MQSeries® for OS/390®

Concepts and Planning Guide

Version 5 Release 2

GC34-5650-00

Note!

Before using this information and the product it supports, be sure to read the general information under "Appendix C. Notices" on page 161.

First edition (November 2000)

This edition applies to MQSeries for OS/390 Version 5 Release 2, and to all subsequent releases and modifications until otherwise indicated in new editions.

This book is based on parts of the *MQSeries for OS/390 System Management Guide*, SC34-5374-01.

Contents

Figures

Tables

About this book

This book describes the concepts of MQSeries® for OS/390®; it does not describe the concepts of MQSeries messaging and queuing. If you are unfamiliar with these concepts, you should read *MQSeries: An Introduction to Messaging and Queuing*. It also describes how to plan your MQSeries for OS/390 systems.

This book is based on parts of the *MQSeries for OS/390 Version 2.1 System Management Guide* and the *MQSeries Planning Guide*.

The *System Management Guide* has been replaced by the following three books:

- *MQSeries for OS/390 Concepts and Planning Guide*. (This book.)
- *MQSeries for OS/390 System Setup Guide*. This book describes the tasks that you have to perform to customize MQSeries after you have installed it. It also describes how to monitor system use and performance, and how to set up security.
- *MQSeries for OS/390 System Administration Guide*. This book describes how to operate MQSeries, how to perform recovery and restart tasks, and how to use the panel interface and utilities supplied with MQSeries.

Who this book is for

This book is for:

- Planners of OS/390 systems that will use MQSeries message queuing techniques.
- System Programmers who have to install, customize, and operate MQSeries for OS/390.

What you need to know to understand this book

This book assumes you are familiar with the basic concepts of:

- CICS®
- DB2® (if you are going to use queue-sharing groups)
- IMS™
- MQSeries
- OS/390
- The OS/390 Coupling Facility (if you are going to use queue-sharing groups)

Conventions used in this book

- Throughout this book, the term *object* refers to any MQSeries queue manager, queue, namelist, channel, storage class, or process.
- The examples in this book are taken from a queue manager with a command prefix string (CPF) of +CSQ1. The commands are shown in UPPERCASE.
- CICS means both CICS Transaction Server for OS/390 and CICS for MVS/ESA™ unless otherwise stated. IMS means IMS/ESA® unless otherwise stated.
- Throughout this book, the default value `thlqual` is used to indicate the target library high-level qualifier for MQSeries data sets in your installation.
- Throughout this book, the term *distributed queuing* refers to the distributed queuing feature (also known as the "non-CICS mover"). The term *distributed*

queuing using CICS ISC is used to refer to the optional CICS distributed queuing feature (also known as the “CICS mover”).

What’s new for this release

If you are already familiar with previous versions of MQSeries for OS/390, “What’s new for this release” on page 149 summarizes the new functions that have been added to for Version 5.2 and explains where to find more information about them.

Part 1. Introduction

Chapter 1. Introduction

This book describes things that you need to know about MQSeries for OS/390 before you can install and use it on your OS/390 system. It explains the concepts, and gives you information to help you plan your MQSeries subsystems. It assumes that you already know about MQSeries messages and queues. If you need to find out about MQSeries on all platforms, see *MQSeries: An Introduction to Messaging and Queuing*.

This chapter introduces the concepts you need to understand, and directs you to more detailed explanations later in the book. It contains the following sections:

- "What is a queue manager?"
- "What is a channel initiator?" on page 7

What is a queue manager?

Before you can let your application programs use MQSeries on your OS/390 system, you must install the MQSeries for OS/390 product and start a queue manager. The queue manager owns and manages the set of resources that are used by MQSeries. These resources include:

- Page sets that hold the MQSeries object definitions and message data
- Logs that are used to recover messages in the event of queue manager failure
- Processor storage
- Connections through which different application environments (CICS, IMS, and Batch) can access the MQSeries API
- The MQSeries channel initiator, which allows communication between MQSeries on your OS/390 system and other systems

Figure 1 on page 4 illustrates a queue manager, showing connections to different application environments, and the channel initiator.

The queue manager subsystem

On OS/390, MQSeries runs as an OS/390 subsystem. The subsystem has a name (the queue manager name) and applications can connect to it using this name. The subsystem is started by executing a JCL procedure that specifies the OS/390 data sets that contain information about the logs, and that hold object definitions and message data (the page sets).

Any changes to the objects are logged. At restart, the objects are recovered to their state at shutdown.

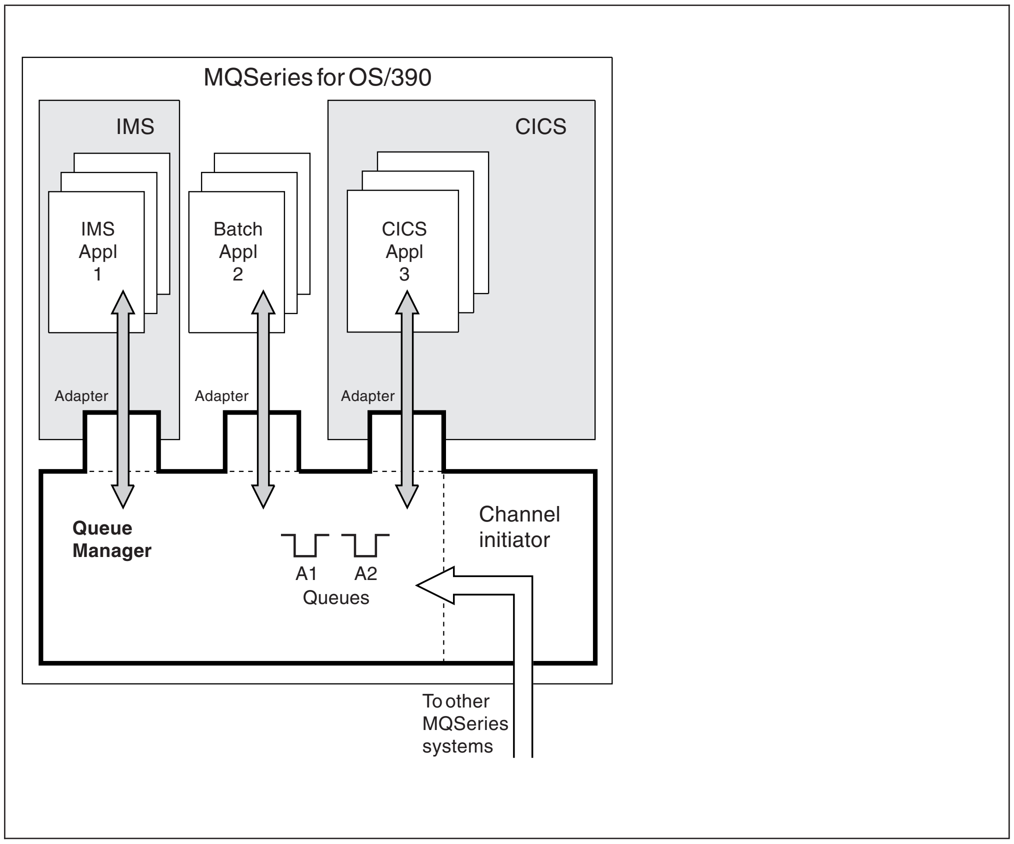


Figure 1. Overview of MQSeries for OS/390

Shared queues

In MQSeries for OS/390 Version 5.2, the concepts of a *queue-sharing group* and a *shared queue* are introduced. A queue-sharing group consists of a number of queue managers, running within a single OS/390 sysplex, that are able to access the same MQSeries object definitions and message data concurrently. Within a queue-sharing group, the shareable object definitions are stored in a shared DB2 database, and the messages are held inside one or more Coupling Facilities. The shared DB2 database and the Coupling Facility structures are resources that are owned by several queue managers.

Shared queues are discussed in "Chapter 2. Shared queues and queue-sharing groups" on page 11.

Page sets and buffer pools

When a message is put on to a queue, the queue manager stores the data on a page set in such a way that it can be retrieved when a subsequent operation attempts to get a message from the same queue. If the message is removed from the queue, space in the page set that holds the data is subsequently freed for reuse. As the number of messages held on a queue increases so the amount of space in the page set holding message data increases, and as the number of messages on a queue reduces the page set space used is reduced.

To reduce the overhead of writing data to and reading data from the page sets, the queue manager buffers the updates into processor storage. The amount of storage used to buffer the page set access is controlled through MQSeries objects called buffer pools.

Page sets and buffer pools are discussed in "Chapter 3. Storage management" on page 27.

Logging

Any changes to objects held on page sets and operations on persistent messages are recorded as log records. These log records are written to a log data set called the active log. The name and size of the active log data set is held in a data set called the bootstrap data set.

When the active log data set fills up, the queue manager switches to another log data set so that logging can continue, and copies the content of the full active log data set to an archive log data set. Information about these actions, including the name of the archive data set, is held in the bootstrap data set. Conceptually, there is a ring of active log data sets that the queue manager cycles through; when an active log is filled, the log data is off-loaded to an archive log, and the active log data set is available for reuse.

The log and bootstrap data sets are discussed in "Chapter 4. Logging" on page 33.

Tailoring the queue manager environment

When the queue manager is started, a set of initialization parameters that control how the queue manager will operate are read. In addition, data sets containing MQSeries commands are read, and the commands they contain are executed. Typically, these data sets contain definitions of the system objects required for MQSeries to run, and you can tailor these to define or initialize the MQSeries objects necessary for your operating environment. When these data sets have been read, any objects defined by them are stored, either on a page set or in DB2.

Initialization parameters and system objects are discussed in "Chapter 5. Defining your system" on page 41.

Recovery and restart

At any point in time during the operation of MQSeries, there might be changes held in processor storage that have not yet been written to the page set. These changes are written out to the page set on a "least recently used" basis by a background task within the queue manager.

If the queue manager should terminate abnormally, the recovery phase of queue manager restart can recover the lost page set changes because persistent message data is held in log records. This means that MQSeries can recover persistent message data and object changes right up to the point of failure.

Recovery and restart are discussed in "Chapter 6. Recovery and restart" on page 49.

Security

You can use an external security manager, such as RACF$^{®}$ to protect the resources that MQSeries owns and manages from access by unauthorized users.

MQSeries security is discussed in "Chapter 7. Security" on page 61.

Availability

There are several features of MQSeries that are designed to increase system availability in the event of queue manager or communications subsystem failure. These features are discussed in "Chapter 8. Availability" on page 69.

Manipulating objects

When the queue manager is running, MQSeries objects can be manipulated either through an OS/390 console interface, or through an administration utility that uses ISPF services under TSO. Both mechanisms allow MQSeries objects to be defined, altered or deleted. Various MQSeries and queue manager functions can be controlled and status displayed.

These facilities are discussed in "Chapter 9. Creating and managing objects" on page 73.

Monitoring and statistics

Several facilities are available to monitor your queue managers and channel initiators. You can also collect statistics for performance evaluation and accounting purposes.

These facilities are discussed in"Chapter 10. Monitoring and statistics" on page 83.

Application environments

When the queue manager has started, applications can connect to it and start using the MQSeries API. These can be CICS, IMS, or Batch applications. MQSeries applications can also access applications on CICS and IMS systems that are not aware of MQSeries, using the CICS and IMS bridges.

These facilities are discussed in "Part 3. MQSeries and other products" on page 85.

Internet Gateway

The Internet Gateway provides a bridge between the synchronous World Wide Web and asynchronous MQSeries applications. With the gateway, Web server software and MQSeries together provide an Internet-connected Web browser with access to MQSeries applications. The gateway enables enterprises to take advantage of the low-cost access to global markets provided by the Internet, while benefiting from the robust infrastructure and assured message delivery of MQSeries. Users interact with the gateway through HTML fill-out form POST requests; MQSeries applications respond by returning HTML pages to the gateway, via an MQSeries queue. The MQSeries Internet Gateway supports the CGI and ICAPI Web server interfaces.

The Internet Gateway is an optional feature of MQSeries, and is supplied on the MQSeries product tape or cartridge. It is described in the *MQSeries Internet Gateway for MVS User's Guide* which is available on the Web at:

`http://www.ibm.com/software/mqseries/library/manuals/`

What is a channel initiator?

The *channel initiator* provides and manages resources that allow MQSeries distributed queuing. MQSeries uses *Message Channel Agents* (MCAs) to send messages from one queue manager to another.

To send messages from queue manager A to queue manager B, a *sending* MCA on queue manager A must set up a communications link to queue manager B. A *receiving* MCA must be started on queue manager B to receive messages from the communications link. This one-way path consisting of the sending MCA, the communications link, and the receiving MCA is known as a *channel*. The sending MCA takes messages from a transmission queue and sends them down a channel to the receiving MCA. The receiving MCA receives the messages and puts them on to the destination queues.

In MQSeries for OS/390, the sending and receiving MCAs all run inside the channel initiator (the channel initiator is also known as the *mover*). The channel initiator runs as an OS/390 address space under the control of the queue manager. There can only be a single channel initiator connected to a queue manager and it is run inside the same OS/390 image as the queue manager. There can be thousands of MCA processes running inside the channel initiator concurrently.

Figure 2 shows two queue managers within a sysplex. Each queue manager has a channel initiator and a local queue. Messages sent by queue managers on AIX® and Windows NT® are placed on the local queue, from where it is retrieved by an application. Reply messages are returned by a similar route.

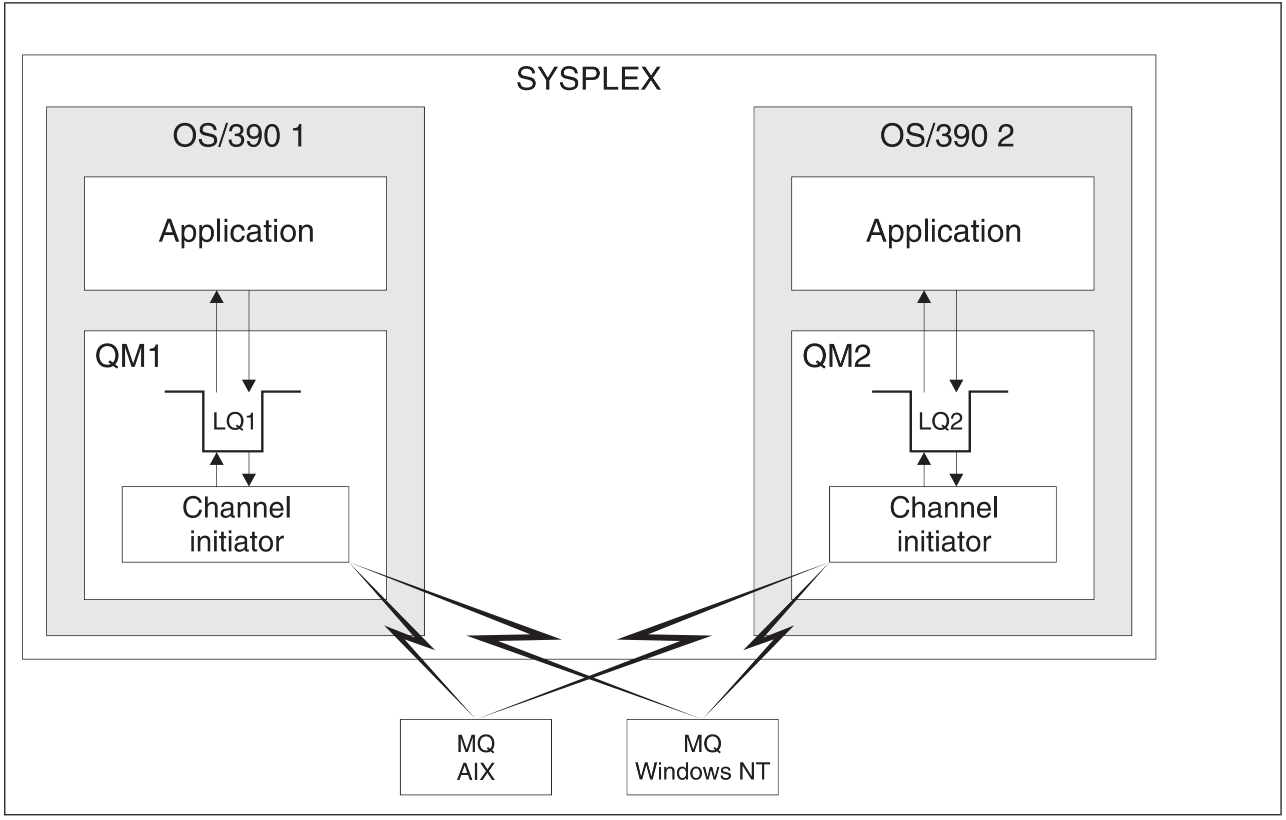


Figure 2. Communication between queue managers

The channel initiator also contains other processes concerned with the management of the channels. These include:

Listeners
These listen for inbound channel requests on a communications subsystem such as TCP, and start a named MCA when an inbound request is received.

Supervisor
This manages the channel initiator address space, for example it is responsible for restarting channels after a failure.

Name server
This is used to resolve TCP names into addresses.

Distributed queuing is described in the *MQSeries Intercommunication* manual.

Queue manager clusters

You can group queue managers in a *cluster*. Queue managers in a cluster 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 the need for many of the object definitions required for standard distributed queuing. Each queue manager in the cluster has a single transmission queue from which it can transmit messages to any other queue manager in the cluster.

Clusters are described in the *MQSeries Queue Manager Clusters* manual.

Part 2. MQSeries for OS/390 concepts

Chapter 2. Shared queues and queue-sharing groups

This chapter describes how several queue managers can share the same queues and the messages on those queues. It discusses the following topics:

What is a shared queue?

