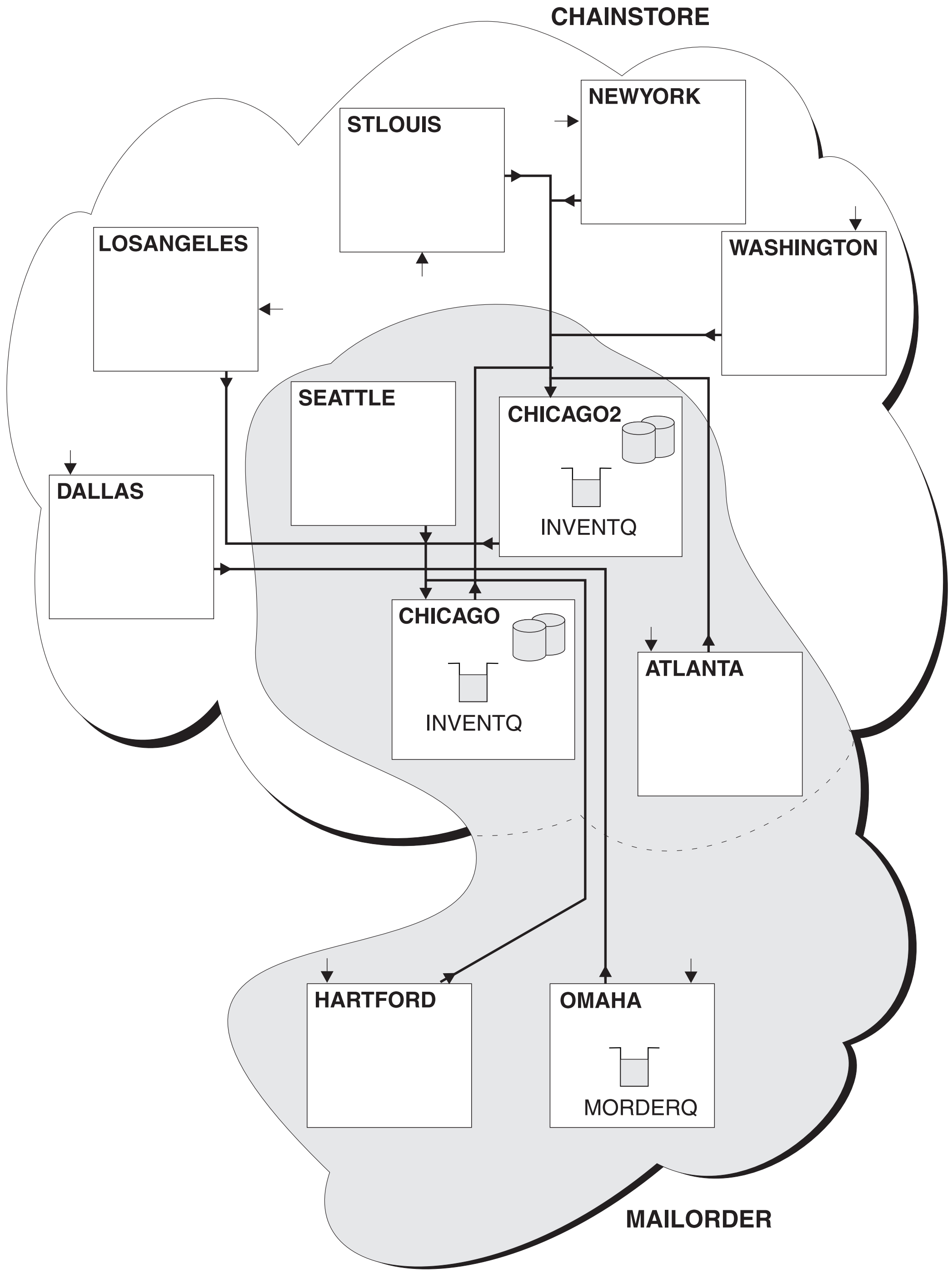

Figure 21. Interconnected clusters

Extensions to this task

Suppose you decide to merge the MAILORDER cluster with the CHAINSTORE cluster to form one large cluster called CHAINSTORE.

To merge the MAILORDER cluster with the CHAINSTORE 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 (CHAINSTORE). For example:

  ```
  ALTER QMGR(CHICAGO) REPOSNL(' ') REPOS(CHAINSTORE)
  ```

- 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 CHAINSTORE. For example, at HARTFORD, enter:

  ```
  ALTER CHANNEL(TO.HARTFORD) CLUSTER(CHAINSTORE)
  ```

  At OMAHA enter:

  ```
  ALTER QLOCAL(MORDERQ) CLUSTER(CHAINSTORE)
  ```

- 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 CHAINSTORE.

From this example, you can see the advantage of using namelists. Instead of altering the queue manager definitions for CHICAGO and CHICAGO2 you can just 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.

Task 9: Removing a cluster network

Scenario:

- A WebSphere MQ cluster has been set up as described in “Task 7: Converting an existing network into a cluster” on page 97.
- 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.

The steps required to complete task 9

Perform the following tasks.

1. Remove cluster queues from the cluster

On both CHICAGO and CHICAGO2, modify the local queue definition for the queue INVENTQ to remove the queue from the cluster. To do this, issue the command:

```
ALTER QLOCAL(INVENTQ) CLUSTER(' ')
```

When you do this, 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 don’t, perform it instead after Step 4.

2. Stop all applications that have access to cluster queues

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 will see a message indicating that the full repository has ended, and one or more messages indicating that there are no longer any repositories available for the cluster CHAINSTORE.

4. Remove cluster channels

On CHICAGO remove the cluster channels:

```
ALTER CHANNEL(TO.CHICAGO2) CHLTYPE(CLUSSDR) CLUSTER(' ')
ALTER CHANNEL(TO.CHICAGO) CHLTYPE(CLUSRCVR) CLUSTER(' ')
```

You will see messages indicating that there are no repositories for the cluster CHAINSTORE.

If you did not remove the cluster queues as described in Step 1, you should do so now.

5. Issue the REFRESH CLUSTER command

On CHICAGO remove the cluster information held in the queue manager’s full repository with the command:

```
REFRESH CLUSTER(CHAINSTORE) REPOS(YES)
```

All cluster information about the cluster CHAINSTORE is now removed from the queue manager. If cluster channels are still active, information regarding these channels remains in the queue manager’s partial repository until the channel is stopped.

6. Repeat Steps 4 and 5 for each queue manager in the cluster

Remove all definitions for cluster channels and cluster queues from each queue manager and issue the REFRESH CLUSTER command.

7. 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.

8. Tidy up the cluster

Delete any queue or channel definitions no longer required.

Task 10: Adding new queue managers that host a shared queue

Note: Queue-sharing groups are supported only on WebSphere MQ for z/OS, V5.3. This task is not applicable to other platforms.

Scenario:

- The INVENTORY cluster has been set up as described in “Task 1: Setting up a new cluster” on page 19. It contains two queue managers, LONDON and NEWYORK.
- You want to add a queue-sharing group to this cluster. The group, QSGPOLO, comprises three queue managers, POLO1, POLO2, and POLO3. They share an instance of the INVENTQ queue, which is to be defined by POLO1.

The steps required to complete task 10

Follow these steps:

1. Determine which full repository the queue managers will refer to first

Every member of a cluster must refer to one or other of the full repositories to gather information about the cluster and so build up its own partial repository. It is of no particular significance which full repository you choose. In this example we choose NEWYORK. Once the queue-sharing group has joined the cluster, it will communicate with both of the full repositories.

2. Define the CLUSRCVR channels

Every queue manager in a cluster needs to define a cluster-receiver channel on which it can receive messages. On POLO1, POLO2, and POLO3, define:

```
DEFINE CHANNEL(TO.POLOn) CHLTYPE(CLUSRCVR) TRPTYPE(TCP)
CONNAME(POLOn.CHSTORE.COM) CLUSTER(INVENTORY)
DESCR('Cluster-receiver channel for sharing queue manager')
```

This advertises each queue manager’s availability to receive messages from other queue managers in the cluster INVENTORY.

3. Define a CLUSSDR channel for the queue-sharing group

Every member of 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. One of the queue managers in the queue-sharing group needs the following group definition in order to ensure every queue manager has a cluster-sender channel definition:

```
DEFINE CHANNEL(TO.NEWYORK) CHLTYPE(CLUSSDR) TRPTYPE(TCP)
CONNAME(NEWYORK.CHSTORE.COM) CLUSTER(INVENTORY) QSGDISP(GROUP)
DESCR('Cluster-sender channel to repository at NEWYORK')
```

4. Define the shared queue

Define the queue INVENTQ on POLO1 as follows:

```
DEFINE QLOCAL(INVENTQ) CLUSTER(INVENTORY) QSGDISP(SHARED) CFSTRUCT(STRUCTURE)
```

Now that you have completed all the definitions, if you have not already done so, start the channel initiator and a listener program on the new queue manager. The listener program listens for incoming network requests and starts the cluster-receiver channel when it is needed. See "Establishing communication in a cluster" on page 14 for more information.