A *shared queue* is a type of local queue. The messages on that queue can be accessed by one or more queue managers that are in a sysplex. The queue managers that can access the same set of shared queues form a group called a *queue-sharing group*.

Messages can be accessed by any queue manager

A shared queue can be accessed by any queue manager in the queue-sharing group. This means that you can put a message on to a shared queue on one queue manager, and get the same message from the queue from a different queue manager. This provides a rapid mechanism for communication within a queue-sharing group that does not require channels to be active between queue managers.

The messages on a shared queue are stored in the OS/390 Coupling Facility. Figure 3 on page 12 shows three queue managers and a Coupling Facility, forming a queue-sharing group. All three queue managers can access the shared queue in the Coupling Facility.

An application can connect to any of the queue managers within the queue-sharing group. Because all the queue managers in the queue-sharing group can access all the shared queues, the application does not depend on the availability of a specific queue manager; any queue manager in the queue-sharing group can service the queue.

This gives greater availability because all the other queue managers in the queue-sharing group can continue processing the queue if one of the queue managers has a problem.

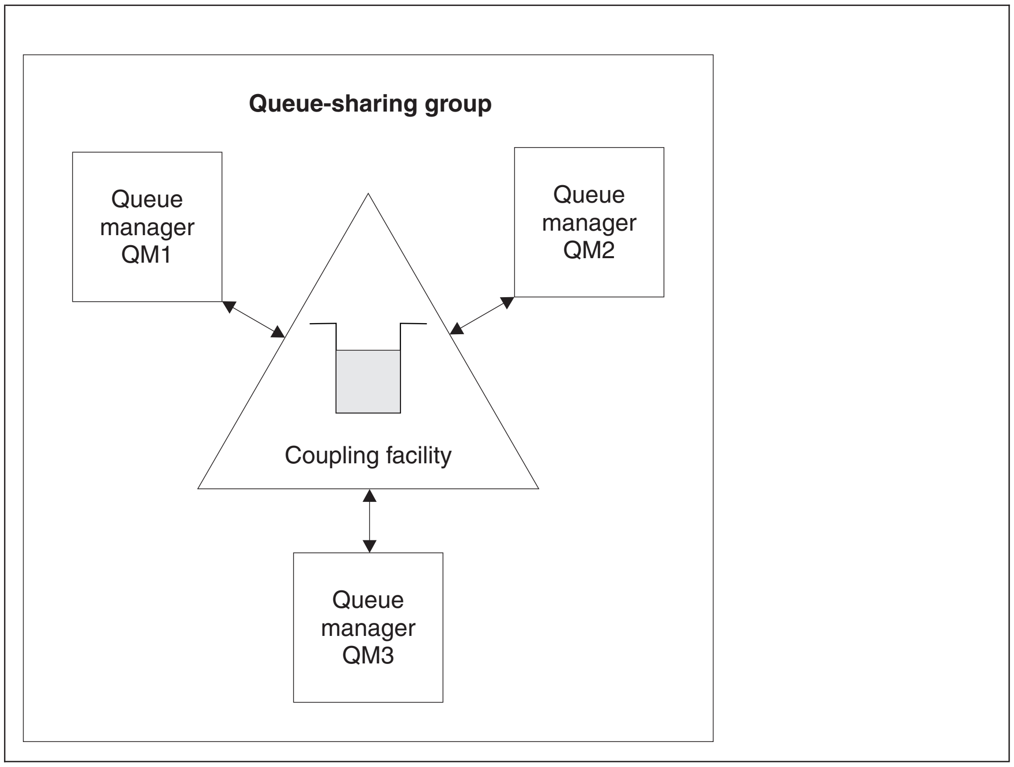


Figure 3. A queue-sharing group

Queue definition shared by all queue managers

The definition of a shared queue is stored in a DB2 shared database called the *shared repository*. Because of this, the queue need only be defined once and then it can be accessed by all the queue managers in the queue-sharing group. This means that there are fewer definitions to make.

By contrast, the definition of a non-shared queue is stored on page set zero of the queue manager that owns the queue (as described in “Page sets” on page 27).

You cannot define a shared queue if a queue with that name has already been defined on the page sets of the defining queue manager. Likewise, you cannot define a local version of a queue on the queue manager page sets if a shared queue with the same name already exists.

What is a queue-sharing group?

The group of queue managers that can access the same shared queues is called a queue-sharing group. Each member of the queue-sharing group has access to the same set of shared queues.

Figure 4 illustrates a queue-sharing group that contains two queue managers. Each queue manager has a channel initiator and its own local page sets and log data sets.

Each member of the queue-sharing group must also connect to a DB2 system. The DB2 systems must all be in the same DB2 data-sharing group so that the queue managers can access the DB2 shared repository used to hold shared object definitions. These are definitions of any type of MQSeries object (for example, queues and channels) that are defined once only and can then be used by any queue manager in the group. These are called *global* definitions and are described in “Private and global definitions” on page 73.

A particular data-sharing group can be referenced by more than one queue-sharing group. You specify the name of the DB2 subsystem and which data-sharing group a queue manager uses in the MQSeries system parameters at startup.

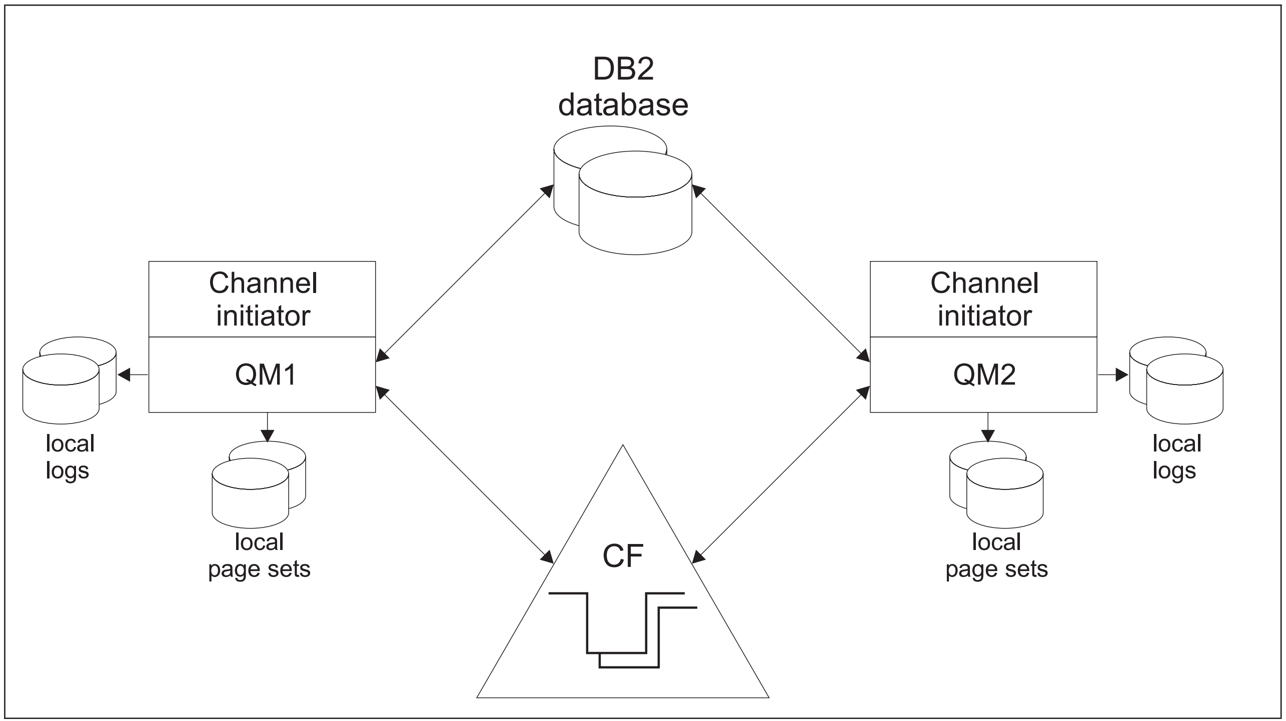


Figure 4. The components of queue managers in a queue-sharing group

When a queue manager has joined a queue-sharing group, it has access to the shared objects defined for that group, and can be used to define new shared objects within the group. If shared queues are defined within the group, this queue manager can be used to put messages to and get messages from those shared queues. Messages held on a shared queue can be retrieved by any queue manager in the group.

Shared queues and queue-sharing groups

You can enter an MQSeries command once, and have it executed on all queue managers within the queue-sharing group as if it had been entered at each queue manager individually. The *command scope* attribute is used for this. This attribute is described in "Directing commands to different queue managers" on page 75.

When a queue manager runs as a member of a queue-sharing group it must be possible to distinguish between MQSeries objects defined privately to that queue manager and MQSeries objects defined globally that are available to all queue managers in the queue-sharing group. The *queue-sharing group disposition* attribute is used for this. This attribute is described in "Private and global definitions" on page 73.

You can define a single set of security profiles that control access to MQSeries objects anywhere within the group. This means that the number of profiles you have to define is greatly reduced.

A queue manager can belong to one queue-sharing group only, and all queue managers in the group must be in the same sysplex. You specify which queue-sharing group the queue manager belongs to in the system parameters at startup.

Where are shared queue messages held?

The messages in shared queues are stored on list structures in the OS/390 Coupling Facility. They can be accessed by many queue managers in the same sysplex. All queue managers also maintain their own logs and page sets (as shown in Figure 4 on page 13) to use non-shared local queues, and store definitions of private objects on page set zero. Messages that are put on to shared queues are not stored on page sets.

Messages on shared queues are not logged in the queue manager log, so persistent messages are not allowed. You should use non-volatile storage to minimize the risk of data loss in the event of a Coupling Facility failure.

The Coupling Facility

The messages held on shared queues are actually stored inside a Coupling Facility. The Coupling Facility lies outside any of the MVS™ images in the Sysplex and is typically configured to run on a different power supply. The Coupling Facility is therefore resilient to software failures and can be configured so that it is resilient to hardware failures or power-outages. This means that messages stored in the Coupling Facility are highly available.

MQSeries uses Coupling Facility list structures to store messages. This means that the maximum length for messages on a shared queue is 63 KB.

Each coupling facility list structure used by MQSeries is dedicated to a specific queue-sharing group, but a Coupling Facility can hold structures for more than one queue-sharing group. Queue managers in different queue-sharing groups cannot share data. Up to 32 queue managers in a queue-sharing group can connect to a Coupling Facility list structure at the same time.

A single Coupling Facility list structure can contain up to 512 shared queues. The amount of message data is limited by the size of the list structure. The size of the list structure is restricted by the following factors:

- It must lie within a single Coupling Facility.
- It might share the available Coupling Facility storage with other structures for MQSeries and other products.

Advantages of using shared queues

The shared queue architecture, where cloned servers pull work from a single shared queue, has some very useful properties:

- It is scalable, by adding new instances of the server application, or even adding a new OS/390 image with a queue manager (in the queue-sharing group) and a copy of the application.
- It is highly available.
- It naturally performs ‘pull’ workload balancing, based on the available processing capacity of each queue manager in the queue-sharing group.

High availability

The following examples illustrate how a shared queue can be used to increase application availability.

Consider an MQSeries scenario where client applications running in the network want to make requests of server applications running on OS/390. The client application constructs a request message and places it on a request queue. The client then waits for a reply from the server, sent to the reply-to queue named in the message descriptor of the request message.

MQSeries manages the transportation of the request message from the client machine to the server’s input queue on OS/390 and of the server’s response back to the client. By defining the server’s input queue as a shared queue, any messages put to the queue can be retrieved on any queue manager in the queue-sharing group. This means that you can configure a queue manager on each OS/390 image in the sysplex and, by connecting them all to the same queue-sharing group, any one of them can access messages on the server’s input queue.

Messages on the server’s input queue will still be available, even if one of the queue managers terminates abnormally or has to be stopped for administrative reasons. In fact, an entire OS/390 image can be taken off-line and the messages will still be available.

To take advantage of this availability of messages on a shared queue, run an instance of the server application on each OS/390 image in the sysplex to provide higher server application capacity and availability, as shown in Figure 5 on page 17.

One instance of the server application retrieves a request message from the shared queue and, based on the content, performs its processing, producing a result to be sent back to the client as an MQSeries message. The response message is destined for the reply-to queue and reply-to queue manager named in the message descriptor of the request message.

There are a number of options that can be used to configure the return path; these are considered in “Distributed queuing and queue-sharing groups” on page 18.

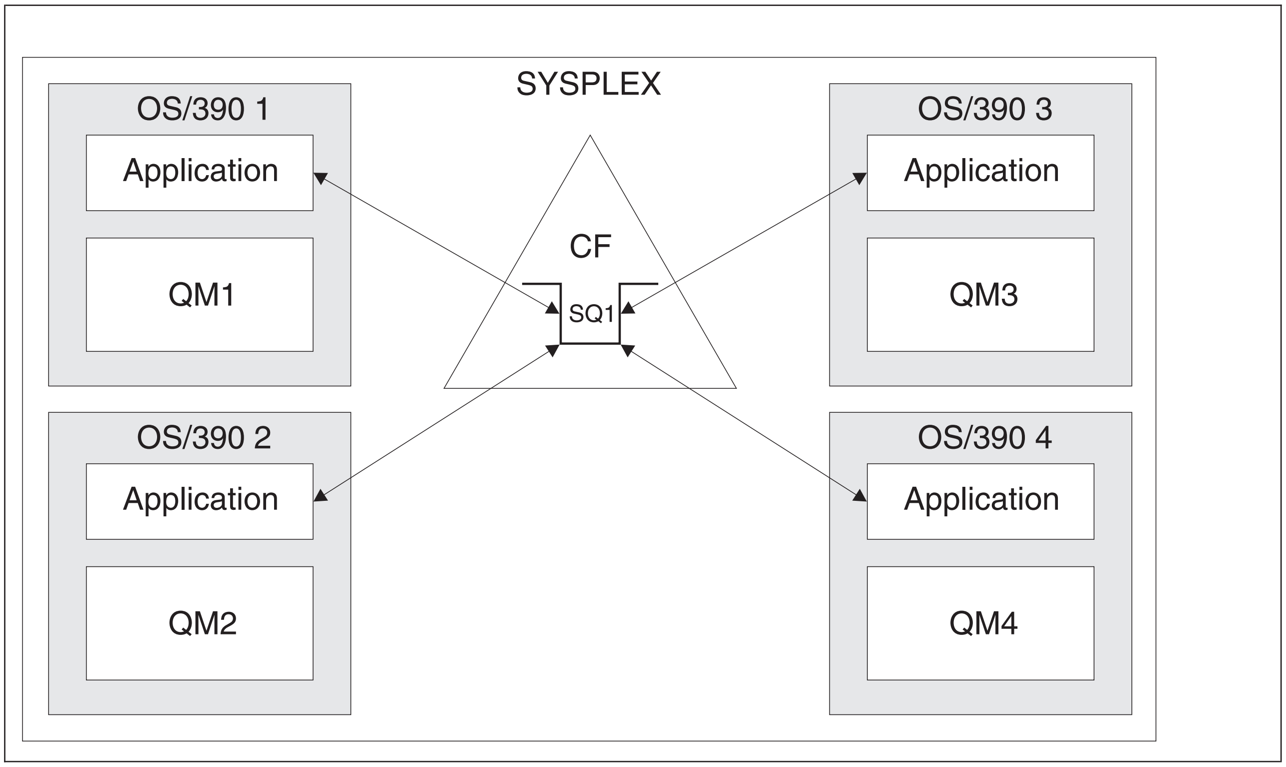


Figure 5. Multiple instances of an application servicing a shared queue

Peer recovery

To further enhance the availability of messages in a queue-sharing group, MQSeries detects if another queue manager in the group disconnects from the Coupling Facility abnormally and completes units of work for that queue manager that are still pending, where possible. This is known as *peer recovery.*

Suppose a queue manager terminates abnormally at a point where an application has retrieved a request message from a queue in syncpoint, but has not yet put the response message or committed the unit of work. Another queue manager in the queue-sharing group detects the failure, and backs out the in-flight units of work being performed on the failed queue manager. This means that the request message is put back on to the request queue and is available for one of the other server instances to process, without waiting for the failed queue manager to restart.

If MQSeries is not able to resolve a unit of work automatically, you can resolve the shared portion manually to enable another queue manager in the queue-sharing group to continue processing that work.

Distributed queuing and queue-sharing groups

To complement the high availability of messages on shared queues, the distributed queuing component of MQSeries has additional functions to provide the following:

- Higher availability to the network.
- Increased capacity for inbound network connections to the queue-sharing group.

Figure 6 illustrates distributed queuing and queue-sharing groups. It shows two queue managers within a sysplex, both of which belong to the same queue-sharing group. They can both access shared queue SQ1. Queue managers in the network (on AIX and Windows NT for example) can put messages onto this queue through the channel initiator of either queue manager. Cloned applications on both queue managers service the queue.

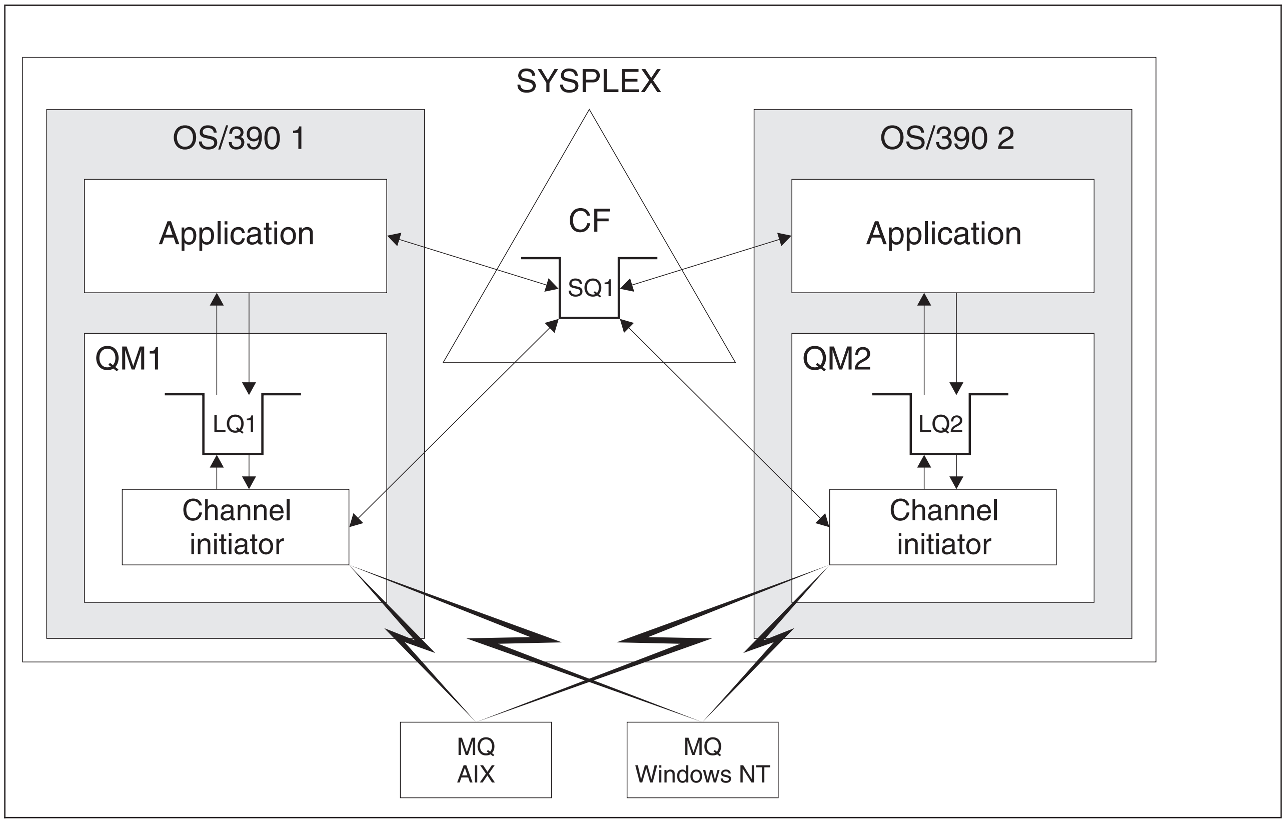


Figure 6. Distributed queuing and queue-sharing groups

Shared channels

A number of networking products provide a mechanism to hide server failures from the network, or to balance inbound network requests across a set of eligible servers. These include:

- VTAM® generic resources
- TCP/IP Domain Name System (DNS)

The channel initiator takes advantage of these products to exploit the capabilities of shared queues.

Shared inbound channels

Each channel initiator in the queue-sharing group starts an additional listener task to listen on a *generic port*. This generic port is made available to the network through one of the technologies mentioned above. This means that an inbound network attach request for the generic port can be dispatched to any one of the listeners in the queue-sharing group that are listening on the generic port.

A channel can only be started on the channel initiator to which the inbound attach is directed if the channel initiator has access to a channel definition for a channel with that name. A channel definition can be defined to be private to a queue manager or stored on the shared repository and available anywhere (a global definition). This means that a channel definition can be made available on any channel initiator in the queue-sharing group by defining it as a global definition.

There is an additional difference when starting a channel through the generic port; channel synchronization is with the queue-sharing group and not with an individual queue manager. For example, consider a client starting a channel through the generic port. When the channel first starts, it might start on queue manager QM1 and messages will flow. If the channel stops and is restarted on queue manager QM2, information about the number of messages that have flowed will still be correct because the synchronization is with the queue-sharing group.

An inbound channel started through the generic port can be used to put messages to any queue. The client does not know whether the target queue is shared or not. If the target queue is a shared queue, the client connects through any available channel initiator in a load-balanced fashion and the messages are put to the shared queue. If the target queue is not a shared queue, the messages might be put on any queue in the queue-sharing group with that name (the environment is one of replicated local queues), and the name of the queue determines the function, regardless of the hosting queue manager.

Shared outbound channels

An outbound channel is considered to be a shared channel if it is taking messages from a shared transmission queue. If it is shared, it holds synchronization information at queue-sharing group level. This means that the channel can be restarted on a different queue manager and channel initiator instance within the queue-sharing group if the communications subsystem, channel initiator, or queue manager fails. Restarting failed channels in this way is a feature of shared channels called *peer channel recovery*.

Workload balancing: An outbound shared channel is eligible for starting on any channel initiator within the queue-sharing group, provided that you have not specified that you want it to be started on a particular channel initiator. The channel initiator selected by MQSeries is determined using the following criteria:

- Is the communications subsystem required currently available to the channel initiator?
- Is a DB2 connection available to the channel initiator?
- Which channel initiator has the lowest current workload? The workload includes channels that are active and retrying.

Shared channel summary

Shared channels differ from non-shared channels in the following ways:

Private channel
Function the same as for previous releases of MQSeries.
- Outbound channel uses a local transmission queue.
- Inbound channel started through a local port.
- Synchronization information held in SYSTEM.CHANNEL.SYNCQ queue.

Shared Channel
Workload balanced with high availability.
- Outbound channel uses a shared transmission queue.
- Inbound channel started through a generic port.
- Synchronization information held in SYSTEM.QSG.CHANNEL.SYNCQ queue.
- Shared outbound channels have a maximum message length of 63 KB, and can only be used for nonpersistent messages.

You specify whether a channel is private or shared when you start the channel. A shared channel can be started by triggering in the same way as a private channel. However, when a shared channel is started, MQSeries performs workload balancing and starts the channel on the most appropriate channel initiator within the queue-sharing group. (If required, you can specify that a shared channel is to be started on a particular channel initiator.)

Shared channel status

The channel initiators in a queue-sharing group maintain a shared channel-status table in DB2. This records which channels are active on which channel initiators. The shared channel-status table is used if there is a channel initiator or communications system failure. It indicates which channels need to be restarted on a different channel initiator in the queue-sharing group.

Intra-group queuing

You can perform fast message transfer between queue managers in a queue-sharing group without defining channels. This uses a system queue called the SYSTEM.QSG.TRANSMIT.QUEUE which is a shared transmission queue. Each queue manager in the queue-sharing group starts a task called the intra-group queuing agent, which waits for messages to arrive on this queue that are destined for their queue manager. When such a message is detected, it is removed from the queue and placed on the correct destination queue.

Standard name resolution rules are used but, if intra-group queuing is enabled and the target queue manager is within the queue-sharing group, the SYSTEM.QSG.TRANSMIT.QUEUE is used to transfer the message to the correct destination queue manager instead of using a transmission queue and channel.

Intra-group queuing is enabled through a queue manager attribute at startup. It can only be used to move nonpersistent messages with a message length of less than 63 KB, including the transmission-queue header (63 KB is the maximum message length for shared queues). Intra-group queuing moves messages outside syncpoint so the message is put to the dead-letter queue if there is a problem delivering it to the target queue. (If you have not defined a dead-letter queue, the message is discarded.)

An inbound shared channel that receives a message destined for a queue on a different queue manager in the queue-sharing group can use intra-group queuing to ‘hop’ the message to the correct destination.

Clusters and queue-sharing groups

You can make your shared queues available to a cluster in a single definition. To do this you specify the name of the cluster when you define the shared queue.

Users in the network see the shared queue as being hosted by each queue manager within the queue-sharing group (the shared queue is not advertised as being hosted by the queue-sharing group). Clients can start sessions with any members of the queue-sharing group to put messages to the same shared queue.

Figure 7 shows how members of a cluster can access a shared queue through any member of the queue-sharing group.

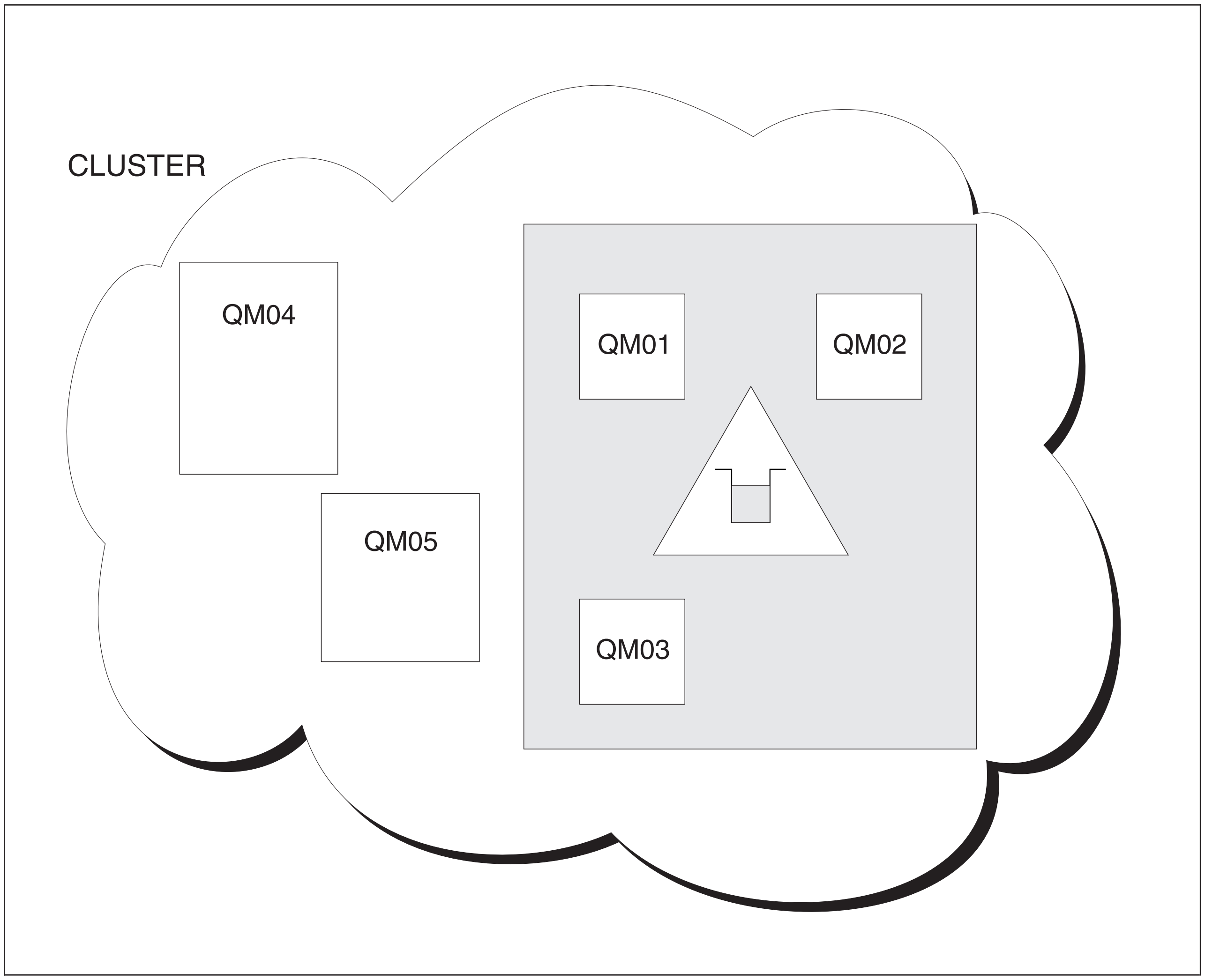


Figure 7. A queue-sharing group as part of a cluster

Application programming with shared queues

This section discusses some of the factors you need to take into account when designing new applications that will use shared queues, and when migrating existing applications to the shared-queue environment.

Serializing your applications

Certain types of applications might have to ensure that messages are retrieved from a queue in exactly the same order as they arrived on the queue. For example, if MQSeries is being used to shadow database updates on to a remote system, a message describing the update to a record must be processed after a message describing the insert of that record. In a local queuing environment, this is often achieved by the application that is getting the messages opening the queue with the MQOO_INPUT_EXCLUSIVE option, thus preventing any other getting application from processing the queue at the same time.

MQSeries allows applications to open shared queues exclusively in the same way. However, if the application is working from a partition of a queue (for example, all database updates are on the same queue, but those for table A have a correlation identifier of A, and those for table B a correlation identifier of B), and applications want to get messages for table A updates and table B updates concurrently, the simple mechanism of opening the queue exclusively is not possible.

If this type of application is to take advantage of the high availability of shared queues, you might decide that another instance of the application that accesses the same shared queues, running on a secondary queue manager, will take over if the primary getting application or queue manager fails.

If the primary queue manager fails, two things happen:

- Shared queue peer recovery ensures that any incomplete updates from the primary application are completed or backed out.
- The secondary application takes over processing the queue.

The secondary application might start before all the incomplete units of work have been dealt with, which could lead to the secondary application retrieving the messages out of sequence. To solve this type of problem, the application can choose to be a *serialized application*.

A serialized application uses the **MQCONNX** call to connect to the queue manager, specifying a connection tag when it connects that is unique to that application. Any units of work performed by the application are marked with the connection tag. MQSeries ensures that units of work within the queue-sharing group with the same connection tag are serialized (according to the serialization options on the **MQCONNX** call).

This means that, if the primary application uses the **MQCONNX** call with a connection tag of `Database shadow retriever`, and the secondary takeover application attempts to use the **MQCONNX** call with an identical connection tag, the secondary application will not be able to connect to the second MQSeries until any outstanding primary units of work have been completed, in this case by peer recovery.

You should consider using the serialized application technique for applications that depend on the exact sequence of messages on a queue. In particular:

- Applications that must not restart after an application or queue manager failure until all commit and back-out operations for the previous execution of the application are complete.

  In this case, the serialized application technique is only applicable if the application works in syncpoint.
- Applications that must not start while another instance of the same application is already running.

  In this case, the serialized application technique is only required if the application cannot open the queue for exclusive input.

Note: MQSeries only guarantees to preserve the sequence of messages when certain criteria are met. These are described in the description of the **MQGET** call in the *MQSeries Application Programming Reference*.

Applications that are not suitable for use with shared queues

Some features of MQSeries are not supported when you are using shared queues, so applications that use these features are not suitable for the shared queue environment. You should consider the following points when designing your shared-queue applications:

- Persistent messages are not supported. However, the nonpersistent messages on shared queues are stored in the coupling facility, so they are not lost if an individual queue manager in the queue-sharing group fails. The nonpersistent messages on shared queues are lost only if the Coupling Facility fails.
- Messages on shared queues cannot be greater than 63 KB in size. Because Coupling Facility storage is limited, you must also consider the number of messages to be generated by the application to ensure that the messages will not fill the queue. However, remember that you can monitor the queue and start more versions of the application on different queue managers to service it if this is a problem.
- Queue indexing is limited for shared queues. If you want to use the message identifier or correlation identifier to select the message you want to get from the queue, the queue must have the correct index defined. If you do not define a queue index, applications can only get the next available message.
- You cannot use temporary dynamic queues as shared queues. You can use permanent dynamic queues however. The models for shared dynamic queues have a DEFTYPE of SHAREDYN (shared dynamic) although they are created and destroyed in the same way as PERMDYN (permanent dynamic) queues.

Deciding whether to share non-application queues

There are queues other than application queues that you might want to consider sharing:

Initiation queues

If you define a shared initiation queue, you do not need to have a trigger monitor running on every queue manager in the queue-sharing group, as long as there is at least one trigger monitor running. (You can also use a shared initiation queue even if there is a trigger monitor running on each queue manager in the queue-sharing group.)

If you have a shared application queue and use the trigger type of EVERY (or a trigger type of FIRST with a small trigger interval, which behaves like a trigger type of EVERY) your initiation queue should always be a shared queue. For more information about when to use a shared initiation queue, see Table 1 on page 25.

Dead-letter queue

You should not define your dead-letter queue as a shared queue. This is because shared queues cannot hold persistent messages, or messages with a size greater than 63 KB. If you use a shared dead-letter queue, any persistent messages (or nonpersistent messages longer than 63 KB) that cannot be delivered to their destination queues will be discarded.

SYSTEM.* queues

You can define the SYSTEM.ADMIN.* queues used to hold event messages as shared queues. This can be useful to check load balancing if an exception occurs. Each event message created by MQSeries contains a correlation identifier indicating which queue manager produced it.

You must define the SYSTEM.QSG.* queues used for shared channels and intra-group queuing as shared queues.

You can also change the definition of the SYSTEM.DEFAULT.LOCAL.QUEUE to be shared, or define your own default shared queue definition. This is described in “Defining system objects” on page 42.

You cannot define any of the other SYSTEM.* queues as shared queues.

Migrating your existing applications to use shared queues

Migrating your existing queues to shared queues is described in the *MQSeries for OS/390 System Administration Guide*.

When you are migrating your existing applications, you should consider the following things, which might work in a slightly differently way in the shared queue environment.

Reason Codes

When you are migrating your existing applications to use shared queues, remember to check for the new reason codes that can be issued.

MQINQ

When you use the **MQINQ** call to display information about a shared queue, the values of the number of **MQOPEN** calls that have the queue open for input and output relate only to the queue manager that issued the call. No information is produced about other queue managers in the queue-sharing group that have the queue open.

Triggering

If you are using a shared application queue, triggering works on committed messages only (on a non-shared application queue, triggering works on all messages).

If you use triggering to start applications, you might want to use a shared initiation queue. Table 1 describes what you need to consider when deciding which type of initiation queue to use.