The cluster achieved by task 10

The cluster set up by this task looks like this:

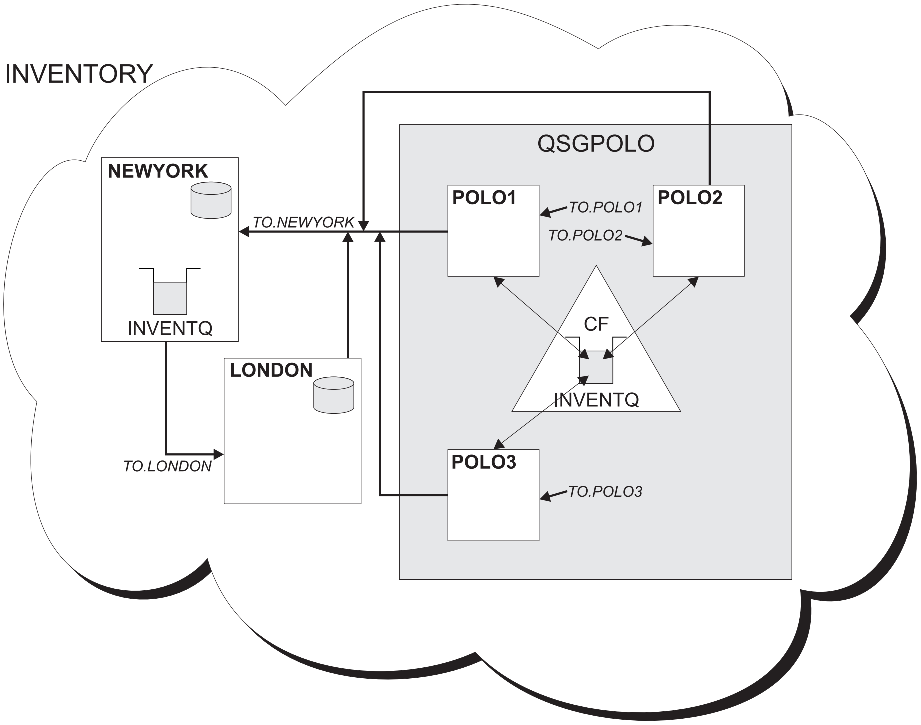

Figure 22. Cluster and queue-sharing group

Now messages put on the INVENTQ queue by LONDON are routed alternately around the four queue managers advertised as hosting the queue.

One of the benefits of having members of a queue-sharing group hosting a cluster queue is that, if the queue manager that made the queue definition (in this case POLO1) becomes unavailable after receiving a message on the shared queue, and is unable to reply, any other member of the queue-sharing group can reply instead.

Note: Also refer to the chapter on Intra-group queuing in the *WebSphere MQ Intercommunication* for details of configuring clustering with intra-group queuing.

Troubleshooting

The following is a list of Symptoms and their causes.

Symptom — A manually defined cluster sender channel is in retry state.

A manually defined cluster sender channel is in retry state.

```
1 : display chs(*)

AMQ8417: Display Channel Status details.

CHANNEL(TO.QM2)
XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)

CONNAME(computer.ibm.com(1414))

CURRENT                                 CHLTYPE(CLUSSDR)

STATUS(RETRYING)
```

Cause

Either the remote, full repository queue manager is not available or there is a bad parameter in the CLUSSDR channel definition, the conname may be incorrect. Alter the CLUSSDR channel definition, then STOP and START the channel.

Symptom — DISPLAY CLUSQMGR shows CLUSQMGR names starting SYSTEM.TEMP.

DISPLAY CLUSQMGR shows CLUSQMGR names starting SYSTEM.TEMP.

```
1 : display clusqmgr(*)

AMQ8441: Display Cluster Queue Manager details.

CLUSQMGR(QM1)                          CLUSTER(DEMO)

CHANNEL(TO.QM1)

AMQ8441: Display Cluster Queue Manager details.

CLUSQMGR(SYSTEM.TEMPQMGR.computer.hursley.ibm.com(1414))

CLUSTER(DEMO)                           CHANNEL(TO.QM2)
```

Cause

This queue manager has not received any information from the full repository queue manager that the manually defined CLUSSDR channel points to. The manually defined CLUSSDR should be in running state, check that the CLUSRCVR definition is also correct, especially its conname and cluster parameters. Alter the channel definition. It may take some time for the remote queue managers to complete a retry cycle and restart their channels with the corrected definition.

Symptom — Applications get rc=2085 MQRC_UNKNOWN_OBJECT_NAME when trying to open a queue in the cluster.

Applications get rc=2085 MQRC_UNKNOWN_OBJECT_NAME when trying to open a queue in the cluster.

Cause

The queue manager where the object exists or this queue manager may not have successfully entered the cluster. Make sure that they can each display all of the full repositories in the cluster. Also make sure that the CLUSSDR channels to the full repositories are not in retry state.

```
 1 : display clusqmgr(*) qmtype status

AMQ8441: Display Cluster Queue Manager details.

CLUSQMGR(QM1)                       CLUSTER(DEMO)

CHANNEL(TO.QM1)                        QMTYPE(NORMAL)

AMQ8441: Display Cluster Queue Manager details.

CLUSQMGR(QM2)                           CLUSTER(DEMO)

CHANNEL(TO.QM2)                         QMTYPE(REPOS)

STATUS(RUNNING)

AMQ8441: Display Cluster Queue Manager details.

CLUSQMGR(QM3)                           CLUSTER(DEMO)

CHANNEL(TO.QM3)                         QMTYPE(REPOS)

STATUS(RUNNING)
```

If the queue is correctly in the cluster check that you have used appropriate open options. You cannot GET from a remote cluster queue so make sure that the open options are for output only.

Symptom — Applications get rc= 2189 MQRC_CLUSTER_RESOLUTION_ERROR when trying to open a queue in the cluster.

Applications get rc= 2189 MQRC_CLUSTER_RESOLUTION_ERROR when trying to open a queue in the cluster.

Cause

The queue is being opened for the first time and the queue manager cannot make contact with any full repositories. Make sure that the CLUSSDR channels to the full repositories are not in retry state.

```
  1 : display clusqmgr(*) qmtype status

AMQ8441: Display Cluster Queue Manager details.
```

```
CLUSQMGR(QM1)                       CLUSTER(DEMO)
CHANNEL(TO.QM1)                        QMTYPE(NORMAL)
AMQ8441: Display Cluster Queue Manager details.
CLUSQMGR(QM2)                           CLUSTER(DEMO)
CHANNEL(TO.QM2)                         QMTYPE(REPOS)
STATUS(RUNNING)
AMQ8441: Display Cluster Queue Manager details.
CLUSQMGR(QM3)                           CLUSTER(DEMO)
CHANNEL(TO.QM3)                         QMTYPE(REPOS)
STATUS(RUNNING)
```

If the queue is correctly in the cluster check that you have used appropriate open options. You cannot GET from a remote cluster queue so make sure that the open options are for output only.

Symptom — Messages are not appearing on the destination queues.

Messages are not appearing on the destination queues.

Cause

The messages may be stuck at their origin queue manager. Make sure that the SYSTEM.CLUSTER.TRANSMIT.QUEUE is empty and also that the channel to the destination queue manager is running.

```
 1 : display ql(SYSTEM.CLUSTER.TRANSMIT.QUEUE) curdepth
AMQ8409: Display Queue details.
   QUEUE(SYSTEM.CLUSTER.TRANSMIT.QUEUE)    CURDEPTH(0)
     2 : display chs(TO.QM2)
AMQ8417: Display Channel Status details.
   CHANNEL(TO.QM2)
XMITQ(SYSTEM.CLUSTER.TRANSMIT.QUEUE)
  CONNAME(comfrey.hursley.ibm.com(1415))
   CURRENT                                CHLTYPE(CLUSSDR)
   STATUS(RUNNING)
```

Symptom — Applications put messages to a QALIAS but they go the SYSTEM.DEAD.LETTER.QUEUE rather then the TARGQ of the alias.

Applications put messages to a QALIAS but they go the SYSTEM.DEAD.LETTER.QUEUE rather then the TARGQ of the alias.

Cause

The alias was opened BIND_OPEN rather than BIND_NOT_FIXED. If BIND_OPEN is used the transmission header for the message contains the queue manager name of the destination queue manager instead of blanks.

When the message reaches the queue manager where the alias is defined it, uses its queue manager name and cannot find the destination queue there. Hence the message is put on the dead letter queue. Change all of the alias queue definitions to specify DEFBIND(NOTFIXED) or use BIND_NOT_FIXED as an open option when the queue is opened.

Symptom — A queue manager does not appear to have up to date information about queues and channels in the cluster.

A queue manager does not appear to have up to date information about queues and channels in the cluster. DISPLAY QCLUSTER and DISPLAY CLUSQMGR show objects which are out of date.

Cause

Check that the queue manager where the object exists and this queue manager are still connected to the cluster. Make sure that they can each display all of the full repositories in the cluster. Also make sure that the CLUSSDR channels to the full repositories are not in retry state, as above.