Table 1. When to use a shared-initiation queue

	Non-shared application queue	**Shared application queue**
Non-shared initiation queue	As for previous releases.	If you are using trigger type of FIRST or DEPTH, you can use a non-shared initiation queue with a shared application queue. There is the possibility of extra trigger messages being generated, but this setup is good for triggering long-running applications (like the CICS bridge) and provides high availability. For trigger type FIRST or DEPTH, a trigger message will trigger an instance of the application on every queue manager that is running a trigger monitor and that does not already have the application queue open for input.
Shared initiation queue	You should not use a shared initiation queue with a non-shared application queue.	If you have a shared application queue that has trigger type EVERY, you should use a shared initiation queue or you will lose trigger messages. For trigger type FIRST or DEPTH, one trigger message is generated by each queue manager that has the named initiation queue open for input. If the initiation queue is defined as a local queue, one trigger message is available to any trigger monitors running on that queue manager against the queue.

Where to find more information

You can find more information about the topics discussed in this chapter from the following sources:

Table 2. Where to find more information about shared queues and queue-sharing groups

Topic	Where to look
Queue-sharing group recovery	"Chapter 6. Recovery and restart" on page 49
Queue-sharing group security	"Chapter 7. Security" on page 61
Private and global object definitions Directing commands to different queue managers	"Chapter 9. Creating and managing objects" on page 73
Planning your Coupling Facility environment	"Defining Coupling Facility resources" on page 123
Planning your DB2 environment	"Planning your DB2 environment" on page 126
Setting up your shared queues System parameters	*MQSeries for OS/390 System Setup Guide*
Utility programs Migrating queues	*MQSeries for OS/390 System Administration Guide*
Console messages	*MQSeries for OS/390 Messages and Codes*
MQSeries commands	*MQSeries MQSC Command Reference*
MQSeries clusters	*MQSeries Queue Manager Clusters*
MQSeries distributed queuing Channel name resolution	*MQSeries Intercommunication*
Writing applications	*MQSeries Application Programming Guide*
MQCONNX call	*MQSeries Application Programming Reference*

Chapter 3. Storage management

This chapter discusses how MQSeries for OS/390 manages storage. It contains the following sections:

Page sets

A *page set* is a linear VSAM data set that has been specially formatted to be used by MQSeries. Page sets are used to store most messages and object definitions.

The exceptions to this are global definitions, which are stored in a shared repository on DB2, and the messages on shared queues. These are not stored on the queue manager page sets. Shared queues are discussed in "Chapter 2. Shared queues and queue-sharing groups" on page 11, and global definitions are discussed in "Private and global definitions" on page 73.

MQSeries page sets can be up to 4 GB in size. Each page set is identified by a page set identifier (PSID), an integer in the range 00 through 99. Each queue manager must have its own page sets.

MQSeries uses page set zero (PSID=00) to store object definitions and other important information relevant to the queue manager subsystem. For normal operation of MQSeries it is essential that page set zero does not become full, so it should not be used to store messages. To improve the performance of your system, you should also separate short lived messages from long lived messages by placing them on different page sets.

Page sets must be formatted so MQSeries provides a FORMAT utility for this, This is described in the *MQSeries for OS/390 System Administration Guide*. Page sets must also be defined to the MQSeries subsystem, this is described in the *MQSeries for OS/390 System Setup Guide*.

If you define secondary extents for your page sets, MQSeries for OS/390 expands a page set dynamically if it becomes full. MQSeries continues to expand the page set if required until 123 logical extents exist, provided that there is sufficient disk storage space available. The extents can span volumes if the LDS is so defined, however, MQSeries cannot expand the page sets beyond 4 GB.

You cannot use page sets from one MQSeries subsystem on a different MQSeries subsystem, or change the subsystem name. If you want to transfer the data from one subsystem to another, you must unload all the objects and messages from the first subsystem and reload them onto another.

Storage classes

A *storage class* maps one or more queues to a page set. This means that messages for that queue are stored on that page set. Shared queues do not use storage classes to obtain a page set mapping because the messages on them are not stored on page sets.

Storage classes allow you to control where non-shared message data is stored for administrative, data set space and load management, or application isolation purposes. Storage classes can also be used to define the XCF group and member name of an IMS region if you are using the IMS bridge (described in "Chapter 12. MQSeries and IMS" on page 99).

How storage classes work

- You define a storage class, using the DEFINE STGCLASS command, specifying a page set identifier (PSID).
- When you define a queue, you can specify a storage class in the STGCLASS attribute.

In the following example, the local queue QE5 is mapped to page set 21 through storage class ARC2.

```
DEFINE STGCLASS(ARC2) PSID(21)
DEFINE QLOCAL(QE5) STGCLASS(ARC2)
```

This means that messages that are put on the queue QE5 are stored on page set 21 (if they stay on the queue long enough to be written to DASD).

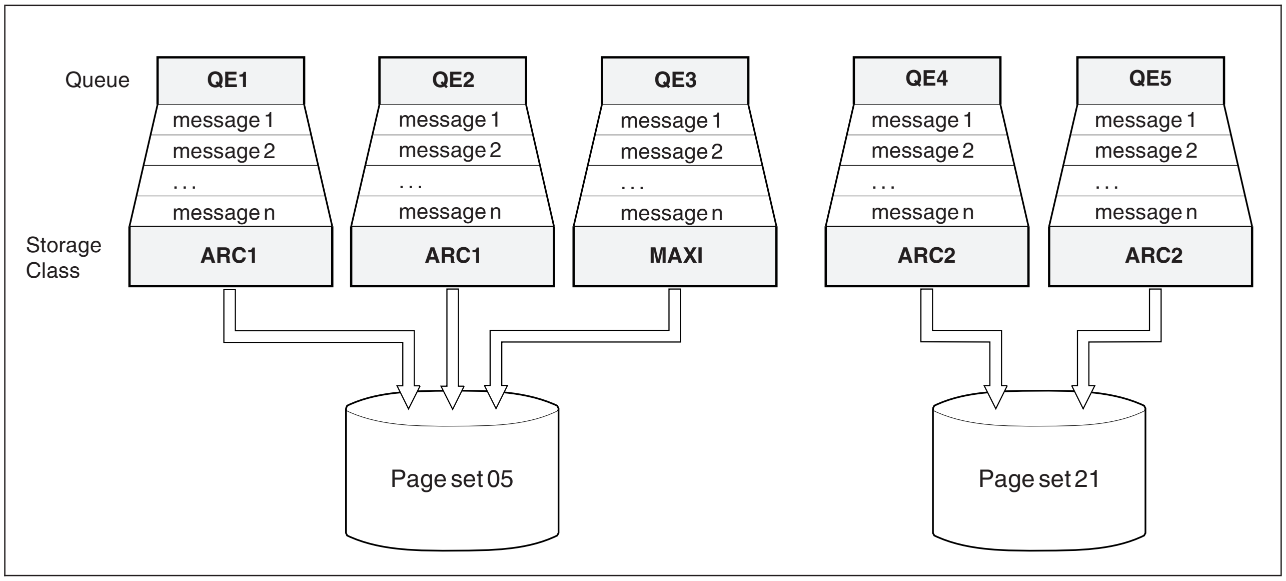


Figure 8. Mapping queues to page sets through storage classes

More than one queue can use the same storage class, and you can define as many storage classes as you like. For example, you can extend the previous example to include more storage class and queue definitions, as follows:

```
DEFINE STGCLASS(ARC1) PSID(05)
DEFINE STGCLASS(ARC2) PSID(21)
DEFINE STGCLASS(MAXI) PSID(05)
DEFINE QLOCAL(QE1) STGCLASS(ARC1) ...
DEFINE QLOCAL(QE2) STGCLASS(ARC1) ...
DEFINE QLOCAL(QE3) STGCLASS(MAXI) ...
DEFINE QLOCAL(QE4) STGCLASS(ARC2) ...
DEFINE QLOCAL(QE5) STGCLASS(ARC2) ...
```

In Figure 8 on page 28, both storage classes ARC1 and MAXI are associated with page set 05. Therefore, the queues QE1, QE2, and QE3 are mapped to page set 05. Similarly, storage class ARC2 associates queues QE4 and QE5 with page set 21.

If you define a queue without specifying a storage class, MQSeries uses a default storage class.

If a message is put on a queue that names a non-existent storage class, the application will receive an error. You must alter the queue definition to give it an existing storage class name, or create the storage class named by the queue.

A storage class can only be changed when:

- All queues that use this storage class are empty, and have no uncommitted activity.
- All queues that use this storage class are closed.

Buffers and buffer pools

For efficiency, MQSeries uses a form of caching whereby messages (and object definitions) are stored temporarily in buffers before being stored in page sets on DASD. Short-lived messages, that is, messages that are retrieved from a queue shortly after they are received, might only ever be stored in the buffers. However, this is all transparent to the user because the buffers are controlled by a buffer manager, which is a component of MQSeries.

The buffers are organized into *buffer pools*. You can define up to four buffer pools (0 through 3) for each MQSeries subsystem; you are recommended to use four buffer pools. Each buffer is 4 KB long. The maximum number of buffers is determined by the amount of storage available in the MQSeries address space, although you should not use more than about 70% of the space for buffers. Usually, the more buffers you have, the more efficient the buffering and the better the performance of the MQSeries subsystem.

Figure 9 shows the relationship between messages, buffers, buffer pools, and page sets. A buffer pool is associated with one or more page sets; each page set is associated with a single buffer pool.

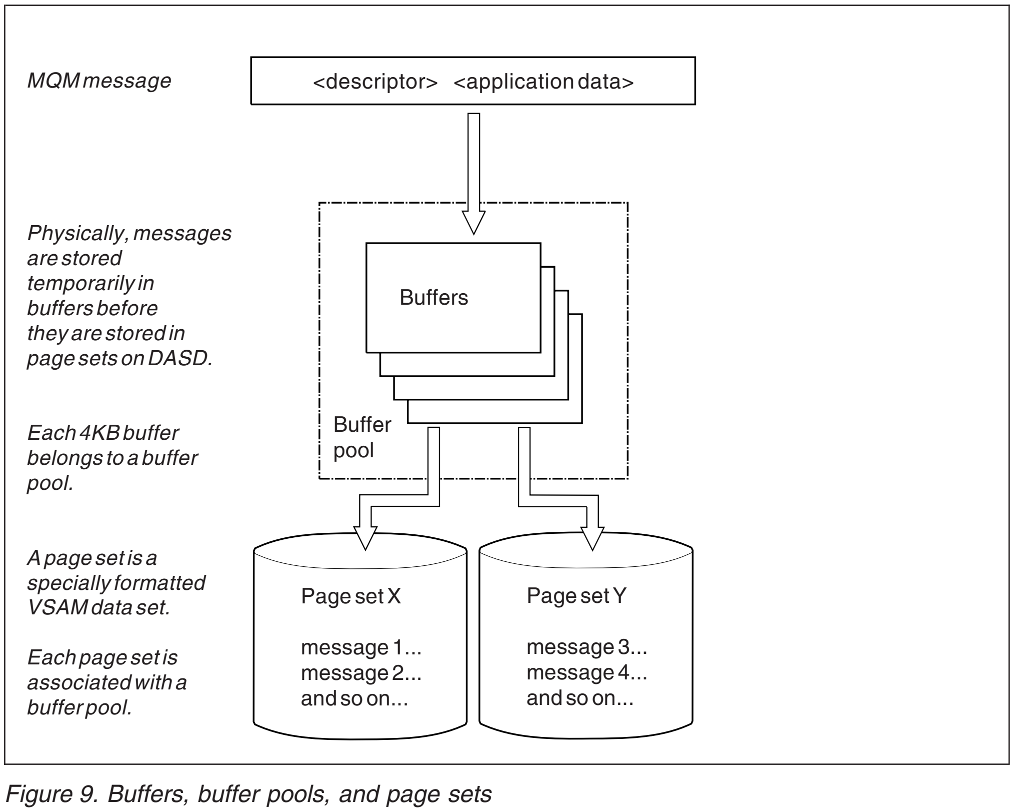

MQM message
<descriptor> <application data>
Physically, messages are stored temporarily in buffers before they are stored in page sets on DASD.
Buffers
Buffer pool
Each 4KB buffer belongs to a buffer pool.
A page set is a specially formatted VSAM data set.
Each page set is associated with a buffer pool.
Page set X
message 1...
message 2...
and so on...
Page set Y
message 3...
message 4...
and so on...

Figure 9. Buffers, buffer pools, and page sets

You specify the number of buffers in a pool with the DEFINE BUFFPOOL command. This command is described in the *MQSeries MQSC Command Reference* manual.

For performance reasons, messages and object definitions should not be in the same buffer pool. You are recommended therefore to use one buffer pool (say number zero) exclusively for page set zero, where the object definitions are kept. Similarly, short-lived messages and long-lived messages should be kept in different buffer pools and therefore on different page sets, and in different queues.

Where to find more information

You can find more information about the topics discussed in this chapter from the following sources:

Table 3. Where to find more information about storage management

Topic	**Where to look**
How much storage you need	"Chapter 14. Planning your storage requirements" on page 107
How large to make your page sets and buffer pools	"Chapter 15. Planning your page sets and buffer pools" on page 113
Defining page sets	*MQSeries for OS/390 System Setup Guide*
Formatting page sets Utility programs	*MQSeries for OS/390 System Administration Guide*
Console messages	*MQSeries for OS/390 Messages and Codes*
MQSeries commands	*MQSeries MQSC Command Reference*

Storage management

Chapter 4. Logging

MQSeries maintains *logs* of data changes and significant events as they occur. The *bootstrap data set* (BSDS) stores information about the data sets that contain the logs.

This chapter contains the following sections:

The log does not contain information for statistics, traces, or performance evaluation. The statistical and monitoring information that MQSeries collects is discussed in "Chapter 10. Monitoring and statistics" on page 83.

What logs are

MQSeries records all significant events as they occur in an *active log*. The log contains the information needed to recover:

- Persistent messages
- MQSeries objects, such as queues
- The MQSeries subsystem

The log does not contain information about messages that are held on shared queues (these are discussed in "Chapter 2. Shared queues and queue-sharing groups" on page 11).

Archiving

Because the active log is finite, MQSeries copies the contents of each log data set periodically to an *archive log*, which is normally a data set on a direct access storage device (DASD) or a magnetic tape. If there is a subsystem or transaction failure, MQSeries uses the active log and, if necessary, the archive log for recovery.

The archive log can contain up to 1000 sequential data sets. Each data set can be cataloged using the OS/390 integrated catalog facility (ICF).

Archiving is an essential component of MQSeries recovery. If a unit of recovery is a long-running one, it is possible that log records within that unit of recovery will be found in the archive log. In this case, recovery will require data from the archive log. However, if archiving is switched off, the active log with new log records will wrap, overwriting earlier log records. This means that MQSeries might not be able to back out the unit of recovery and messages might be lost. MQSeries then terminates abnormally.

Therefore, in a production environment, **you should never switch archiving off**. If you do, you run the risk of losing data after a system or transaction failure. Only if you are running in a test environment can you consider switching archiving off. If you need to do this, use the CSQ6LOGP macro, which is described in the *MQSeries for OS/390 System Setup Guide*.

In order to help prevent problems with unplanned long-running units of work, MQSeries issues a message (CSQJ160I or CSQJ161I) if a long-running unit of work is detected during active log offload.

Dual logging

You can configure MQSeries to run with either *single logging* or *dual logging*. With single logging, log records are written once to an active log data set. With dual logging, each log record is written to two different active log data sets. Dual logging minimizes the likelihood of data loss problems during restart. If possible, the two log data sets should be on separate volumes. This reduces the risk of them both being lost if one of the volumes is corrupted or destroyed. If both copies of the log are lost, the probability of data loss is high.

Note: You should always use dual logging and dual BSDSs rather than dual writing to DASD.

Single logging gives you 2 through 53 active log data sets, whereas dual logging gives you 4 through 106. Each active log data set is a single-volume, single-extent VSAM linear data set (LDS).

Although the minimum number of log data sets required is two, in practice you should have at least three, and on a busy system you might need more. This is to allow time for each log data set to be copied to archive before it is reused in the active log cycle.

Log data

The log can contain up to 280 million million ($2.8*10^{14}$) bytes. Each byte can be addressed by its offset from the beginning of the log, and that offset is known as its *relative byte address* (RBA).

The log is made up of *log records*, each of which is a set of log data treated as a single unit. A log record is identified by the RBA of the first byte of its header; that RBA is called the relative byte address of the record. The RBA uniquely identifies a record that starts at a particular point in the log.

Each log record has a header that gives its type, the MQSeries subcomponent that made the record, and, for unit of recovery records, a unit of recovery identifier.

There are three types of log record, described under these headings:
- "Unit-of-recovery log records"
- "Checkpoint records" on page 35
- "Page set control records" on page 35

Unit-of-recovery log records

Most of the log records describe changes to MQSeries queues. All such changes are made within units of recovery.

MQSeries uses special logging techniques involving *undo/redo* and *compensating log records* to reduce restart times and improve system availability.

One effect of this is that the restart time is bounded. If a failure occurs during a restart so that MQSeries has to be restarted a second time, all the recovery activity that completed to the point of failure in the first restart does not need to be re-applied during a second restart. This means that successive restarts do not take progressively longer times to complete.

Checkpoint records

To reduce restart time, MQSeries takes periodic checkpoints during normal operation. These occur:

- When a predefined number of log records has been written. This number is defined by the checkpoint frequency operand called LOGLOAD of the system parameter macro CSQ6SYSP, described in the *MQSeries for OS/390 System Setup Guide*.
- At the end of a successful restart.
- At normal termination.
- Whenever MQSeries switches to the next active log data set in the cycle.

At the time a checkpoint is taken, MQSeries issues the DISPLAY THREAD command (described in the *MQSeries MQSC Command Reference*) internally so that a list of threads currently in doubt is written to the OS/390 console log.

Page set control records

These records register the page sets known to the MQSeries subsystem at each checkpoint, and record information about the log ranges required to perform media recovery of the page set at the time of the checkpoint.

How the log is structured

Each active log data set must be a VSAM linear data set (LDS). The physical output unit written to the active log data set is a 4 KB control interval (CI). Each CI contains one VSAM record.

Physical and logical log records

One VSAM CI is a *physical* record. The information to be logged at a particular time forms a *logical* record, whose length varies independently of the space available in the CI. So one physical record might contain:

- Several logical records
- One or more logical records and part of another logical record
- Part of one logical record only

The term “log record” refers to the *logical* record, regardless of how many *physical* records are needed to store it.

How the logs are written

MQSeries writes each log record to a DASD data set called the *active log*. When the active log is full, MQSeries copies its contents to a DASD or tape data set called the *archive log*. This process is called *off-loading*.

Figure 10 illustrates the process of logging. Log records typically go through the following cycle:

1. MQSeries notes changes to data and significant events in recovery log records.
2. MQSeries processes recovery log records and breaks them into segments, if necessary.
3. Log records are placed sequentially in *output log buffers*, which are formatted as VSAM CIs. Each log record is identified by a relative byte address in the range 0 through 2^{48}–1.
4. The CIs are written to a set of predefined DASD active log data sets, which are used sequentially and recycled.
5. If archiving is active, as each active log data set becomes full, its contents are automatically off-loaded to a new archive log data set.

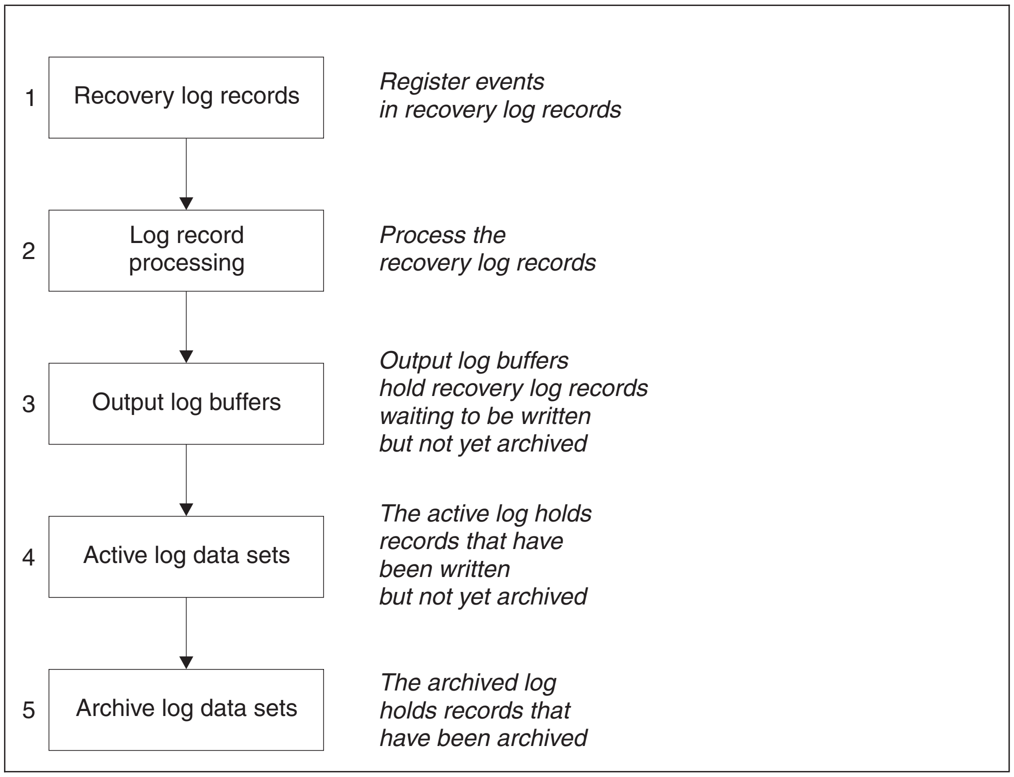

1
Recovery log records
Register events in recovery log records
2
Log record processing
Process the recovery log records
3
Output log buffers
Output log buffers hold recovery log records waiting to be written but not yet archived
4
Active log data sets
The active log holds records that have been written but not yet archived
5
Archive log data sets
The archived log holds records that have been archived

Figure 10. The logging process

When the active log is written

The in-storage log buffers are written to an active log data set whenever any of the following occur:

- The log buffers become full.
- The write threshold is reached (as specified in the CSQ6LOGP macro).
- Certain significant events occur, such as a commit point.

When MQSeries is initialized, the active log data sets named in the BSDS are dynamically allocated for exclusive use by MQSeries and remain allocated exclusively to MQSeries until MQSeries terminates. To add or replace active log data sets, you must terminate and restart MQSeries.

When the archive log is written

The process of copying active logs to archive logs is called *off-loading*. The relation of off-loading to other logging events is shown schematically in Figure 11.

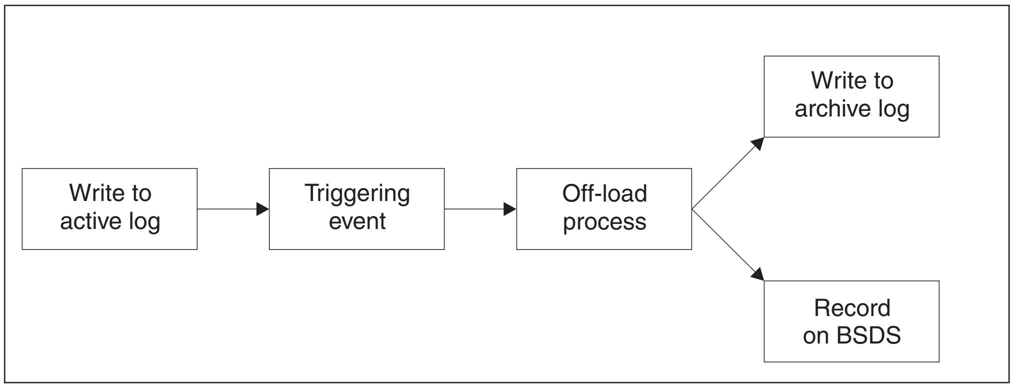


Figure 11. The off-loading process

Triggering an off-load

The off-load of an active log to an archive log can be triggered by several events. For example:

- Filling an active log data set.
- Using the MQSeries command ARCHIVE LOG.
- An error occurring while writing to an active log data set.

The data set is truncated before the point of failure, and the record that failed to be written becomes the first record of the new data set. Off-load is triggered for the truncated data set as it would be for a normal full log data set. If there are dual active logs, both copies are truncated so that the two copies remain synchronized.

Message CSQJ110E is issued when the last available active log is 75% full and at 5% increments thereafter, stating the percentage of the log's capacity in use. If all the active logs become full, MQSeries stops processing until off-loading occurs and issues this message:

```
CSQJ111A +CSQ1 OUT OF SPACE IN ACTIVE LOG DATA SETS
```

The off-load process

When all the active logs become full, the MQSeries subsystem runs an off-load and halts processing until the off-load has been completed. If the off-load processing fails when the active logs are full, MQSeries abends.

When an active log is ready to be off-loaded, a request is sent to the OS/390 console operator to mount a tape or prepare a DASD unit. The value of the ARCWTOR logging option (discussed in the *MQSeries for OS/390 System Setup Guide*) determines whether the request is received. If you are using tape for off-loading, specify ARCWTOR=YES. If the value is YES, the request is preceded by a WTOR (message number CSQJ008E) telling the operator to prepare an archive log data set to be allocated.

The operator need not respond to this message immediately. However, delaying the response delays the off-load process. It does not affect MQSeries performance unless the operator delays the response for so long that MQSeries runs out of active logs.

The operator can respond by canceling the off-load. In this case, if the allocation is for the first copy of dual archive data sets, the off-load is merely delayed until the next active log data set becomes full. If the allocation is for the second copy, the archive process switches to single copy mode, but for this data set only.

Interruptions and errors while off-loading

A request to stop the queue manager does not take effect until off-loading has finished. If MQSeries fails while off-loading is in progress, off-load begins again when MQSeries is restarted. Off-load handling of I/O errors on the logs is discussed in the *MQSeries for OS/390 System Administration Guide*.

Messages during off-load

Off-load messages are sent to the OS/390 console by MQSeries and the off-load process. These messages can be used to find the RBA ranges in the various log data sets. For an explanation of the off-load messages, see the *MQSeries for OS/390 Messages and Codes* manual.

MQSeries and SMS

MQSeries parameters enable you to specify Storage Management Subsystem (MVS/DFP SMS) storage classes when allocating MQSeries archive log data sets dynamically. MQSeries initiates the archiving of log data sets, but SMS can be used to perform allocation of the archive data set.

What the bootstrap data set is for

The *bootstrap data set* (BSDS) is a VSAM key-sequenced data set (KSDS) that holds information needed by MQSeries. It contains:

- An inventory of all active and archived log data sets known to MQSeries. MQSeries uses this inventory to:
  - Track the active and archived log data sets
  - Locate log records so that it can satisfy log read requests during normal processing
  - Locate log records so that it can handle restart processing

  MQSeries stores information in the inventory each time an archive log data set is defined or an active log data set is reused. For active logs, the inventory shows which are full and which are available for reuse. The inventory holds the relative byte address (RBA) of each log data set.
- A *wrap-around* inventory of all recent MQSeries activity. This is needed if MQSeries has to be restarted.

The BSDS is required if the subsystem has an error and has to be restarted. MQSeries must have a BSDS; it is not optional. To minimize the likelihood of problems during a restart, MQSeries can be configured with dual BSDSs, each recording the same information. This is known as running in *dual mode*. If possible, the copies should be on separate volumes. This reduces the risk of them both being lost if the volume is corrupted or destroyed. You should use dual BSDSs rather than dual write to DASD.

The BSDS is set up when MQSeries is customized and the inventory can be managed using the change log inventory utility (CSQJU003). This utility is discussed in the *MQSeries for OS/390 System Administration Guide*. It is referenced by a DD statement in the MQSeries startup procedure.

Normally, MQSeries keeps duplicate copies of the BSDS. If an I/O error occurs, it deallocates the failing copy and continues with a single BSDS. You can restore dual mode operation, this is described in the *MQSeries for OS/390 System Administration Guide*.

The active logs are first registered in the BSDS when MQSeries is installed. They cannot be replaced, nor can new ones be added, without terminating and restarting MQSeries.

Archive log data sets are allocated dynamically. When one is allocated, the data set name is registered in the BSDS. The list of archive log data sets expands as archives are added, and wraps when a user-determined number of entries has been reached. The maximum number of entries is 1000 for single archive logging and 2000 for dual logging.

You can use a tape management system to delete the archive log data sets (MQSeries does not have an automated method). Therefore, the information about an archive log data set can be in the BSDS long after the archive log data set has been deleted by the system administrator.

Conversely, the maximum number of archive log data sets could have been exceeded, and the data from the BSDS dropped long before the data set has reached its expiry date.

Logging

If the system parameter module specifies that archive log data sets are cataloged when allocated, the BSDS points to the integrated catalog facility (ICF) catalog for the information needed for later allocations. Otherwise, the BSDS entries for each volume register the volume serial number and unit information that is needed for later allocations.

Archive log data sets and BSDS copies

Each time a new archive log data set is created, a copy of the BSDS is also created. If the archive log is on tape, the BSDS is the first data set on the first output volume. If the archive log is on DASD, the BSDS is a separate data set.

The data set names of the archive log and the BSDS copy are the same, except that the lowest-level qualifier of the archive log name begins with A and the BSDS copy begins with B, for example:

Archive log name
CSQ.ARCHLOG1.E00186.T2336229.*A*0000001

BSDS copy name
CSQ.ARCHLOG1.E00186.T2336229.*B*0000001

If there is a read error while copying the BSDS, the copy is not created, message CSQJ125E is issued, and the off-load to the new archive log data set continues without the BSDS copy.

Where to find more information

You can find more information about the topics discussed in this chapter from the following sources:

Table 4. Where to find more information about logging

Topic	Where to look
How many logs are required How large to make the logs	"Chapter 17. Planning your logging environment" on page 127
Setting up your logs System parameter macros	*MQSeries for OS/390 System Setup Guide*
Day to day logging tasks Resolving problems with logs Utility programs	*MQSeries for OS/390 System Administration Guide*
Console messages	*MQSeries for OS/390 Messages and Codes*
MQSeries commands	*MQSeries MQSC Command Reference*

Chapter 5. Defining your system

This chapter discusses the following topics:

- "Setting system parameters"
- "Defining system objects" on page 42
- "Sample definitions supplied with MQSeries" on page 45
- "Where to find more information" on page 48

Setting system parameters

In MQSeries for OS/390, a system parameter module controls the logging, archiving, tracing, and connection environments that MQSeries uses in its operation. The system parameters are specified by three assembler macros, as follows:

CSQ6SYSP
System parameters, including setting the connection and tracing environment.

CSQ6LOGP
Logging parameters.

CSQ6ARVP
Log archive parameters.

There is also a channel initiator parameter module that controls how the channel initiator operates; its macro is called CSQ6CHIP.

Default parameter modules are supplied with MQSeries for OS/390. If these do not contain the values that you want to use, you can create your own parameter modules using the samples supplied with MQSeries. The samples are thlqual.SCSQPROC(CSQ4ZPRM) and thlqual.SCSQPROC(CSQ4XPRM).

Defining system objects

There are several objects that you need to define before you can use MQSeries for OS/390. These are called the system objects and are described here. Sample definitions are supplied with MQSeries to help you define these objects. These samples are described in “Sample definitions supplied with MQSeries” on page 45.

System default objects

The system default objects are used to provide default attributes when you define an object and do not specify the name of another object to base the definition on.

The names of the default system object definitions begin with the characters “SYSTEM.DEFAULT” or “SYSTEM.DEF”. For example, the system default local queue is named SYSTEM.DEFAULT.LOCAL.QUEUE.

These objects define the system defaults for the attributes of these MQSeries objects:
- Local queues
- Model queues
- Alias queues
- Remote queues
- Processes
- Namelists
- Channels
- Storage classes

Shared queues are a special type of local queue, so when you define a shared queue, the definition is based on the SYSTEM.DEFAULT.LOCAL.QUEUE. You need to remember to supply a value for the Coupling Facility structure name because one is not specified in the default definition. Alternatively, you could define your own default shared queue definition to use as a basis for shared queues so that they all inherit the required attributes. Remember that you need to define a shared queue on one queue manager in the queue-sharing group only.

System command objects

The names of the system command objects begin with the characters SYSTEM.COMMAND. You must define these objects before the MQSeries operations and control panels can be used to issue commands to an MQSeries subsystem.

There are two system-command objects:
1. The system-command input queue is a local queue on which commands are put before they are processed by the MQSeries command processor. It must be called SYSTEM.COMMAND.INPUT.
2. SYSTEM.COMMAND.REPLY.MODEL is a model queue that defines the system-command reply-to queue.

Commands are normally sent using nonpersistent messages so both the system-command objects should have the DEFPSIST(NO) attribute so that applications using them (including the supplied applications like the utility program and the operations and control panels) will get nonpersistent messages by default. If you have an application that uses persistent messages for commands, you should set the DEFTYPE(PERMDYN) attribute for the reply-to queue because the reply messages to such commands will be persistent.

System administration objects

These queues are used for event messages. The names of the system administration objects begin with the characters SYSTEM.ADMIN.

There are three system-administration objects:
- The SYSTEM.ADMIN.QMGR.EVENT queue
- The SYSTEM.ADMIN.PERFM.EVENT queue
- The SYSTEM.ADMIN.CHANNEL.EVENT queue

Channel queues

To use distributed queuing, you need to define the following objects:
- A local queue with the name SYSTEM.CHANNEL.SYNCQ, which is used to maintain sequence numbers and logical units of work identifiers (LUWID) of channels. To improve channel performance, you should define this queue with an index type of MSGID (as shown in the supplied sample queue definition).
- Channel command queues with the names SYSTEM.CHANNEL.INITQ and SYSTEM.CHANNEL.REPLY.INFO.

Do not define these queues as shared queues.

Cluster queues

To use MQSeries clusters, you need to define the following objects:
- A local queue called the SYSTEM.CLUSTER.COMMAND.QUEUE, which is used to communicate repository changes between queue managers. Messages written to this queue contain updates to the repository data to be applied to the local copy of the repository, or requests for repository data.
- A local queue called SYSTEM.CLUSTER.REPOSITORY.QUEUE, which is used to hold a persistent copy of the repository.
- A local queue called SYSTEM.CLUSTER.TRANSMIT.QUEUE, which is the transmission queue for all destinations in the cluster. For performance reasons you should define this queue with an index type of CORRELID (as shown in the sample queue definition).

These queues typically contain large numbers of messages.

Do not define these queues as shared queues.

Queue-sharing group queues

To use shared channels and intra-group queuing, you need to define the following objects:
- A shared queue with the name SYSTEM.QSG.CHANNEL.SYNCQ, which is used to hold synchronization information for shared channels.
- A shared queue with the name SYSTEM.QSG.TRANSMIT.QUEUE, which is used as the transmission queue for intra-group queuing. If you are running in a queue-sharing group, you must define this queue, even if you are not using intra-group queuing.

Storage classes

You are recommended to define the following six storage classes. You must define four of them because they are required by MQSeries. The other storage class definitions are recommended because they are used in the sample queue definitions.

DEFAULT (required)
This storage class is used for all message queues that are not performance critical and that don't fit in to any of the other storage classes. It is also the supplied default storage class if you do not specify one when defining a queue.