Next make sure that the full repositories have enough CLUSSDR channels defined to correctly connect them together. The updates to the cluster only flow between the full repositories over manually defined CLUSSDR channels. After the cluster has formed these will show as DEFTYPE(CLUSSDRB) channels because they are both manual and automatic channels. There must be enough of these to form a complete network among all of the full repositories.

```
 1 : display ql(SYSTEM.CLUSTER.TRANSMIT.QUEUE) curdepth

AMQ8441: Display Cluster Queue Manager details.

   CLUSQMGR(QM1)                          CLUSTER(DEMO)

   CHANNEL(TO.QM1)                         DEFTYPE(CLUSSDRA)

   QMTYPE(NORMAL)                          STATUS(RUNNING)

AMQ8441: Display Cluster Queue Manager details.

   CLUSQMGR(QM2)                          CLUSTER(DEMO)

   CHANNEL(TO.QM2)                        DEFTYPE(CLUSRCVR)

   QMTYPE(REPOS)

AMQ8441: Display Cluster Queue Manager details.

   CLUSQMGR(QM3)                           CLUSTER(DEMO)

   CHANNEL(TO.QM3)                         DEFTYPE(CLUSSDRB)

   QMTYPE(REPOS)                           STATUS(RUNNING)

AMQ8441: Display Cluster Queue Manager details.

   CLUSQMGR(QM4)                           CLUSTER(DEMO)

   CHANNEL(TO.QM4)                         DEFTYPE(CLUSSDRA)

   QMTYPE(NORMAL)                          STATUS(RUNNING)
```

Symptom — No changes in the cluster are being reflected in the local queue manager.

No changes in the cluster are being reflected in the local queue manager.

Cause

The repository manager process in not processing repository commands. Check that the SYSTEM.CLUSTER.COMMAND.QUEUE is empty.

```
     1 : display ql(SYSTEM.CLUSTER.COMMAND.QUEUE) curdepth
AMQ8409: Display Queue details.
   QUEUE(SYSTEM.CLUSTER.COMMAND.QUEUE)     CURDEPTH(0)
```

Next check that the channel initiator is running on z/OS. Then check that there are no error messages in the error logs indicating the queue manager has a temporary resource shortage.

Symptom — DISPLAY CLUSQMGR, shows a queue manager twice.

DISPLAY CLUSQMGR, shows a queue manager twice.

```
     1 : display clusqmgr(QM1) qmid
AMQ8441: Display Cluster Queue Manager details.
   CLUSQMGR(QM1)                          CLUSTER(DEMO)
   CHANNEL(TO.QM1)                         QMID(QM1_2002-03-
04_11.07.01)
AMQ8441: Display Cluster Queue Manager details.
   CLUSQMGR(QM1)                          CLUSTER(DEMO)
   CHANNEL(TO.QM1)                         QMID(QM1_2002-03-
04_11.04.19)
```

Cause

The queue manager may have been deleted and then recreated and redefined, or it may have been cold started on z/OS, without first following the procedure in Task 5 ″Removing a Queue Manager from a cluster″. The cluster will function correctly with the older version of the queue manager being ignored, until it ages out of the cluster completely after about 90 days. To remove all trace of the queue manager immediately use the RESET CLUSTER command from a full repository queue manager, to remove the older unwanted queue manager and its queues from the cluster.

```
     2 : reset cluster(DEMO) qmid('QM1_2002-03-04_11.04.19')
action(FORCEREMOVE) queues(yes)
AMQ8559: RESET CLUSTER accepted.
```

Symptom — RESET CLUSTER and REFRESH CLUSTER commands were issued, but the queue manager would not rejoin the cluster.

RESET CLUSTER and REFRESH CLUSTER commands were issued, but the queue manager would not rejoin the cluster.

Cause

A side effect of the RESET and REFRESH commands may be that a channel is stopped, in order that the correct version of the channel runs when the command is completed. Check that the channels between the problem queue manager and the full repositories are running and use the START CHANNEL command if necessary.

Resolving Problems

The following problems can all be resolved using the REFRESH CLUSTER command. It is unlikely that you will need to use this command during normal circumstances. Use it only if you want your queue manager to make a fresh start in a cluster. Issue the REFRESH CLUSTER command from a queue manager to discard all locally held information about a cluster. For example you might use it if you think your full repository is not up-to-date, perhaps because you have accidentally restored an out-of-date backup. The format of the command is:

```
REFRESH CLUSTER(clustername) REPOS(YES/NO)
```

The queue manager from which you issue this command loses all the information in its full repository concerning the named cluster. **It also loses any auto-defined channels that are not in doubt and which are not attached to a full repository queue manager**. The queue manager has to make a cold-start in that cluster. It must reissue all information about itself and renew its requests for updates to other information that it is interested in. (It does this automatically.)

Here are some procedures for recovering clusters.

Problem 1 — Out of date information in a restored cluster.

An image backup of QM1, a partial repository in CLUSTER DEMO has been restored and the cluster information it contains is out of date.

On QM1, issue the command REFRESH CLUSTER(DEMO)

QM1 will remove all information it has about the cluster DEMO, except that relating to the cluster queue managers which are the full repositories in the cluster. Assuming that this information is still correct, QM1 will contact the full repositories. QM1 will then inform them about itself and its queues and recover the information for queues and queue managers that exist elsewhere in the cluster as they are opened.

Problem 2 — cluster DEMO force removed by mistake.

RESET CLUSTER(DEMO) QMNAME(QM1) ACTION(FORCEREMOVE) was issued on a full repository in cluster DEMO by mistake

On QM1, issue the command REFRESH CLUSTER(DEMO)

See solution to problem “Problem 1 — Out of date information in a restored cluster.”

Problem 3 — Possible repository messages deleted.

Messages destined for QM1 were removed from the SYSTEM.CLUSTER.TRANSMIT.QUEUE in other queue managers and they might have been repository messages

On QM1, issue the command REFRESH CLUSTER(DEMO)

See solution to problem "Problem 1 — Out of date information in a restored cluster." on page 115

Problem 4 — 2 full repositories moved at the same time.

Cluster DEMO contains two full repositories, QM1 and QM2. They were both moved to a new location on the network at the same time.

Alter the CONNAME in the CLUSRCVR's and CLUSSDR's to specify the new network addresses.

Alter one of the queue managers (QM1 or QM2) so it is no longer a full repository for any cluster.

On the altered queue manager, issue the command REFRESH CLUSTER(*) REPOS(YES).

Alter the queue manager so it is acting as a full repository.

This problem could have been avoided if, after moving one of the queue managers (for example QM2) to its new network address it was allowed to start its CLUSRCVR, altered with the new address. Having informed the rest of the cluster and the other full repository queue manager (QM1) of the new address of QM2. The other queue manager (QM1) can then be moved to its new network address, restarted and its CLUSRCVR modified to show its new network address. The manually defined CLUSSDR channels should also be modified for the sake of clarity, even though at this stage they are not needed for the correct operation of the cluster.

This procedure forces QM2 to reuse the information from the correct CLUSSDR to re-establish contact with QM1 and then rebuild its knowledge of the cluster. Additionally, having once again made contact with QM1 it will be given its own correct network address based on the CONNAME in its CLUSRCVR definition.

Problem 5 — Unknown state of a cluster.

The state of the cluster is unknown and it is required to completely reset all of the systems in it.

For all full repository queue managers, follow these steps:

1. Alter queue managers that are full repositories so they are no longer full repositories.
2. Resolve any in doubt CLUSSDR channels.
3. Wait for the CLUSSDR channels to become inactive.
4. Stop the CLUSRCVR channels.
5. When all of the CLUSRCVR channels on all of the full repository systems are stopped, issue the command REFRESH CLUSTER(DEMO) REPOS(YES)
6. Alter the queue managers so they are full repositories.
7. Start the CLUSRCVR channels to re-enable them for communication.

Carry out the following steps on the other partial repository queue managers:

1. Resolve any in doubt CLUSSDR channels.
2. Make sure all CLUSSDR channels on the queue manager are stopped or inactive.
3. Issue the command REFRESH CLUSTER(DEMO) REPOS(YES).

Note:

> Under normal conditions the full repositories will exchange information about the queues and queue managers in the cluster. If one full repository is refreshed, the cluster information is recovered from the other. To stop this happening all of the CLUSRCVR channels to full repositories are stopped and the CLUSSDR's should be inactive. When you REFRESH the full repository systems, none of them are able to communicate, so will start from the same cleared state.
>
> As you REFRESH the partial repository systems they rejoin the cluster and rebuild it to the complete set of queue managers and queues.

Part 3. Reference information

Chapter 10. Cluster workload exit call and data structures

This chapter provides reference information for the cluster workload exit and the data structures it uses. This is general-use programming interface information.

You can write cluster workload exits in the following programming languages:
- C
- System/390® assembler (WebSphere MQ for z/OS)

The call is described in:
- "MQ_CLUSTER_WORKLOAD_EXIT - Cluster workload exit" on page 122

The structure data types used by the exit are described in:
- "MQWXP - Cluster workload exit parameter structure" on page 123
- "MQWDR - Cluster workload destination-record structure" on page 131
- "MQWQR - Cluster workload queue-record structure" on page 135
- "MQWCR - Cluster workload cluster-record structure" on page 139

Constants relating to the cluster workload exit are listed with their values in Chapter 12, "Constants for the cluster workload exit", on page 147.

Throughout the following pages queue-manager attributes and queue attributes are shown in full, as defined in the *WebSphere MQ Application Programming Reference* book. The equivalent names that are used in the MQSC commands described in the *WebSphere MQ Script (MQSC) Command Reference* book are shown in Table 4 and Table 5.

Table 4. Queue-manager attributes

Full name	**Name used in MQSC**
ClusterWorkloadData	CLWLDATA
ClusterWorkloadExit	CLWLEXIT
ClusterWorkloadLength	CLWLLEN

Table 5. Queue attributes

Full name	**Name used in MQSC**
DefBind	DEFBIND
DefPersistence	DEFPSIST
DefPriority	DEFPRTY
InhibitPut	PUT
QDesc	DESCR

MQ_CLUSTER_WORKLOAD_EXIT - Cluster workload exit

This call definition describes the parameters that are passed to the cluster workload exit called by the queue manager.