NODEFINE (required)
This storage class is used if the storage class specified when you define a queue is not defined.

REMOTE (required)
This storage class is used primarily for transmission queues, that is, system related queues with short-lived performance-critical messages.

SYSLNGLV
This storage class is used for long-lived, performance-critical messages.

SYSTEM (required)
This storage class is used for performance critical, system related message queues, for example the SYSTEM.CHANNEL.SYNQ and the SYSTEM.CLUSTER.* queues.

SYSVOLAT
This storage class is used for short-lived, performance-critical messages.

You can modify their attributes and add other storage class definitions as required.

Dead-letter queue

The dead-letter queue is used if the message destination is not valid. MQSeries puts such messages on a local queue called the dead-letter queue. Although having a dead-letter queue is not mandatory, it should be regarded as essential, especially if you are using either distributed queuing or one of the MQSeries bridges.

If you decide to define a dead-letter queue, you must also tell the queue manager its name. To do this use the ALTER QMGR command, as shown in the sample.

Default transmission queue

The default transmission queue is used when no other suitable transmission queue is available for sending messages to another queue manager. If you define a default transmission queue, you must also define a channel to serve the queue. If you do not do this, messages that are put on to the default transmission queue will not be transmitted to the remote queue manager and will remain on the queue.

If you decide to define a default transmission queue, you must also tell the queue manager its name. To do this use the ALTER QMGR command, as shown in the sample.

Sample definitions supplied with MQSeries

The following sample definitions are supplied with MQSeries. You can use them to define the system objects and to customize your own objects. You can include some of them in the initialization input data sets (described in "Initialization commands" on page 77).

Table 5. MQSeries sample definitions for system objects

Initialization input data set	Sample name
CSQINP1	thlqual.SCSQPROC(CSQ4INP1)

CSQINP2	thlqual.SCSQPROC(CSQ4INSG)
	thlqual.SCSQPROC(CSQ4INSS)
	thlqual.SCSQPROC(CSQ4INSX)
	thlqual.SCSQPROC(CSQ4INYC)
	thlqual.SCSQPROC(CSQ4INYD)
	thlqual.SCSQPROC(CSQ4INYG)

Other	thlqual.SCSQPROC(CSQ4DISP)
	thlqual.SCSQPROC(CSQ4DISQ)
	thlqual.SCSQPROC(CSQ4INPX)
	thlqual.SCSQPROC(CSQ4IVPQ)
	thlqual.SCSQPROC(CSQ4IVPG)

The CSQINP1 sample

The sample CSQINP1 data set thlqual.SCSQPROC(CSQ4INP1) contains definitions of buffer pools, page set to buffer pool associations, MAXSMSGS and an ALTER SECURITY command. The sample should be included in the CSQINP1 concatenation of your MQSeries started task procedure.

CSQ4INSG system object sample

The sample CSQINP2 data set thlqual.SCSQPROC(CSQ4INSG) contains definitions for the following system objects for general use:
- System default objects
- System command objects
- System administration objects

You must define the objects in this sample, but you need to do it only once when the subsystem is first started. Including the definitions in the CSQINP2 data set is the best way to do this. They are maintained across MQSeries subsystem shutdown and restart. You must not change the object names, but you can change their attributes if required.

CSQ4INSS system object sample

You can define additional system objects if you are using queue-sharing groups.

Sample data set thlqual.SCSQPROC(CSQ4INSS) contains a set of definitions for the system objects required for shared channels and intra-group queuing.

You cannot use this sample as is; you must customize it before use. Then you can include this member in the CSQINP2 DD concatenation of the MQSeries startup procedure, or you can use it as input to the COMMAND function of the CSQUTIL utility to issue the required DEFINE commands.

When you are defining group or shared objects, you only need to include them in the CSQINP2 DD concatenation for one queue manager in the queue-sharing group.

CSQ4INSX system object sample

You must define additional system objects if you are using distributed queuing and clustering.

Sample data set thlqual.SCSQPROC(CSQ4INSX) contains the queue definitions required. You can include this member in the CSQINP2 DD concatenation of the MQSeries startup procedure, or you can use it as input to the COMMAND function in CSQUTIL utility to issue the required DEFINE commands.

There are two types of object definitions:
- SYSTEM.CHANNEL.xx, needed for any distributed queuing
- SYSTEM.CLUSTER.xx, needed for clustering

CSQ4INYC object sample

If you are using clustering, definitions equivalent to the channel definitions and remote queue definitions of distributed queuing are created automatically, when needed. However, some manual channel definitions are needed – a cluster-receiver channel for the cluster and a cluster-sender definition to at least one cluster repository queue manager.

Sample data set thlqual.SCSQPROC(CSQ4INYC) contains the following sample definitions that you can use for customizing your clustering objects:
- Definitions for the queue manager
- Definitions for the receiving channel
- Definitions for the sending channel
- Definitions for cluster queues
- Definitions for lists of clusters

You cannot use this sample as is; you must customize it before use. Then you can include this member in the CSQINP2 DD concatenation of the MQSeries startup procedure, or you can use it as input to the COMMAND function of the CSQUTIL utility to issue the required DEFINE commands. (This is preferable because it means that you don’t have to redefine these objects each time you restart MQSeries).

CSQ4INYD object sample

If you are using distributed queuing and not clustering, you need to set up your own queues, processes, and channels.

Sample data set thlqual.SCSQPROC(CSQ4INYD) contains sample definitions that you can use for customizing your distributed queuing objects. It comprises:
- A set of definitions for the sending end
- A set of definitions for the receiving end
- A set of definitions for using clients

You cannot use this sample as is; you must customize it before use. Then you can include this member in the CSQINP2 DD concatenation of the MQSeries startup procedure, or you can use it as input to the COMMAND function of the CSQUTIL utility to issue the required DEFINE commands. (This is preferable because it means that you don’t have to redefine these objects each time you restart MQSeries).

CSQ4INYG object sample

Sample data set thlqual.SCSQPROC(CSQ4INYG) contains the following sample definitions that you can use for customizing your own objects for general use:

- Storage classes
- Dead-letter queue
- Default transmission queue
- CICS adapter objects

You cannot use this sample as is, you must customize it before use. Then you can include this member in the CSQINP2 DD concatenation of the MQSeries startup procedure, or you can use it as input to the COMMAND function of the CSQUTIL utility to issue the required DEFINE commands. (This is preferable because it means that you don’t have to redefine these objects each time you restart MQSeries).

In addition to the sample definitions here, you can use the system object definitions as the basis for your own resource definitions. For example, you could make a working copy of SYSTEM.DEFAULT.LOCAL.QUEUE and name it MY.DEFAULT.LOCAL.QUEUE. You can then change any of the parameters in this copy as required. You could then issue a DEFINE command by whichever method you choose, provided you have the authority to create resources of that type.

Default transmission queue

Read “Default transmission queue” on page 44 before you decide whether you want to define a default transmission queue.

- If you decide that you do want to define a default transmission queue, remember that you must also define a channel to serve it.
- If you decide that you do not want to define one, remember to remove the DEFXMITQ statement from the ALTER QMGR command in the sample.

CICS adapter objects

The sample defines an initiation queue named CICS01.INITQ. This queue is used by the MQSeries-supplied CKTI transaction. You can change the name of this queue; however it must match the name specified in the CICS system initialization table (SIT) or SYSIN override in the INITPARM statement.

CSQ4DISP display sample

Sample data set thlqual.SCSQPROC(CSQ4DISP) contains a set of generic DISPLAY commands that display all the defined resources on your MQSeries subsystem. This includes the definitions for all MQSeries objects and definitions such as storage classes and trace. These commands can generate a large amount of output. This sample can be used in the CSQINP2 data set or as input to the COMMAND function of the CSQUTIL utility.

CSQ4DISQ distributed queuing using CICS sample

Sample data set thlqual.SCSQPROC(CSQ4DISQ) contains a set of commands that are required to implement distributed queuing using CICS ISC. (This is described in the *MQSeries for OS/390 System Setup Guide*).

You can include this member in the CSQINP2 DD concatenation of the MQSeries startup procedure, or you can use the it as input to COMMAND function of the CSQUTIL utility to issue the required DEFINE commands.

CSQ4INPX sample

Sample data set thlqual.SCSQPROC(CSQ4INPX) contains a set of commands that you might want to execute each time the channel initiator starts. You must customize this sample before use; you can then include it in the CSQINPX data set for the channel initiator.

CSQ4IVPQ and CSQ4IVPG samples

Sample data sets thlqual.SCSQPROC(CSQ4IVPQ) and thlqual.SCSQPROC(CSQ4IVPG) contain sets of DEFINE commands that are required to run the installation verification programs (IVPs).

You can include these samples in the CSQINP2 data set. When you have run the IVPs successfully, you do not need to run them again each time MQSeries is restarted. Therefore, you do not need to keep these samples in the CSQINP2 concatenation permanently.

Where to find more information

You can find more information about the topics discussed in this chapter from the following sources:

Table 6. Where to find more information about system parameters and system objects

Topic	Where to look
Using initialization input data sets System parameter macros Installation verification program	*MQSeries for OS/390 System Setup Guide*
Utility programs	*MQSeries for OS/390 System Administration Guide*
MQSeries commands	*MQSeries MQSC Command Reference*
MQSeries clusters	*MQSeries Queue Manager Clusters*
MQSeries events	*MQSeries Event Monitoring*

Chapter 6. Recovery and restart

This chapter describes how MQSeries for OS/390 recovers after it has stopped, and what happens when the system is restarted. It contains the following sections:

How changes are made to data

MQSeries must interact with other subsystems to keep all the data consistent. This section discusses *units of recovery*; what they are and how they are used in *back outs*.

Units of recovery

A *unit of recovery* is the processing done by a single MQSeries subsystem for an application program, that changes MQSeries data from one point of consistency to another. A *point of consistency* – also called a *syncpoint* or *commit point* – is a point in time when all the recoverable data that an application program accesses is consistent.

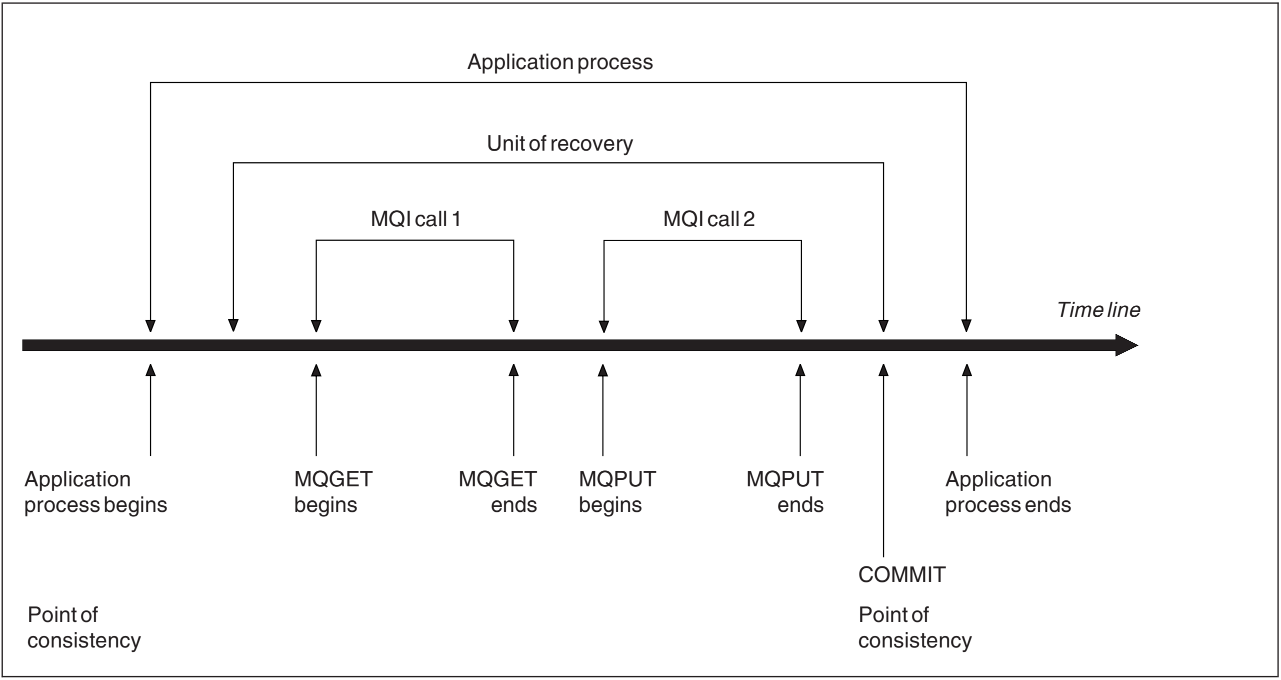


Figure 12. A unit of recovery within an application program. Typically, the unit of recovery consists of more than one MQI call. More than one unit of recovery can occur within an application program.

A unit of recovery begins with the first change to the data after the beginning of the program or following the previous point of consistency; it ends with a later point of consistency. Figure 12 shows the relationship between units of recovery, the point of consistency, and an application program. In this example, the application program makes changes to queues through MQI calls 1 and 2. The application program can include more than one unit of recovery or just one. However, any complete unit of recovery ends in a commit point.

For example, a bank transaction transfers funds from one account to another. First, the program subtracts the amount from the first account, account A. Then, it adds the amount to the second account, B. After subtracting the amount from A, the two accounts are inconsistent and MQSeries cannot commit. They become consistent when the amount is added to account B. When both steps are complete, the program can announce a point of consistency through a commit, making the changes visible to other application programs.

Normal termination of an application program automatically causes a point of consistency. Some program requests in CICS and IMS programs also cause a point of consistency, for example, EXEC CICS SYNCPOINT.

Backing out work

If an error occurs within a unit of recovery, MQSeries removes any changes to data, returning the data to its state at the start of the unit of recovery; that is, MQSeries backs out the work. The events are shown in Figure 13.

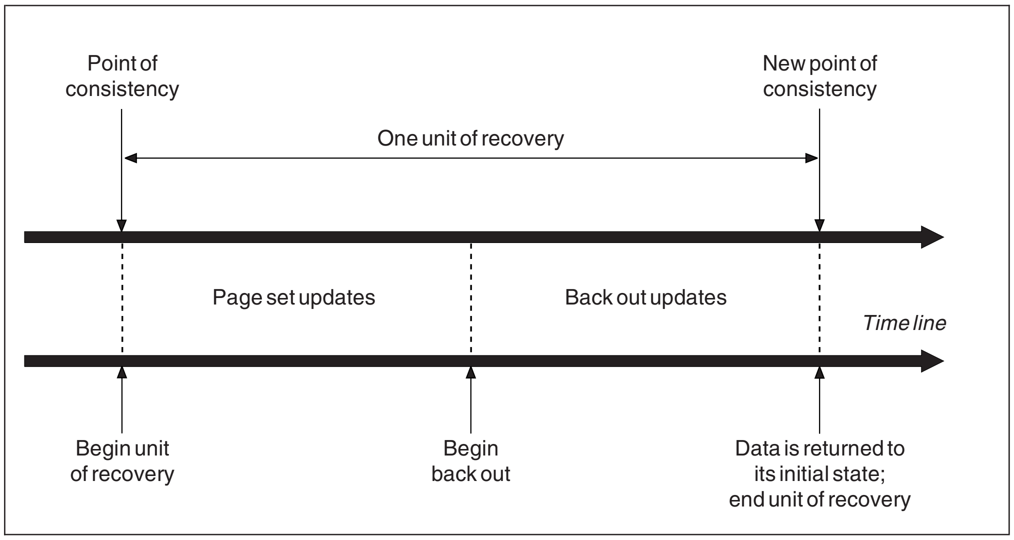


Figure 13. A unit of recovery showing back out

How consistency is maintained

If data in an MQSeries subsystem is to be consistent with batch, CICS, IMS, or TSO, any data changed in one must be matched by a change in the other. Before one system commits the changed data, it must know that the other system can make the corresponding change. So, the systems must communicate.

During a *two-phase commit* (for example under CICS), one subsystem coordinates the process. That subsystem is called the *coordinator*; the other is the *participant*. CICS or IMS is always the coordinator in interactions with MQSeries, and MQSeries is always the participant. In the batch or TSO environment, MQSeries can participate in two-phase commit protocols coordinated by OS/390 RRS.

During a *single-phase commit* (for example under TSO or batch), MQSeries is always the coordinator in the interactions and completely controls the commit process.

Consistency with CICS or IMS

The connection between MQSeries and CICS or IMS supports the following syncpoint protocols:

- Two-phase commit – for transactions that update resources owned by more than one resource manager.

  This is the standard distributed syncpoint protocol. It involves more logging and message flows than a single-phase commit.
- Single-phase commit – for transactions that update resources owned by a single resource manager (MQSeries).

  This protocol is optimized for logging and message flows.
- Bypass of syncpoint – for transactions that involve MQSeries but which do nothing in the queue manager that requires a syncpoint (for example, browsing a queue).

In each case, CICS or IMS acts as the syncpoint manager.

The stages of the two-phase commit that MQSeries uses to communicate with CICS or IMS are:

1. In phase 1, each system determines independently whether it has recorded enough recovery information in its log, and can commit its work.

At the end of the phase, the systems communicate. If they agree, each begins the next phase.

2. In phase 2, the changes are made permanent. If one of the systems abends during phase 2, the operation is completed by the recovery process during restart.

Illustration of the two-phase commit process

Figure 14 illustrates the two-phase commit process. Events in the CICS or IMS coordinator are shown on the upper line, events in MQSeries on the lower line.