Note: No entry point called MQ_CLUSTER_WORKLOAD_EXIT is actually provided by the queue manager. This is because the name of the cluster workload exit is defined by the `ClusterWorkloadExit` queue-manager attribute.

| This exit is supported in the following environments: AIX, Compaq Tru64 UNIX, Compaq OpenVMS Alpha, Compaq NonStop Kernel, HP-UX, Linux, OS/2, z/OS, OS/400, Solaris, and Windows.

Syntax

MQ_CLUSTER_WORKLOAD_EXIT *(ExitParms)*

Parameters

The MQ_CLUSTER_WORKLOAD_EXIT call has the following parameters.

ExitParms (MQWXP) – input/output
Exit parameter block.

This structure contains information relating to the invocation of the exit. The exit sets information in this structure to indicate how the workload should be managed.

Usage notes

1. The function performed by the cluster workload exit is defined by the provider of the exit. The exit, however, must conform to the rules defined in the associated control block MQWXP.
2. No entry point called MQ_CLUSTER_WORKLOAD_EXIT is actually provided by the queue manager. However, a **typedef** is provided for the name MQ_CLUSTER_WORKLOAD_EXIT in the C programming language, and this can be used to declare the user-written exit, to ensure that the parameters are correct.

C invocation

```
exitname (&ExitParms);
```

Declare the parameters as follows:

```
MQWXP  ExitParms;  /* Exit parameter block */
```

System/390 assembler invocation

```
         CALL EXITNAME,(EXITPARMS)
```

Declare the parameters as follows:

```
EXITPARMS          CMQWXPA             Exit parameter block
```

MQWXP - Cluster workload exit parameter structure

The following table summarizes the fields in the structure.

Table 6. Fields in MQWXP

Field	Description	Page
StrucId	Structure identifier	123
Version	Structure version number	124
ExitId	Type of exit	124
ExitReason	Reason for invoking exit	124
ExitResponse	Response from exit	125
ExitResponse2	Secondary response from exit	125
Feedback	Feedback code	126
ExitUserArea	Exit user area	126
ExitData	Exit data	126
MsgDescPtr	Address of message descriptor (MQMD)	127
MsgBufferPtr	Address of buffer containing some or all of the message data	127
MsgBufferLength	Length of buffer containing message data	127
MsgLength	Length of complete message	127
QName	Name of queue	127
QMgrName	Name of local queue manager	128
DestinationCount	Number of possible destinations	128
DestinationChosen	Destination chosen	128
DestinationArrayPtr	Address of an array of pointers to destination records (MQWDR)	128
QArrayPtr	Address of an array of pointers to queue records (MQWQR)	128
CacheContext	Context information	128
CacheType	Type of cluster cache	128

The MQWXP structure describes the information that is passed to the cluster workload exit.

| This structure is supported in the following environments: AIX, Compaq Tru64
| UNIX, Compaq OpenVMS Alpha, Compaq NonStop Kernel, HP-UX, Linux, OS/2,
| z/OS, OS/400, Solaris, and Windows.

Fields

StrucId (MQCHAR4)
Structure identifier.

The value is:

MQWXP_STRUC_ID
Identifier for cluster workload exit parameter structure.

For the C programming language, the constant MQWXP_STRUC_ID_ARRAY is also defined; this has the same value as MQWXP_STRUC_ID, but is an array of characters instead of a string.

This is an input field to the exit.

Version (MQLONG)
Structure version number.

Possible values are:

MQWXP_VERSION_1
Version-1 cluster workload exit parameter structure.

This version is supported in all environments.

MQWXP_VERSION_2
Version-2 cluster workload exit parameter structure.

This version is supported in the following environments: AIX, HP-UX, Linux, OS/400, Solaris and Windows NT.

The following constant specifies the version number of the current version:

MQWXP_CURRENT_VERSION
Current version of cluster workload exit parameter structure.

This is an input field to the exit.

ExitId (MQLONG)
Type of exit.

This indicates the type of exit being called. The value is:

MQXT_CLUSTER_WORKLOAD_EXIT
Cluster workload exit.

This type of exit is supported in the following environments: AIX, Compaq Tru64 UNIX, Compaq OpenVMS Alpha, Compaq NonStop Kernel, HP-UX, Linux, OS/2, z/OS, OS/400, Solaris, and Windows.

This is an input field to the exit.

ExitReason (MQLONG)
Reason for invoking exit.

This indicates the reason why the exit is being called. Possible values are:

MQXR_INIT
Exit initialization.

This indicates that the exit is being invoked for the first time. It allows the exit to acquire and initialize any resources that it may need (for example: main storage).

MQXR_TERM
Exit termination.

This indicates that the exit is about to be terminated. The exit should free any resources that it may have acquired since it was initialized (for example: main storage).

MQXR_CLWL_OPEN
Called from MQOPEN processing.

MQXR_CLWL_PUT
Called from MQPUT or MQPUT1 processing.

MQXR_CLWL_MOVE
Called from MCA when the channel state has changed.

MQXR_CLWL_REPOS
Called from MQPUT or MQPUT1 processing for a repository-manager PCF message.

MQXR_CLWL_REPOS_MOVE
Called from MCA for a repository-manager PCF message when the channel state has changed.

This is an input field to the exit.

ExitResponse (MQLONG)
Response from exit.

This is set by the exit to indicate whether processing of the message should continue. It must be one of the following:

MQXCC_OK
Continue normally.

This indicates that processing of the message should continue normally. *DestinationChosen* identifies the destination to which the message should be sent.

MQXCC_SUPPRESS_FUNCTION
Suppress function.

This indicates that processing of the message should be discontinued:

- For MQXR_CLWL_OPEN, MQXR_CLWL_PUT, and MQXR_CLWL_REPOS invocations, the MQOPEN, MQPUT, or MQPUT1 call fails with completion code MQCC_FAILED and reason code MQRC_STOPPED_BY_CLUSTER_EXIT.
- For MQXR_CLWL_MOVE and MQXR_CLWL_REPOS_MOVE invocations, the message is placed on the dead-letter queue.

MQXCC_SUPPRESS_EXIT
Suppress exit.

This indicates that processing of the current message should continue normally, but that the exit should not be invoked again until termination of the queue manager. The queue manager processes subsequent messages as if the *ClusterWorkloadExit* queue-manager attribute were blank. *DestinationChosen* identifies the destination to which the current message should be sent.

If any other value is returned by the exit, the queue manager processes the message as if MQXCC_SUPPRESS_FUNCTION had been specified.

This is an output field from the exit.

ExitResponse2 (MQLONG)

| Secondary response from exit.

This is set to zero on entry to the exit. It can be set by the exit to provide further information to the queue manager.

When *ExitReason* has the value MQXR_INIT, the exit can set one of the following values in *ExitResponse2*:

MQXR2_STATIC_CACHE
Exit requires a static cluster cache.

If the exit returns this value, the exit need not use the MQXCLWLN call to navigate the chains of records in the cluster cache, but the cache must be static.

If the exit returns this value and the cluster cache is dynamic, the exit cannot navigate correctly through the records in the cache. In this situation, the queue manager processes the return from the MQXR_INIT call as though the exit had returned MQXCC_SUPPRESS_EXIT in the *ExitResponse* field.

MQXR2_DYNAMIC_CACHE
Exit can operate with either a static or dynamic cache.

If the exit returns this value, the exit must use the MQXCLWLN call to navigate the chains of records in the cluster cache.

If the exit sets neither value, MQXR2_STATIC_CACHE is assumed.

This is an input/output field to the exit.

Feedback (MQLONG)
Reserved.

This is a reserved field. The value is zero.

Reserved (MQLONG)
Reserved.

This is a reserved field. The value is zero.

ExitUserArea (MQBYTE16)
Exit user area.

This is a field that is available for the exit to use. It is initialized to MQXUA_NONE (binary zero) before the first invocation of the exit. Any changes made to this field by the exit are preserved across the invocations of the exit that occur between the MQCONN call and the matching MQDISC call. The field is reset to MQXUA_NONE when the MQDISC call occurs. The first invocation of the exit is indicated by the *ExitReason* field having the value MQXR_INIT.

The following value is defined:

MQXUA_NONE
No user information.

The value is binary zero for the length of the field.

For the C programming language, the constant MQXUA_NONE_ARRAY is also defined; this has the same value as MQXUA_NONE, but is an array of characters instead of a string.

The length of this field is given by MQ_EXIT_USER_AREA_LENGTH. This is an input/output field to the exit.

ExitData (MQCHAR32)
Exit data.

This is set on input to the exit routine to the value of the *ClusterWorkloadData* queue-manager attribute. If no value has been defined for that attribute, this field is all blanks.

The length of this field is given by MQ_EXIT_DATA_LENGTH. This is an input field to the exit.

MsgDescPtr (PMQMD)
Address of message descriptor.

This is the address of a copy of the message descriptor (MQMD) for the message being processed. Any changes made to the message descriptor by the exit are ignored by the queue manager.

No message descriptor is passed to the exit if *ExitReason* has one of the following values:
MQXR_INIT
MQXR_TERM
MQXR_CLWL_OPEN

In these cases, *MsgDescPtr* is the null pointer.

This is an input field to the exit.

MsgBufferPtr (PMQVOID)
Address of buffer containing some or all of the message data.

This is the address of a buffer containing a copy of the first *MsgBufferLength* bytes of the message data. Any changes made to the message data by the exit are ignored by the queue manager.

No message data is passed to the exit when:
- *MsgDescPtr* is the null pointer.
- The message has no data.
- The *ClusterWorkloadLength* queue-manager attribute is zero.

In these cases, *MsgBufferPtr* is the null pointer.

This is an input field to the exit.

MsgBufferLength (MQLONG)
Length of buffer containing message data.

This is the length of the message data passed to the exit. This length is controlled by the *ClusterWorkloadLength* queue-manager attribute, and may be less than the length of the complete message (see *MsgLength*).

This is an input field to the exit.

MsgLength (MQLONG)
Length of complete message.

The length of the message data passed to the exit (*MsgBufferLength*) might be less than the length of the complete message. *MsgLength* is zero if *ExitReason* is MQXR_INIT, MQXR_TERM, or MQXR_CLWL_OPEN.

This is an input field to the exit.

QName (MQCHAR48)
Queue name.

This is the name of the destination queue; this queue is a cluster queue.

The length of this field is given by MQ_Q_NAME_LENGTH. This is an input field to the exit.

QMgrName (MQCHAR48)
Name of local queue manager.

This is the name of the queue manager that has invoked the cluster workload exit.

The length of this field is given by MQ_Q_MGR_NAME_LENGTH. This is an input field to the exit.

DestinationCount (MQLONG)
Number of possible destinations.

This specifies the number of destination records (MQWDR) that describe instances of the destination queue. There is one MQWDR structure for each possible route to each instance of the queue. The MQWDR structures are addressed by an array of pointers (see *DestinationArrayPtr*).

This is an input field to the exit.

DestinationChosen (MQLONG)
Destination chosen.

This is the number of the MQWDR structure that identifies the route and queue instance to which the message should be sent. The value is in the range 1 through *DestinationCount*.

On input to the exit, *DestinationChosen* indicates the route and queue instance that the queue manager has selected. The exit can accept this choice, or choose a different route and queue instance. However, the value returned by the exit must be in the range 1 through *DestinationCount*. If any other value is returned, the queue manager uses the value of *DestinationChosen* on input to the exit.

This is an input/output field to the exit.

DestinationArrayPtr (PPMQWDR)
Address of an array of pointers to destination records.

This is the address of an array of pointers to destination records (MQWDR). There are *DestinationCount* destination records.

This is an input field to the exit.

QArrayPtr (PPMQWQR)
Address of an array of pointers to queue records.

This is the address of an array of pointers to queue records (MQWQR). If queue records are available, there are *DestinationCount* of them. If no queue records are available, *QArrayPtr* is the null pointer.

Note: *QArrayPtr* can be the null pointer even when *DestinationCount* is greater than zero.

This is an input field to the exit.

| *CacheContext* (MQPTR)
| Context information.

| This field is reserved for use by the queue manager. The exit must not alter the
| value of this field.

| This is an input field to the exit.

| *CacheType* (MQLONG)
| Type of cluster cache.

This is the type of the cluster cache. It is one of the following:

MQCLCT_STATIC
Cache is static.

If the cache has this type, the size of the cache is fixed, and cannot grow as the queue manager operates. The MQXCLWLN call need not be used to navigate the records in this type of cache.

MQCLCT_DYNAMIC
Cache is dynamic.

If the cache has this type, the size of the cache can increase in order to accommodate the varying cluster information. The MQXCLWLN call must be used to navigate the records in this type of cache.

This is an input field to the exit.

C declaration

```
typedef struct tagMQWXP {
  MQCHAR4   StrucId;             /* Structure identifier */
  MQLONG    Version;             /* Structure version number */
  MQLONG    ExitId;              /* Type of exit */
  MQLONG    ExitReason;          /* Reason for invoking exit */
  MQLONG    ExitResponse;        /* Response from exit */
  MQLONG    ExitResponse2;       /* Reserved */
  MQLONG    Feedback;            /* Reserved */
  MQLONG    Reserved;            /* Reserved */
  MQBYTE16  ExitUserArea;        /* Exit user area */
  MQCHAR32  ExitData;            /* Exit data */
  PMQMD     MsgDescPtr;          /* Address of message descriptor */
  PMQVOID   MsgBufferPtr;        /* Address of buffer containing some
                                    or all of the message data */
  MQLONG    MsgBufferLength;     /* Length of buffer containing message
                                    data */
  MQLONG    MsgLength;           /* Length of complete message */
  MQCHAR48  QName;               /* Queue name */
  MQCHAR48  QMgrName;            /* Name of local queue manager */
  MQLONG    DestinationCount;    /* Number of possible destinations */
  MQLONG    DestinationChosen;   /* Destination chosen */
  PPMQWDR   DestinationArrayPtr; /* Address of an array of pointers to
                                    destination records */
  PPMQWQR   QArrayPtr;           /* Address of an array of pointers to
                                    queue records */
  MQPTR     CacheContext;        /* Context information */
  MQLONG    CacheType;           /* Type of cluster cache */
} MQWXP;
```

System/390 assembler declaration

```
MQWXP                         DSECT
MQWXP_STRUCID                 DS   CL4      Structure identifier
MQWXP_VERSION                 DS   F        Structure version number
MQWXP_EXITID                  DS   F        Type of exit
MQWXP_EXITREASON              DS   F        Reason for invoking exit
MQWXP_EXITRESPONSE            DS   F        Response from exit
MQWXP_EXITRESPONSE2           DS   F        Reserved
MQWXP_FEEDBACK                DS   F        Reserved
MQWXP_RESERVED                DS   F        Reserved
MQWXP_EXITUSERAREA            DS   XL16     Exit user area
MQWXP_EXITDATA                DS   CL32     Exit data
MQWXP_MSGDESCPTR              DS   F        Address of message
*                                           descriptor
MQWXP_MSGBUFFERPTR            DS   F        Address of buffer containing
*                                           some or all of the message
*                                                data
MQWXP_MSGBUFFERLENGTH          DS   F        Length of buffer containing
*                                            message data
MQWXP_MSGLENGTH                DS   F        Length of complete message
MQWXP_QNAME                    DS   CL48     Queue name
MQWXP_QMGRNAME                 DS   CL48     Name of local queue manager
MQWXP_DESTINATIONCOUNT         DS   F        Number of possible
*                                            destinations
MQWXP_DESTINATIONCHOSEN        DS   F        Destination chosen
MQWXP_DESTINATIONARRAYPTR      DS   F        Address of an array of
*                                            pointers to destination
*                                            records
MQWXP_QARRAYPTR                DS   F        Address of an array of
*                                            pointers to queue records
MQWXP_LENGTH                   EQU  *-MQWXP  Length of structure
                               ORG  MQWXP
MQWXP_AREA                     DS   CL(MQWXP_LENGTH)
```

MQWDR - Cluster workload destination-record structure

The following table summarizes the fields in the structure.

Table 7. Fields in MQWDR

Field	Description	Page
StrucId	Structure identifier	131
Version	Structure version number	131
StrucLength	Length of MQWDR structure	132
QMgrFlags	Queue-manager flags	132
QMgrIdentifier	Queue-manager identifier	132
QMgrName	Queue-manager name	132
ClusterRecOffset	Offset of first cluster record (MQWCR)	132
ChannelState	Channel state	133
ChannelDefOffset	Offset of channel-definition structure (MQCD)	133

The MQWDR structure contains information relating to one of the possible destinations for the message. There is one MQWDR structure for each instance of the destination queue.

| This structure is supported in the following environments: AIX, Compaq Tru64
| UNIX, Compaq OpenVMS Alpha, Compaq NonStop Kernel, HP-UX, Linux, OS/2,
| z/OS, OS/400, Solaris, and Windows.

Fields

StrucId (MQCHAR4)
Structure identifier.

The value is:

MQWDR_STRUC_ID
Identifier for cluster workload destination record.

For the C programming language, the constant MQWDR_STRUC_ID_ARRAY is also defined; this has the same value as MQWDR_STRUC_ID, but is an array of characters instead of a string.

This is an input field to the exit.

Version (MQLONG)
Structure version number.

The value is:

MQWDR_VERSION_1
Version-1 cluster workload destination record.

The following constant specifies the version number of the current version:

MQWDR_CURRENT_VERSION
Current version of cluster workload destination record.

This is an input field to the exit.

MQWDR structure

`StrucLength` (MQLONG)
Length of MQWDR structure.

The value is:

MQWDR_LENGTH_1
Length of version-1 cluster workload destination record.

The following constant specifies the length of the current version:

MQWDR_CURRENT_LENGTH
Length of current version of cluster workload destination record.

This is an input field to the exit.

`QMgrFlags` (MQLONG)
Queue-manager flags

These are bit flags that indicate various properties of the queue manager that hosts the instance of the destination queue described by this MQWDR structure. The following flags are defined:

MQQMF_REPOSITORY_Q_MGR
Destination is a full repository queue manager.

MQQMF_CLUSSDR_USER_DEFINED
Cluster sender channel was defined manually.

MQQMF_CLUSSDR_AUTO_DEFINED
Cluster sender channel was defined automatically.

MQQMF_AVAILABLE
Destination queue manager is available to receive messages.

Note: Other flags in the field might be set by the queue manager for internal purposes.

This is an input field to the exit.

`QMgrIdentifier` (MQCHAR48)
Queue-manager identifier.