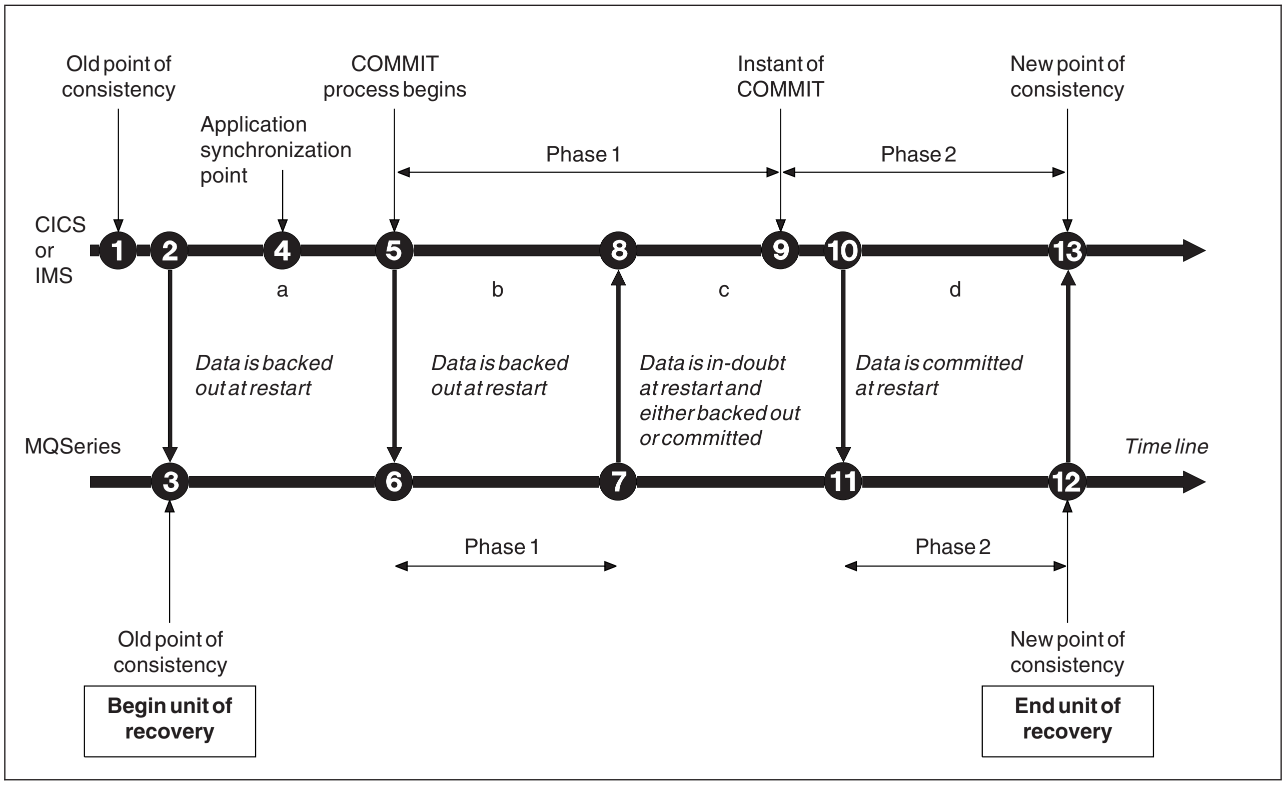


Figure 14. The two-phase commit process

The numbers in the following discussion are linked to those in the figure.

1. The data in the coordinator is at a point of consistency.
2. An application program in the coordinator calls MQSeries to update a queue by adding a message.
3. This starts a unit of recovery in MQSeries.
4. Processing continues in the coordinator until an application synchronization point is reached.
5. The coordinator then starts commit processing. CICS programs use a SYNCPOINT command or a normal application termination to start the commit. IMS programs can start the commit by using a CHKP call, a SYNC call, a GET UNIQUE call to the IOPCB, or a normal application termination. Phase 1 of commit processing begins.
6. As the coordinator begins phase 1 processing, so does MQSeries.
7. MQSeries successfully completes phase 1, writes this fact in its log, and notifies the coordinator.
8. The coordinator receives the notification.
9. The coordinator successfully completes its phase 1 processing. Now both subsystems agree to commit the data changes, because both have completed phase 1 and could recover from any errors. The coordinator records in its log the instant of commit – the irrevocable decision of the two subsystems to make the changes.

The coordinator now begins phase 2 of the processing – the actual commitment.

10. The coordinator notifies MQSeries to begin its phase 2.
11. MQSeries logs the start of phase 2.
12. Phase 2 is successfully completed, and this is now a new point of consistency for MQSeries. MQSeries then notifies the coordinator that it has finished its phase 2 processing.
13. The coordinator finishes its phase 2 processing. The data controlled by both subsystems is now consistent and available to other applications.

How consistency is maintained after an abnormal termination

When MQSeries is restarted after an abnormal termination, it must determine whether to commit or to back out units of recovery that were active at the time of termination. For certain units of recovery, MQSeries has enough information to make the decision. For others, it does not, and must get information from the coordinator when the connection is reestablished.

Figure 14 shows four periods within the two phases: a, b, c, and d. The status of a unit of recovery depends on the period in which termination happened. The status can be:

In flight
: MQSeries ended before finishing phase 1 (period a or b); during restart, MQSeries backs out the updates.

In doubt
: MQSeries ended after finishing phase 1 and before starting phase 2 (period c); only the coordinator knows whether the error happened before or after the commit (point 9). If it happened before, MQSeries must back out its changes; if it happened after, MQSeries must make its changes and commit them. At restart, MQSeries waits for information from the coordinator before processing this unit of recovery.

In commit
: MQSeries ended after it began its own phase 2 processing (period d); it makes committed changes.

In backout
: MQSeries ended after a unit of recovery began to be backed out but before the process was complete (not shown in the figure); during restart, MQSeries continues to back out the changes.

What happens during termination

MQSeries terminates normally in response to the command STOP QMGR. If MQSeries stops for any other reason, the termination is considered to be abnormal.

Normal termination

In a normal termination, MQSeries stops all activity in an orderly way. You can stop MQSeries using either quiesce, force, or restart mode. The effects are given in Table 7:

Table 7. Termination using QUIESCE, FORCE, and RESTART

Thread type	QUIESCE	FORCE	RESTART
Active threads	Run to completion	Back out	Back out
New threads	Can start	Not permitted	Not permitted
New connections	Not permitted	Not permitted	Not permitted

Batch applications are notified if a termination occurs while the application is still connected.

With CICS, a current thread runs only to the end of the unit of recovery. With CICS, stopping MQSeries in quiesce mode stops the CICS adapter, and so if an active task contains more than one unit of recovery, the task does not necessarily run to completion.

If you stop MQSeries in force or restart mode, no new threads are allocated, and work on connected threads is rolled back. Using these modes can create in-doubt units of recovery for threads that are between commit processing phases. They are resolved when MQSeries is reconnected with the controlling CICS, IMS, or RRS subsystem.

When you stop MQSeries, in any mode, the steps are:

1. Connections are ended.
2. MQSeries ceases to accept commands.
3. MQSeries ensures that any outstanding updates to the page sets are completed.
4. The DISPLAY USAGE command is issued internally by MQSeries so that the restart RBA is recorded on the OS/390 console log.
5. The shutdown checkpoint is taken and the BSDS is updated.

Terminations that specify quiesce mode do not affect in-doubt units of recovery. Any unit that is in doubt remains in doubt.

Abnormal termination

An abnormal termination can leave data in an inconsistent state, for example:

- A unit of recovery has been interrupted before reaching a point of consistency.
- Committed data has not been written to page sets.
- Uncommitted data has been written to page sets.
- An application program has been interrupted between phase 1 and phase 2 of the commit process, leaving the unit of recovery in doubt.

MQSeries resolves any data inconsistencies arising from abnormal termination during restart and recovery.

What happens during restart and recovery

MQSeries uses its recovery log and the bootstrap data set (BSDS) to determine what to recover when it restarts. The BSDS identifies the active and archive log data sets, and the location of the most recent MQSeries checkpoint in the log.

After MQSeries has been initialized, the restart process takes place as follows:
- Log initialization
- Current status rebuild
- Forward log recovery
- Backward log recovery
- Queue index rebuilding

When recovery has been completed:
- Committed changes are reflected in the data.
- In-doubt activity is reflected in the data. However, the data is locked and cannot be used until MQSeries recognizes and acts on the in-doubt decision.
- Interrupted in-flight and in-abort changes have been removed from the queues. The messages are consistent and can be used.
- A new checkpoint has been taken.
- New indexes have been built for indexed queues containing persistent messages (described in "Rebuilding queue indexes").

Batch applications are not notified when restart occurs *after* the application has requested a connection.

If dual BSDSs are in use, MQSeries checks the consistency of the time stamps in the BSDS:
- If both copies of the BSDS are current, MQSeries tests whether the two time stamps are equal. If they are not, MQSeries issues message CSQJ120E and terminates. This can happen when the two copies of the BSDS are maintained on separate DASD volumes and one of the volumes was restored while MQSeries was stopped. MQSeries detects the situation at restart.
- If one copy of the BSDS was deallocated, and logging continued with a single BSDS, a problem could arise. If both copies of the BSDS are maintained on a single volume, and the volume was restored, or if both BSDS copies were restored separately, MQSeries might not detect the restoration. In that case, log records not noted in the BSDS would be unknown to the system.

Rebuilding queue indexes

To increase the speed of **MQGET** operations on a queue where messages are not retrieved sequentially, you can specify that you want MQSeries to maintain an index of the message or correlation identifiers for all the messages on that queue (as described in the *MQSeries Application Programming Guide*).

When MQSeries is restarted, these indexes are rebuilt for each queue. This only applies to persistent messages; nonpersistent messages are deleted at restart. If your indexed queues contain large numbers of persistent messages, this will increase the time taken to restart MQSeries.

How in-doubt units of recovery are resolved

If MQSeries loses its connection to CICS, IMS, or RRS, it normally attempts to recover all inconsistent objects at restart. The information needed to resolve in-doubt units of recovery must come from the coordinating system. The next section describes the process of resolution.

How in-doubt units of recovery are resolved from CICS

The resolution of in-doubt units has no effect on CICS resources. CICS is in control of recovery coordination and, when it restarts, automatically commits or backs out each unit, depending on whether there was a log record marking the beginning of the commit. The existence of in-doubt objects does not lock CICS resources while MQSeries is being reconnected.

One of the functions of the CICS adapter is to keep data synchronized between CICS and MQSeries. If MQSeries abends while connected to CICS, it is possible for CICS to commit or back out work without MQSeries being aware of it. When MQSeries restarts, that work is termed *in doubt*.

MQSeries cannot resolve these in-doubt units of recovery (that is, commit or back out the changes made to MQSeries resources) until the connection to CICS is restarted or reconnected.

A process to resolve in-doubt units of recovery is initiated during startup of the CICS adapter. The process starts when the adapter requests a list of in-doubt units of recovery. Then:

- The adapter receives a list of in-doubt units of recovery for this connection ID from MQSeries, and passes them to CICS for resolution.
- CICS compares entries from this list with entries in its own log. CICS determines from its own list what action it took for each in-doubt unit of recovery.

Under some circumstances, CICS cannot run the MQSeries process to resolve in-doubt units of recovery. When this happens, MQSeries sends one of these messages:

- CSQC404E
- CSQC405E
- CSQC406E
- CSQC407E

followed by the message CSQC408I.

For details of what these messages mean, see the *MQSeries for OS/390 Messages and Codes* manual.

For all resolved units, MQSeries updates the queues as necessary and releases the corresponding locks. Unresolved units can remain after restart. Resolve them by the methods described in the *MQSeries for OS/390 System Administration Guide*.

How in-doubt units of recovery are resolved from IMS

Resolving in-doubt units of recovery in IMS has no effect on DL/I resources. IMS is in control of recovery coordination and, when it restarts, automatically commits or backs out incomplete DL/I work. The decision to commit or back out for online regions (non-fast-path) is on the presence or absence of IMS log record types X'3730' and X'3801' respectively. The existence of in-doubt units of recovery does not imply that DL/I records are locked until MQSeries connects.

During restart, MQSeries makes a list of in-doubt units of recovery. IMS builds its own list of residual recovery entries (RREs). The RREs are logged at IMS checkpoints until all entries are resolved.

During reconnection of an IMS region to MQSeries, IMS indicates to MQSeries whether to commit or back out units of work marked in MQSeries as in doubt.

When in-doubt units are resolved:

1. If MQSeries recognizes that it has marked an entry for commit and IMS has marked it to be backed out, MQSeries issues message CSQQ010E. MQSeries issues this message for all inconsistencies of this type between MQSeries and IMS.
2. If MQSeries has any remaining in-doubt units, the adapter issues message CSQQ008I.

For all resolved units, MQSeries updates queues as necessary and releases the corresponding locks.

MQSeries maintains locks on in-doubt work that was not resolved. This can cause a backlog in the system if important locks are being held. The connection remains active so you can resolve the IMS RREs. Recover the in-doubt threads by the methods described in the *MQSeries for OS/390 System Administration Guide*.

All in-doubt work should be resolved unless there are software or operating problems, such as with an IMS cold start. In-doubt resolution by the IMS control region takes place in two circumstances:

1. At the start of the connection to MQSeries, during which resolution is done synchronously.
2. When a program abends, during which the resolution is done asynchronously.

How in-doubt units of recovery are resolved from RRS

One of the functions of the RRS adapter is to keep data synchronized between MQSeries and other RRS-participating resource managers. If a failure occurs when MQSeries has completed phase one of the commit and is waiting for a decision from RRS (the commit coordinator), the unit of recovery enters the in-doubt state.

When communication is reestablished between RRS and MQSeries, RRS automatically commits or backs out each unit of recovery, depending on whether there was a log record marking the beginning of the commit. MQSeries cannot resolve these in-doubt units of recovery (that is, commit or back out the changes made to MQSeries resources) until the connection to RRS is reestablished.

Under some circumstances, RRS cannot resolve in-doubt units of recovery. When this happens, MQSeries sends one of the following messages to the OS/390 console:
- CSQ3011I
- CSQ3013I
- CSQ3014I
- CSQ3016I

For details of what these messages mean, see the *MQSeries for OS/390 Messages and Codes* manual.

For all resolved units of recovery, MQSeries updates the queues as necessary and releases the corresponding locks. Unresolved units of recovery can remain after restart. Resolve them by the method described in the *MQSeries for OS/390 System Administration Guide*.

Shared queue recovery

This section describes MQSeries recovery in the queue-sharing group environment.

Transactional recovery

When an application issues an **MQBACK** call or terminates abnormally (for example, because of an EXEC CICS ROLLBACK or an IMS abend) thread level information stored in the queue manager ensures that the in-flight unit of work is rolled back. **MQPUT** and **MQGET** operations within syncpoint on shared queues are rolled back in the same way as updates to non-shared queues.

Peer recovery

If a queue manager fails, it disconnects abnormally from the Coupling Facility structures that it is currently connected to. If the connection between the OS/390 instance and the Coupling Facility fails (for example, physical link failure or power off of a Coupling Facility or partition) this is also detected as an abnormal termination of the connection between the queue manager and the Coupling Facility structures involved. Other queue managers in the same queue-sharing group that remain connected to that structure detect the abnormal disconnection and all attempt to initiate *peer recovery* for the failed queue manager on that structure. Only one of these queue managers initiates peer recovery successfully, but all the other queue managers cooperate in the recovery of units of work that were owned by the queue manager that failed.

If a queue manager fails when there are no peers connected to a structure, recovery will be performed when another queue manager connects to that structure, or when the queue manager that failed restarts.

Peer recovery is performed on a structure by structure basis and it is possible for a single queue manager to be participating in the recovery of more than one structure at the same time. However, the set of peers cooperating in the recovery of different structures might vary depending on which queue managers were connected to the different structures at the time of failure.

When the failed queue manager restarts, it reconnects to the structures that it was connected to at the time of failure, and recovers any remaining unresolved units of work that were not recovered by peer recovery.

Peer recovery is a multi-phase process. During the first phase, units of work that had progressed beyond the in-flight phase are recovered; this might involve committing messages for units of work that are in-commit and locking messages for units of work that are in-doubt. During the second phase, queues that had threads active against them in the failing queue manager are checked, uncommitted messages related to in-flight units of work are rolled back, and information about active handles on shared queues in the failed queue manager are reset. This means that MQSeries will reset any indicators that the failing queue manager had a shared queue open for input-exclusive, allowing other active queue managers to open the queue for input.

Shared queue definitions

The queue objects that represent the attributes of a shared queue are held in the shared DB2 repository used by the queue-sharing group. You should ensure that adequate procedures are in place for the backup and recovery of the DB2 tables used to hold MQSeries objects. You can also use the MQSeries CSQUTIL utility to create MQSC commands for replay into a queue manager to redefine MQSeries objects, including shared queue and group definitions stored in DB2.

Coupling Facility failure

In the unlikely event of a coupling facility failure, any MQSeries messages stored in the affected structures will be lost. Any queue manager connected to a structure in a failing Coupling Facility will terminate abnormally. Although the administration structure contains no message data, special considerations apply because all queue managers in the queue-sharing group are connected to it. Because information about units of work touching shared queues is stored in the administration structure, an administration structure failure means that units of work can neither commit or back out, so to maintain unit of work integrity, the queue manager must terminate.

When all queue managers connected to a failing structure have terminated, the structure can be reallocated provided there are no failed disconnected connections (these should be purged using the OS/390 command SETXCF FORCE,CONNECTION). The structure is reallocated automatically when a queue manager is restarted and attempts to connect to the structure. For example, if a Coupling Facility structure used to hold shared queues fails it will be reallocated when the queue manager reconnects. The shared queues can be opened immediately and used because all the object definitions are stored in DB2, however any messages that were on the queues before the failure will have been lost.