This is a string that acts as a unique identifier for the queue manager. It is generated by the queue manager.

The length of this field is given by MQ_Q_MGR_IDENTIFIER_LENGTH. This is an input field to the exit.

`QMgrName` (MQCHAR48)
Queue-manager name.

This is the name of the queue manager that hosts the instance of the destination queue described by this MQWDR structure. This can be the name of the local queue manager.

The length of this field is given by MQ_Q_MGR_NAME_LENGTH. This is an input field to the exit.

`ClusterRecOffset` (MQLONG)
Offset of first cluster record.

This is the offset of the first MQWCR structure that belongs to this MQWDR structure. The offset is measured in bytes from the start of the MQWDR structure.

This is an input field to the exit.

ChannelState (MQLONG)
Channel state.

This indicates the state of the channel that links the local queue manager to the queue manager identified by this MQWDR structure. The following values are possible:

MQCHS_INACTIVE
Channel is not active.

MQCHS_BINDING
Channel is negotiating with the partner.

MQCHS_STARTING
Channel is waiting to become active.

MQCHS_RUNNING
Channel is transferring or waiting for messages.

MQCHS_STOPPING
Channel is stopping.

MQCHS_RETRYING
Channel is reattempting to establish connection.

MQCHS_STOPPED
Channel has stopped.

MQCHS_REQUESTING
Requester channel is requesting connection.

MQCHS_PAUSED
Channel has paused.

MQCHS_INITIALIZING
Channel is initializing.

This is an input field to the exit.

ChannelDefOffset (MQLONG)
Offset of channel definition structure.

This is the offset of the channel definition (MQCD) for the channel that links the local queue manager to the queue manager identified by this MQWDR structure. The offset is measured in bytes from the start of the MQWDR structure.

This is an input field to the exit.

C declaration

```
typedef struct tagMQWDR {
  MQCHAR4   StrucId;          /* Structure identifier */
  MQLONG    Version;          /* Structure version number */
  MQLONG    StrucLength;      /* Length of MQWDR structure */
  MQLONG    QMgrFlags;        /* Queue-manager flags */
  MQCHAR48  QMgrIdentifier;   /* Queue-manager identifier */
  MQCHAR48  QMgrName;         /* Queue-manager name */
  MQLONG    ClusterRecOffset; /* Offset of first cluster record */
  MQLONG    ChannelState;     /* Channel state */
  MQLONG    ChannelDefOffset; /* Offset of channel definition
                                 structure */
} MQWDR;
```

System/390 assembler declaration

```
MQWDR                         DSECT
MQWDR_STRUCID                 DS   CL4      Structure identifier
MQWDR_VERSION                 DS   F        Structure version number
MQWDR_STRUCLENGTH             DS   F        Length of MQWDR structure
MQWDR_QMGRFLAGS               DS   F        Queue-manager flags
MQWDR_QMGRIDENTIFIER          DS   CL48     Queue-manager identifier
MQWDR_QMGRNAME                DS   CL48     Queue-manager name
MQWDR_CLUSTERRECOFFSET        DS   F        Offset of first cluster
*                                           record
MQWDR_CHANNELSTATE            DS   F        Channel state
MQWDR_CHANNELDEFOFFSET        DS   F        Offset of channel definition
*                                           structure
MQWDR_LENGTH                  EQU  *-MQWDR  Length of structure
                              ORG  MQWDR
MQWDR_AREA                    DS   CL(MQWDR_LENGTH)
```

MQWQR - Cluster workload queue-record structure

The following table summarizes the fields in the structure.

Table 8. Fields in MQWQR

Field	Description	Page
StrucId	Structure identifier	135
Version	Structure version number	135
StrucLength	Length of MQWQR structure	136
QFlags	Queue flags	136
QName	Queue name	136
QMgrIdentifier	Queue-manager identifier	136
ClusterRecOffset	Offset of first cluster record (MQWCR)	136
QType	Queue type	136
QDesc	Queue description	137
DefBind	Default binding	137
DefPersistence	Default message persistence	137
DefPriority	Default message priority	137
InhibitPut	Whether put operations on the queue are allowed	137

The MQWQR structure contains information relating to one of the possible destinations for the message. There is one MQWQR structure for each instance of the destination queue.

| This structure is supported in the following environments: AIX, Compaq Tru64 UNIX, Compaq OpenVMS Alpha, Compaq NonStop Kernel, HP-UX, Linux, OS/2, z/OS, OS/400, Solaris, and Windows.

Fields

StrucId (MQCHAR4)
Structure identifier.

The value is:

MQWQR_STRUC_ID
Identifier for cluster workload queue record.

For the C programming language, the constant MQWQR_STRUC_ID_ARRAY is also defined; this has the same value as MQWQR_STRUC_ID, but is an array of characters instead of a string.

This is an input field to the exit.

Version (MQLONG)
Structure version number.

The value is:

MQWQR_VERSION_1
Version-1 cluster workload queue record.

The following constant specifies the version number of the current version:

MQWQR_CURRENT_VERSION
Current version of cluster workload queue record.

This is an input field to the exit.

StrucLength (MQLONG)
Length of MQWQR structure.

The value is:

MQWQR_LENGTH_1
Length of version-1 cluster workload queue record.

The following constant specifies the length of the current version:

MQWQR_CURRENT_LENGTH
Length of current version of cluster workload queue record.

This is an input field to the exit.

QFlags (MQLONG)
Queue flags.

These are bit flags that indicate various properties of the queue. The following flag is defined:

MQQF_LOCAL_Q
Destination is a local queue.

Note: Other flags in the field might be set by the queue manager for internal purposes.

This is an input field to the exit.

QName (MQCHAR48)
Queue name.

The length of this field is given by MQ_Q_NAME_LENGTH. This is an input field to the exit.

QMgrIdentifier (MQCHAR48)
Queue-manager identifier.

This is a string that acts as a unique identifier for the queue manager that hosts the instance of the queue described by this MQWQR structure. The identifier is generated by the queue manager.

The length of this field is given by MQ_Q_MGR_IDENTIFIER_LENGTH. This is an input field to the exit.

ClusterRecOffset (MQLONG)
Offset of first cluster record.

This is the offset of the first MQWCR structure that belongs to this MQWQR structure. The offset is measured in bytes from the start of the MQWQR structure.

This is an input field to the exit.

QType (MQLONG)
Queue type.

The following values are possible:

MQCQT_LOCAL_Q
Local queue.

MQCQT_ALIAS_Q
Alias queue.

MQCQT_REMOTE_Q
Remote queue.

MQCQT_Q_MGR_ALIAS
Queue-manager alias.

This is an input field to the exit.

QDesc (MQCHAR64)
Queue description.

This is the value of the *QDesc* queue attribute as defined on the queue manager that hosts the instance of the destination queue described by this MQWQR structure.

The length of this field is given by MQ_Q_DESC_LENGTH. This is an input field to the exit.

DefBind (MQLONG)
Default binding.

This is the value of the *DefBind* queue attribute as defined on the queue manager that hosts the instance of the destination queue described by this MQWQR structure. The following values are possible:

MQBND_BIND_ON_OPEN
Binding fixed by MQOPEN call.

MQBND_BIND_NOT_FIXED
Binding not fixed.

This is an input field to the exit.

DefPersistence (MQLONG)
Default message persistence.

This is the value of the *DefPersistence* queue attribute as defined on the queue manager that hosts the instance of the destination queue described by this MQWQR structure. The following values are possible:

MQPER_PERSISTENT
Message is persistent.

MQPER_NOT_PERSISTENT
Message is not persistent.

This is an input field to the exit.

DefPriority (MQLONG)
Default message priority.

This is the value of the *DefPriority* queue attribute as defined on the queue manager that hosts the instance of the destination queue described by this MQWQR structure. Priorities are in the range zero (lowest) through *MaxPriority* (highest), where *MaxPriority* is the queue-manager attribute of the queue manager that hosts this instance of the destination queue.

This is an input field to the exit.

InhibitPut (MQLONG)
Whether put operations on the queue are allowed.

This is the value of the *InhibitPut* queue attribute as defined on the queue manager that hosts the instance of the destination queue described by this MQWQR structure. The following values are possible:

MQQA_PUT_INHIBITED
Put operations are inhibited.

MQQA_PUT_ALLOWED
Put operations are allowed.

This is an input field to the exit.

C declaration

```
typedef struct tagMQWQR {
  MQCHAR4   StrucId;          /* Structure identifier */
  MQLONG    Version;          /* Structure version number */
  MQLONG    StrucLength;      /* Length of MQWQR structure */
  MQLONG    QFlags;           /* Queue flags */
  MQCHAR48  QName;            /* Queue name */
  MQCHAR48  QMgrIdentifier;   /* Queue-manager identifier */
  MQLONG    ClusterRecOffset; /* Offset of first cluster record */
  MQLONG    QType;            /* Queue type */
  MQCHAR64  QDesc;            /* Queue description */
  MQLONG    DefBind;          /* Default binding */
  MQLONG    DefPersistence;   /* Default message persistence */
  MQLONG    DefPriority;      /* Default message priority */
  MQLONG    InhibitPut;       /* Whether put operations on the queue
                                 are allowed */
} MQWQR;
```

System/390 assembler declaration

```
MQWQR                         DSECT
MQWQR_STRUCID                 DS   CL4      Structure identifier
MQWQR_VERSION                 DS   F        Structure version number
MQWQR_STRUCLENGTH             DS   F        Length of MQWQR structure
MQWQR_QFLAGS                  DS   F        Queue flags
MQWQR_QNAME                   DS   CL48     Queue name
MQWQR_QMGRIDENTIFIER          DS   CL48     Queue-manager identifier
MQWQR_CLUSTERRECOFFSET        DS   F        Offset of first cluster
*                                           record
MQWQR_QTYPE                   DS   F        Queue type
MQWQR_QDESC                   DS   CL64     Queue description
MQWQR_DEFBIND                 DS   F        Default binding
MQWQR_DEFPERSISTENCE          DS   F        Default message persistence
MQWQR_DEFPRIORITY             DS   F        Default message priority
MQWQR_INHIBITPUT              DS   F        Whether put operations on
*                                           the queue are allowed
MQWQR_LENGTH                  EQU  *-MQWQR  Length of structure
                              ORG  MQWQR
MQWQR_AREA                    DS   CL(MQWQR_LENGTH)
```

MQWCR - Cluster workload cluster-record structure

The following table summarizes the fields in the structure.

Table 9. Fields in MQWCR

Field	Description	Page
`ClusterName`	Name of cluster	139
`ClusterRecOffset`	Offset of next cluster record (MQWCR)	139
`ClusterFlags`	Cluster flags	139

The MQWCR structure contains information relating to a cluster to which an instance of the destination queue belongs. There is one MQWCR for each such cluster.

This structure is supported in the following environments: AIX, Compaq Tru64 UNIX, Compaq OpenVMS Alpha, Compaq NonStop Kernel, HP-UX, Linux, OS/2, z/OS, OS/400, Solaris, and Windows.

Fields

`ClusterName` (MQCHAR48)
Cluster name.

This is the name of a cluster to which the instance of the destination queue that owns this MQWCR structure belongs. The destination queue instance is described by an MQWDR structure.

The length of this field is given by MQ_CLUSTER_NAME_LENGTH. This is an input field to the exit.

`ClusterRecOffset` (MQLONG)
Offset of next cluster record.

This is the offset of the next MQWCR structure. If there are no more MQWCR structures, `ClusterRecOffset` is zero. The offset is measured in bytes from the start of the MQWCR structure.

This is an input field to the exit.

`ClusterFlags` (MQLONG)
Cluster flags.

These are bit flags that indicate various properties of the queue manager identified by this MQWCR structure. The following flags are defined:

MQQMF_REPOSITORY_Q_MGR
Destination is a full repository queue manager.

MQQMF_CLUSSDR_USER_DEFINED
Cluster sender channel was defined manually.

MQQMF_CLUSSDR_AUTO_DEFINED
Cluster sender channel was defined automatically.

MQQMF_AVAILABLE
Destination queue manager is available to receive messages.

Note: Other flags in the field might be set by the queue manager for internal purposes.

This is an input field to the exit.

C declaration

```
typedef struct tagMQWCR {
  MQCHAR48  ClusterName;       /* Cluster name */
  MQLONG    ClusterRecOffset;  /* Offset of next cluster record */
  MQLONG    ClusterFlags;      /* Cluster flags */
} MQWCR;
```

System/390 assembler declaration

```
MQWCR                          DSECT
MQWCR_CLUSTERNAME              DS   CL48     Cluster name
MQWCR_CLUSTERRECOFFSET         DS   F        Offset of next cluster
*                                            record
MQWCR_CLUSTERFLAGS             DS   F        Cluster flags
MQWCR_LENGTH                   EQU  *-MQWCR  Length of structure
                               ORG  MQWCR
MQWCR_AREA                     DS   CL(MQWCR_LENGTH)
```

MQXCLWLN - Navigate Cluster workload records

The MQXCLWLN call is used to navigate through the chains of MQWDR, MQWQR, and MQWCR records stored in the cluster cache. The cluster cache is an area of main storage used to store information relating to the cluster.

This call is supported in the following environments: AIX, HP-UX, Linux, iSeries, Solaris, and Windows

Syntax

MQXCLWLN *(`ExitParms`, `CurrentRecord`, `NextOffset`, `NextRecord`, `Compcode`, `Reason`)*

Parameters

The MQ_CLUSTER_WORKLOAD_EXIT call has the following parameters.

`ExitParms` (MQWXP) – input/output
Exit parameter block.

This structure contains information relating to the invocation of the exit. The exit sets information in this structure to indicate how the workload should be managed.

`CurrentRecord` (MQPTR) – input
Address of current record.

This structure contains information relating to the address of the record currently being examined by the exit. The record must be one of the following types:

- Cluster workload destination record (MQWDR)
- Cluster workload queue record (MQWQR)
- Cluster workload cluster record (MQWCR)

`NextOffset` (MQLONG) – input
Offset of next record.

This structure contains information relating to the offset of the next record or structure. `NextOffset` is the value of the appropriate offset field in the current record, and must be one of the following fields:

- `ChannelDefOffset` field in MQWDR
- `ClusterRecOffset` field in MQWDR
- `ClusterRecOffset` field in MQWQR
- `ClusterRecOffset` field in MQWCR

`NextRecord` (MQPTR) – output
Address of next record or structure.

This structure contains information relating to the address of the next record or structure. If *CurrentRecord* is the address of an MQWDR, and *NextOffset* is the value of the *ChannelDefOffset* field, *NextRecord* is the address of the channel definition structure (MQCD).

If there is no next record or structure, the queue manager sets *NextRecord* to the null pointer, and the call returns completion code MQCC_WARNING and reason code MQRC_NO_RECORD_AVAILABLE.

CompCode (MQLONG) – output

Completion code.

It is one of the following:

MQCC_OK
Successful completion.

MQCC_WARNING
Warning (partial completion).

MQCC_FAILED
Call failed.

Reason (MQLONG) – output

Reason code qualifying *CompCode*

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000) No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_NO_RECORD_AVAILABLE
(2359, X'0937') No record available

If *CompCode* is MQCC_FAILED:

MQRC_CURRENT_RECORD_ERROR
(2357, X'0935') Current-record parameter not valid.

MQRC_ENVIRONMENT_ERROR
(2012, X'07DC') Call not valid in environment.

MQRC_NEXT_OFFSET_ERROR
(2358, X'0936') Next-offset parameter not valid.

MQRC_NEXT_RECORD_ERROR
(2361, X'0939') NextRecord parameter not valid.

MQRC_WXP_ERROR
(2356, X'0934') Workload exit parameter structure not valid.

For more information on these reason codes, see the *WebSphere MQ Application Programming Reference* manual.

Usage notes

1. If the cluster cache is dynamic, the MQXCLWLN call must be used to navigate through the records; the exit will terminate abnormally if simple pointer-and-offset arithmetic is used to navigate through the records.

   If the cluster cache is static, MQXCLWLN need not be used to navigate through the records. However, it is recommended that MQXCLWLN be used even when the cache is static, as this allows migration to a dynamic cache without needing to change the workload exit.

   For more information on dynamic and static cluster cache's see 128

C invocation

```
MQXCLWLN (&ExitParms, CurrentRecord, NextOffset, &NextRecord, &CompCode, &Reason) ;
```

Declare the parameters as follows:

```
typedef struct tagMQXCLWLN {
  MQWXP     ExitParms;           /* Exit parameter block */
  MQPTR     CurrentRecord;       /* Address of current record*/
  MQLONG    NextOffset;          /* Offset of next record */
  MQPTR     NextRecord;          /* Address of next record or structure */
  MQLONG    CompCode;            /* Completion code */
  MQLONG    Reason;              /* Reason code qualifying CompCode */
```

Chapter 11. Using your own cluster workload exits

You may write your own cluster workload exit and override the WebSphere MQ default behaviour. On non-z/OS platforms, using CLWLMode=FAST each operating system process loads its own copy of the exit, and different connections to the queue manager can cause different copies of the exit to be invoked. When the exit is run in the default safe mode, CLWLMode=SAFE, a single copy of the exit runs in its own separate process. For more information on how to set CLWLMode see the*WebSphere MQ System Administration Guide*. Below is the round robin algorithm which WebSphere MQ uses by default.

Algorithm: The steps in choosing a destination for a message:

1. If a queue name is specified, eliminate queues that are not PUT enabled.

   Eliminate remote instances of queues that do not share a cluster with the local queue manager.

   Eliminate remote CLUSRCVR channels that are not in the same cluster as the queue.
2. If a queue manager name is specified, eliminate queue manager alias' that are not PUT enabled.

   Eliminate remote CLUSRCVR channels that are not in the same cluster as the local queue manager.
3. If the result above contains the local instance of the queue, choose it.
4. If the message is a cluster PCF message, eliminate any queue manager you have already sent a publication or subscription to.
5. If only remote instances of a queue remains, choose *Resumed* queue managers over *Suspended* ones.
6. If more than one remote instance of a queue remains, include all MQCHS_INACTIVE and MQCHS_RUNNING channels.
7. If less than one remote instance of a queue remains, include all MQCHS_BINDING, MQCHS_INITIALIZING, MQCHS_STARTING, and MQCHS_STOPPING channels.
8. If less than one remote instance of a queue remains, include all MQCHS_RETRYING channels.
9. If less than one remote instance of a queue remains, include all MQCHS_REQUESTING, MQCHS_PAUSED and MQCHS_STOPPED channels.
10. If more than one remote instance of a queue remains and the message is a cluster PCF message, choose locally defined CLUSSDR channels.
11. If more than one remote instance of a queue remains to any queue manager, choose channels with the highest NETPRTY to each queue manager.
12. If more than one remote instance of a queue remains, choose the least recently used channel.

Using your own cluster workload exits.

Chapter 12. Constants for the cluster workload exit

This chapter specifies the values of the named constants that apply to the cluster workload exit. This information is general-use programming interface information.

The constants are grouped according to the parameter or field to which they relate. The names of the constants in a group begin with a common prefix of the form `MQ`*`xxxx`*`_`, where *`xxxx`* represents a string of 0 through 4 characters that indicates the parameter or field to which the values relate. The constants are ordered alphabetically by this prefix.

Notes:

1. For constants with numeric values, the values are shown in both decimal and hexadecimal forms.
2. Hexadecimal values are represented using the notation `X'hhhh'`, where each `h` denotes a single hexadecimal digit.
3. Character values are shown delimited by single quotation marks; the quotation marks are not part of the value.
4. Blanks in character values are represented by one or more occurrences of the symbol ƀ.

List of constants

The following sections list the named constants that are mentioned in this book, and show their values.

MQ_* (Lengths of character string and byte fields)

MQ_CF_STRUC_NAME_LENGTH	12	X'0000000C'
MQ_CLUSTER_NAME_LENGTH	48	X'00000030'
MQ_EXIT_DATA_LENGTH	32	X'00000020'
MQ_EXIT_USER_AREA_LENGTH	16	X'00000010'
MQ_Q_DESC_LENGTH	64	X'00000040'
MQ_Q_MGR_IDENTIFIER_LENGTH	48	X'00000030'
MQ_Q_MGR_NAME_LENGTH	48	X'00000030'
MQ_Q_NAME_LENGTH	48	X'00000030'
MQ_QSG_NAME_LENGTH	4	X'00000004'

MQBND_* (Binding)

See the *`DefBind`* field described in "MQWQR - Cluster workload queue-record structure" on page 135.

MQBND_BIND_ON_OPEN	0	X'00000000'
MQBND_BIND_NOT_FIXED	1	X'00000001'

Constants

MQCHS_* (Channel status)

See the *ChannelState* field described in “MQWDR - Cluster workload destination-record structure” on page 131.

MQCHS_INACTIVE	0	X'00000000'
MQCHS_BINDING	1	X'00000001'
MQCHS_INITIALIZING	13	X'0000000D'
MQCHS_STARTING	2	X'00000002'
MQCHS_RUNNING	3	X'00000003'
MQCHS_STOPPING	4	X'00000004'
MQCHS_RETRYING	5	X'00000005'
MQCHS_STOPPED	6	X'00000006'
MQCHS_REQUESTING	7	X'00000007'
MQCHS_PAUSED	8	X'00000008'

MQCQT_* (Cluster queue type)

See the *QType* field described in “MQWQR - Cluster workload queue-record structure” on page 135.

MQCQT_LOCAL_Q	1	X'00000001'
MQCQT_ALIAS_Q	2	X'00000002'
MQCQT_REMOTE_Q	3	X'00000003'
MQCQT_Q_MGR_ALIAS	4	X'00000004'

MQPER_* (Persistence)

See the *DefPersistence* field described in “MQWQR - Cluster workload queue-record structure” on page 135.

MQPER_NOT_PERSISTENT	0	X'00000000'
MQPER_PERSISTENT	1	X'00000001'

MQQA_* (Inhibit put)

See the *InhibitPut* field described in “MQWQR - Cluster workload queue-record structure” on page 135.

MQQA_PUT_ALLOWED	0	X'00000000'
MQQA_PUT_INHIBITED	1	X'00000001'

MQQF_* (Queue flags)

See the *QFlags* field described in “MQWQR - Cluster workload queue-record structure” on page 135.

MQQF_LOCAL_Q	1	X'00000001'

MQQMF_* (Queue-manager flags)

See the *QMgrFlags* field described in “MQWDR - Cluster workload destination-record structure” on page 131.

MQQMF_CLUSSDR_AUTO_DEFINED	16	X'00000010'
MQQMF_REPOSITORY_Q_MGR	2	X'00000002'
MQQMF_AVAILABLE	32	X'00000020'
MQQMF_CLUSSDR_USER_DEFINED	8	X'00000008'

MQWDR_* (Cluster workload exit destination-record length)

See the *StrucLength* field described in “MQWDR - Cluster workload destination-record structure” on page 131.

MQWDR_LENGTH_1	124	X'0000007C'
MQWDR_CURRENT_LENGTH	124	X'0000007C'

MQWDR_* (Cluster workload exit destination-record structure identifier)

See the *StrucId* field described in “MQWDR - Cluster workload destination-record structure” on page 131.

MQWDR_STRUC_ID	`'WDRƀ'`

For the C programming language, the following array version is also defined:

MQWDR_STRUC_ID_ARRAY	`'W','D','R','ƀ'`

MQWDR_* (Cluster workload exit destination-record version)

See the *Version* field described in “MQWDR - Cluster workload destination-record structure” on page 131.

MQWDR_VERSION_1	1	X'00000001'
MQWDR_CURRENT_VERSION	1	X'00000001'

MQWQR_* (Cluster workload exit queue-record length)

See the *StrucLength* field described in “MQWQR - Cluster workload queue-record structure” on page 135.

MQWQR_LENGTH_1	200	X'000000C8'
MQWQR_CURRENT_LENGTH	200	X'000000C8'

MQWQR_* (Cluster workload exit queue-record structure identifier)

See the *StrucId* field described in “MQWQR - Cluster workload queue-record structure” on page 135.

MQWQR_STRUC_ID	`'WQRƀ'`

For the C programming language, the following array version is also defined:

MQWQR_STRUC_ID_ARRAY	`'W','Q','R','ƀ'`

MQWQR_* (Cluster workload exit queue-record version)

See the *Version* field described in “MQWQR - Cluster workload queue-record structure” on page 135.

MQWQR_VERSION_1	1	X'00000001'
MQWQR_CURRENT_VERSION	1	X'00000001'

MQWXP_* (Cluster workload exit structure identifier)

See the *StrucId* field described in “MQWXP - Cluster workload exit parameter structure” on page 123.

MQWXP_STRUC_ID	`'WXPƀ'`

For the C programming language, the following array version is also defined:

MQWXP_STRUC_ID_ARRAY	`'W','X','P','ƀ'`

MQWXP_* (Cluster workload exit structure version)

See the *Version* field described in “MQWXP - Cluster workload exit parameter structure” on page 123.

MQWXP_VERSION_1	1	X'00000001'
MQWXP_CURRENT_VERSION	1	X'00000001'

MQXCC_* (Exit response)

See the *ExitResponse* field described in “MQWXP - Cluster workload exit parameter structure” on page 123.

MQXCC_SUPPRESS_FUNCTION	-1	X'FFFFFFFF'
MQXCC_SUPPRESS_EXIT	-5	X'FFFFFFFB'
MQXCC_OK	0	X'00000000'

MQXR_* (Exit reason)

See the *ExitReason* field described in "MQWXP - Cluster workload exit parameter structure" on page 123.

MQXR_INIT	11	X'0000000B'
MQXR_TERM	12	X'0000000C'
MQXR_CLWL_OPEN	20	X'00000014'
MQXR_CLWL_PUT	21	X'00000015'
MQXR_CLWL_MOVE	22	X'00000016'
MQXR_CLWL_REPOS	23	X'00000017'
MQXR_CLWL_REPOS_MOVE	24	X'00000018'

MQXT_* (Exit identifier)

See the *ExitId* field described in "MQWXP - Cluster workload exit parameter structure" on page 123.

MQXT_CLUSTER_WORKLOAD_EXIT	20	X'00000014'

MQXUA_* (Exit user area)

See the *ExitUserArea* field described in "MQWXP - Cluster workload exit parameter structure" on page 123.

MQXUA_NONE	`X'00...00' (16 nulls)`

For the C programming language, the following array version is also defined:

MQXUA_NONE_ARRAY	`'\0','\0',...'\0','\0'`

Part 4. Appendixes

Appendix. Notices

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

IBM may have patents or pending patent applications covering subject matter described in this information. The furnishing of this information does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:
IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

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

Notices

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

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

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

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

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.

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. 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 IBM's application programming interfaces.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

AIX
CICS
DB2
IBM
IBMLink
iSeries
MQSeries
OS/2
OS/390
OS/400
RACF
SP2
System/390
VTAM
WebSphere
z/OS
zSeries

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

Index

Sending your comments to IBM

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book.

Please limit your comments to the information in this book and the way in which the information is presented.

To make comments about the functions of IBM products or systems, talk to your IBM representative or to your IBM authorized remarketer.

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

You can send your comments to IBM in any of the following ways:

- By mail, to this address:

  User Technologies Department (MP095)
  IBM United Kingdom Laboratories
  Hursley Park
  WINCHESTER,
  Hampshire
  SO21 2JN
  United Kingdom
- By fax:
  - From outside the U.K., after your international access code use 44–1962–816151
  - From within the U.K., use 01962–816151
- Electronically, use the appropriate network ID:
  - IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
  - IBMLink™: HURSLEY(IDRCF)
  - Internet: idrcf@hursley.ibm.com

Whichever method you use, ensure that you include:

- The publication title and order number
- The topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.

Printed in U.S.A.

SC34-6061-02