Where to find more information

You can find more information about the topics discussed in this chapter from the following sources:

Table 8. Where to find more information about recovery and restart

Topic	Where to look
Planning your backup strategy	"Chapter 18. Planning for backup and recovery" on page 131
System parameters	*MQSeries for OS/390 System Setup Guide*
Routine backup and recovery procedures Resolving in-doubt threads Resolving problems during restart Utility programs	*MQSeries for OS/390 System Administration Guide*
Console messages	*MQSeries for OS/390 Messages and Codes*
MQSeries commands	*MQSeries MQSC Command Reference*

Chapter 7. Security

This chapter discusses MQSeries security. It contains the following sections:
- “Why you need to protect MQSeries resources”
- “Security controls and options” on page 62
- “Resources you can protect” on page 64
- “Channel security” on page 67
- “Where to find more information” on page 67

Why you need to protect MQSeries resources

Because MQSeries handles the transfer of information that is potentially valuable, it needs the safeguard of a security system. This is to ensure that the resources MQSeries owns and manages are protected from unauthorized access that might lead to the loss or disclosure of the information. It is essential that none of the following are accessed or changed by any unauthorized user or process:
- Connections to MQSeries
- MQSeries objects such as queues, processes, and namelists
- MQSeries transmission links
- MQSeries system control commands
- MQSeries messages
- Context information associated with messages

To provide the necessary security, MQSeries uses the OS/390 system authorization facility (SAF) to route authorization requests to an External Security Manager (ESM), for example Resource Access Control Facility (RACF). MQSeries does no security verification of its own. Where distributed queuing or clients are being used, additional security measures might be required, for which MQSeries provides channel exits and the MCAUSER channel attribute.

The decision to allow access to an object is made by the ESM and MQSeries follows that decision. If the ESM cannot make a decision, MQSeries prevents access to the object.

If you do nothing

If you do nothing about security, the most likely effect is that ***all*** users can access and change ***every*** resource. This includes not only local users, but also those on remote systems using distributed queuing or clients, where the logon security controls might be less strict than is normally the case for OS/390.

In order to enable security checking you must do the following:
- Install and activate an ESM (for example, RACF)
- Define the MQADMIN class if you are using an ESM other than RACF
- Activate the MQADMIN class

Security controls and options

You can specify whether security is turned on for the whole MQSeries subsystem, and whether you want to perform security checks at queue manager or queue-sharing group level. You can also control the number of user IDs checked for API-resource security.

Subsystem security

Subsystem security is a control that specifies whether any security checking is done on the whole MQSeries subsystem. If you do not require security checking (for example, on a test system), or if you are satisfied with the level of security on all the resources that can connect to MQSeries (including clients and channels), you can turn security checking off for the queue manager or queue-sharing group so that no further security checking takes place.

This is the only check that can turn security off completely and determine whether any other security checks are performed or not. That is, if you turn off checking for the queue manager or queue-sharing group, no other MQSeries checking is done; if you leave it turned on, MQSeries checks your security requirements for other MQSeries resources.

Queue manager or queue-sharing group level checking

Security can be implemented at queue manager level or at queue-sharing group level. If you implement security at queue-sharing group level, all the queue managers in the group share the same profiles. This means that there are fewer profiles to define and maintain, making security management easier.

It is also possible to implement a combination of both if your installation requires it, for example, during migration or if you have one queue manager in the queue-sharing group that requires different levels of security to the other queue managers in the group.

Queue-sharing group level security

Queue-sharing group level security checking is performed for the entire queue-sharing group. It enables you to simplify security administration because it requires you to define fewer security profiles. The authorization of a user ID to use a particular resource is handled at the queue-sharing group level, and is independent of which queue manager that user ID is using to access the resource.

For example, say a server application runs under user ID SERVER and wants access to a queue called SERVER.REQUEST, and you want to run an instance of SERVER on each OS/390 image in the sysplex. Rather than permitting SERVER to open SERVER.REQUEST on each queue manager individually (queue manager level security), you can permit access once at the queue-sharing group level.

Queue-sharing group level security profiles can be used to protect all types of resource, whether local or shared.

Queue manager level security

Queue manager level security checking is performed at subsystem level using security profiles specific to that queue manager. This is how security is managed on Version 2.1 of MQSeries for OS/390, or earlier.

Queue manager level security profiles can be used to protect all types of resource, whether local or shared.

Combination of both levels

You can use a combination of both queue manager and queue-sharing group level security.

You can override queue-sharing group level security settings for a particular queue manager that is a member of that group. This means that you can perform a different level of security checks on an individual queue manager to those performed on the other queue managers in the group.

Controlling the number of user IDs checked

RESLEVEL is a RACF profile that controls the number of user IDs checked for MQSeries resource security. Normally, when a user attempts to access an MQSeries resource, RACF checks the relevant user ID or IDs to see if access is allowed to that resource. By defining a RESLEVEL profile you can control whether zero, one or, where applicable, two user IDs are checked.

These controls are done on a connection by connection basis, and last for the life of the connection.

There is only one RESLEVEL profile per queue manager. Control is implemented by the access that a user ID has to this profile.

Resources you can protect

When MQSeries starts, or when instructed by an operator command, MQSeries determines which resources you want to protect. You can control which security checks are performed for each individual queue manager. For example, you could implement a number of security checks on a production queue manager, but none on a test queue manager.

Connection security

Connection security checking is carried out either when an application program tries to connect to a queue manager by issuing an **MQCONN** or **MQCONNX** request, or when the channel initiator, or CICS or IMS adapter issues a connection request.

If you are using queue manager level security, you can turn connection security checking off for a particular MQSeries subsystem, but if you do any user can connect to that subsystem.

For the CICS adapter, only the CICS address space user ID is used for the connection security check—not the individual CICS terminal user ID. For the IMS adapter, when the IMS control or dependent regions connect to MQSeries, the IMS address space user ID is checked. For the channel initiator, the user ID used by the channel initiator address space is checked.

Connection security checking can be turned on or off at either queue manager or queue-sharing group level.

API-resource security

Resources are checked when an application opens an object with an **MQOPEN** or an **MQPUT1** call. The access needed to open an object depends on what open options are specified when the queue is opened.

API-resource security is subdivided into these checks:
- Queue
- Process
- Namelist
- Alternate user
- Context

No security checks are performed when opening the queue manager object or when accessing storage class objects.

Queue security

Queue security checking controls who is allowed to open which queue, and what options they are allowed to open it with. For example, a user might be allowed to open a queue called PAYROLL.INCREASE.SALARY to browse the messages on the queue (via the MQOO_BROWSE option), but not to remove messages from the queue (via one of the MQOO_INPUT_* options). If you turn checking for queues off, any user can open any queue with any valid open option (that is, any valid MQOO_* option on an **MQOPEN** or **MQPUT1** call).

Queue security checking can be turned on or off at either queue manager or queue-sharing group level.

Process security

Process security checking is carried out when a user opens a process definition object. If you turn checking for processes off, any user can open any process.

Process security checking can be turned on or off at either queue manager or queue-sharing group level.

Namelist security

Namelist security checking is carried out when a user opens a namelist. If you turn checking for namelists off, any user can open any namelist.

Namelist security checking can be turned on or off at either queue manager or queue-sharing group level.

Alternate user security

Alternate user security controls whether one user ID can use the authority of another user ID to open an MQSeries object.

For example:

- A server program running under user ID PAYSERV retrieves a request message from a queue that was put on the queue by user ID USER1.
- When the server program gets the request message, it processes the request and puts the reply back into the reply-to queue specified with the request message.
- Instead of using its own user ID (PAYSERV) to authorize opening the reply-to queue, the server can specify some other user ID, in this case, USER1. In this example, alternate user security would control whether user ID PAYSERV is allowed to specify user ID USER1 as an alternate user ID when opening the reply-to queue.

The alternate user ID is specified in the `AlternateUserId` field of the object descriptor (MQOD).

You can use alternate user IDs on any MQSeries object, for example, processes or namelists. It does not affect the user ID used by any other resource managers, for example, for CICS security or for OS/390 data set security.

If alternate user security is not active, any user can use any other user ID as an alternate user ID.

Alternate user security checking can be turned on or off at either queue manager or queue-sharing group level.

Context security

Context is information that is applicable to a particular message and is contained in the message descriptor (MQMD) that is part of the message. The context information comes in two sections:

Identity section

The user of the application that first put the message to a queue. It consists of the following fields:

- *UserIdentifier*
- *AccountingToken*
- *ApplIdentityData*

Origin section
The application that put the message on the queue where it is currently stored. It consists of the following fields:
- *PutApplType*
- *PutApplName*
- *PutDate*
- *PutTime*
- *ApplOriginData*

Applications can specify the context data when either an **MQPUT** or an **MQPUT1** call is made. This data might be generated by the application, it might be passed on from another message, or it might be generated by the queue manager by default. For example, context data can be used by server programs to check the identity of the requester, that is, did this message come from the correct application? Typically, the *UserIdentifier* field is used to determine the user ID of an alternate user.

You use context security to control whether the user can specify any of the context options on any **MQOPEN** or **MQPUT1** call. For information about the context options, see the *MQSeries Application Programming Guide*; for descriptions of the message descriptor fields relating to context, see the *MQSeries Application Programming Reference* manual.

If you turn context security checking off, any user can use any of the context options that the queue security allows.

Context security checking can be turned on or off at either queue manager or queue-sharing group level.

Command security

Command security checking is carried out when a user issues an MQSeries command from any of the sources described in “Issuing commands” on page 73. A separate check can be made on the resource specified by the command as described in “Command resource security” on page 67.

If you turn off command checking, issuers of commands are not checked to see whether they have the authority to issue the command.

If MQSeries commands are entered from a console, the console must have the OS/390 SYS console authority attribute. Commands that are issued from the CSQINP1 or CSQINP2 data sets, or internally by the queue manager, are exempt from all security checking while those for CSQINPX use the user ID of the channel initiator address space. You should control who is allowed to update these data sets through normal data set protection.

Command security checking can be turned on or off at either queue manager or queue-sharing group level.

Command resource security

Some MQSeries commands, for example defining a local queue, involve the manipulation of MQSeries resources. When command resource security is active, each time a command involving a resource is issued, MQSeries checks to see if the user is allowed to change the definition of that resource.

You can use command resource security to help enforce naming standards. For example, a payroll administrator might be allowed to delete and define only queues with names beginning “PAYROLL”. If command resource security is inactive, no security checks are made on the resource that is being manipulated by the command. Do not confuse command resource security with command security; the two are independent.

Turning off command resource security checking does not affect the resource checking that is done specifically for other types of processing that do not involve commands.

Command resource security checking can be turned on or off at either queue manager or queue-sharing group level.

Channel security

When you are using channels, the security features available depend on which communications protocol you are going to use. If you use TCP, there are no security features provided with the communications protocol. If you are using APPC, you can flow user ID information from the application that first puts a message, through the network to the destination application for verification.

For both protocols, you can specify which user IDs you want to check for security purposes, and how many. Again, the choices available to you depend on which protocol you are using, what you specify when you define the channel, and the RESLEVEL settings for the channel initiator.

You can also write your own security exit programs to be called by the MCA.

Where to find more information

You can find more information about the topics discussed in this chapter from the following sources:

Table 9. Where to find more information about security

Topic	**Where to look**
Setting up your security Channel security	*MQSeries for OS/390 System Setup Guide*
Console messages	*MQSeries for OS/390 Messages and Codes*
MQSeries commands Defining channels	*MQSeries MQSC Command Reference*
Channel security exits	*MQSeries Intercommunication*

Chapter 8. Availability

There are several features of MQSeries that are designed to increase system availability if the queue manager or channel initiator fails. These are discussed in the following sections:

- "Shared queues"
- "Shared channels" on page 70
- "Using the OS/390 Automatic Restart Manager (ARM)" on page 71
- "Using the OS/390 Extended Recovery Facility (XRF)" on page 72

Shared queues

In the queue-sharing group environment, an application can connect to any of the queue managers within the queue-sharing group. Because all the queue managers in the queue-sharing group can access the same set of shared queues, the application does not depend on the availability of a particular queue manager; any queue manager in the queue-sharing group can service any queue. This gives greater availability if a queue manager stops because all the other queue managers in the queue-sharing group can continue processing the queue. For information about high availability of shared queues, see "Advantages of using shared queues" on page 16.

To further enhance the availability of messages in a queue-sharing group, MQSeries detects if another queue manager in the group disconnects from the Coupling Facility abnormally, and completes units of work for that queue manager that are still pending, where possible. This is known as *peer recovery* and is described in "Peer recovery" on page 59.

Peer recovery cannot recover units of work that were in doubt at the time of the failure. You can use the Automatic Restart Manager (ARM) to restart all the systems involved in the failure (CICS, DB2, and MQSeries for example), and to ensure that they are all restarted on the same new processor. This means that they can resynchronize, and gives rapid recovery of in-doubt units of work. This is described in "Using the OS/390 Automatic Restart Manager (ARM)" on page 71.

Shared channels

In the queue-sharing group environment, MQSeries provides functions that give high availability to the network. The channel initiator enables you to use networking products that balance network requests across a set of eligible servers and hide server failures from the network (for example, VTAM generic resources). MQSeries uses a generic port for inbound requests so that attach requests can be routed to any available channel initiator in the queue-sharing group. This is described in "Shared channels" on page 18.

Shared outbound channels take the messages they send from a shared transmission queue. Information about the status of a shared channel is held in one place for the whole queue-sharing group level. This means that a channel can be restarted automatically on a different channel initiator in the queue-sharing group if the channel initiator, queue manager, or communications subsystem fails. This is called *peer channel recovery* and is described in "Shared outbound channels" on page 19.

Using the OS/390 Automatic Restart Manager (ARM)

MQSeries for OS/390 can be used in conjunction with the OS/390 automatic restart manager (ARM). If a queue manager or a channel initiator has failed, ARM will restart it on the same OS/390 image. If OS/390 fails, a whole group of related subsystems and applications will also fail. ARM can restart all the failed systems automatically, in a predefined order, on another OS/390 image within the sysplex. This is called a cross-system restart.

ARM enables rapid recovery of in-doubt transactions in the shared queue environment. It also gives higher availability if you are not using queue-sharing groups.

You can use ARM to restart an MQSeries subsystem that uses LU 6.2 communication protocols on a different OS/390 image within the sysplex in the event of OS/390 failure. (You cannot do this if you use TCP communication protocols.)

To enable automatic restart:

- You must set up an ARM coupling data set.
- You must define the automatic restart actions that you want OS/390 to perform in an *ARM policy*.
- The ARM policy must be started.
- The subsystem must register with ARM at startup.

If you want to restart queue managers in different OS/390 images automatically, every queue manager must be defined in each OS/390 image on which that queue manager might be restarted, with a sysplex-wide unique 4-character subsystem name.

Using ARM with MQSeries is described in the *MQSeries for OS/390 System Administration Guide*.

Using the OS/390 Extended Recovery Facility (XRF)

MQSeries can be used in an extended recovery facility (XRF) environment. All MQSeries-owned data sets (executable code, BSDSs, logs, and page sets) must be on DASD shared between the active and alternate XRF processors.

If you use XRF for recovery, you must stop MQSeries on the active processor and start it on the alternate. For CICS, this can be done using the command list table (CLT) provided by CICS, or manually by the system operator. For IMS, this is a manual operation and must be done after the coordinating IMS system has completed the processor switch.

MQSeries utilities must be completed or terminated before MQSeries can be switched to the alternate processor. Consider the effect of this potential interruption carefully when planning your XRF recovery plans.

Take care to prevent MQSeries starting on the alternate processor before the MQSeries system on the active processor terminates. A premature start can cause severe integrity problems in data, the catalog, and the log. Using global resource serialization (GRS) helps avoid the integrity problems by preventing simultaneous use of MQSeries on the two systems. The BSDS must be included as a protected resource, and the active and alternate XRF processors must be included in the GRS ring.

Where to find more information

You can find more information about the topics discussed in this chapter from the following sources:

Table 10. Where to find more information about availability

Topic	Where to look
Queue-sharing groups	"Chapter 2. Shared queues and queue-sharing groups" on page 11
System parameters	*MQSeries for OS/390 System Setup Guide*
Using the Automatic Restart Manager Utility programs	*MQSeries for OS/390 System Administration Guide*
Console messages	*MQSeries for OS/390 Messages and Codes*
MQSeries commands	*MQSeries MQSC Command Reference*

Chapter 9. Creating and managing objects

This chapter discusses how to use the MQSeries commands and utilities to create objects and manage your queue managers. It includes the following sections:

- "Issuing commands"
- "The MQSeries for OS/390 utilities" on page 79
- "Where to find more information" on page 81

Issuing commands

MQSeries for OS/390 supports MQSC commands, which can be issued from the following sources:

- The OS/390 console or equivalent (such as SDSF/TSO).
- The initialization input data sets.
- The supplied batch utility, CSQUTIL, processing a list of commands in a sequential data set.
- A suitably authorized application, by sending a command as a message to the command queue. This can be either:
  - A batch region program
  - A CICS application
  - An IMS application
  - A TSO application
  - An application program or utility on another MQSeries system

Much of the functionality of these commands is available in a user-friendly way from the MQSeries for OS/390 operations and controls panels.

Changes made to the resource definitions of a queue manager using the commands (directly or indirectly) are preserved across restarts of the MQSeries subsystem.

Private and global definitions

When you define an object on MQSeries for OS/390, you can choose whether you want to share that definition with other queue managers (a *global* definition), or whether the object definition is to be used by one queue manager only (a *private* definition). This is called the object *disposition*.

Global definition

If your queue manager belongs to a queue-sharing group, you can choose to share any object definitions you make with the other members of the group. This means that an object has to be defined once only, reducing the total number of definitions required for the whole system.

Global object definitions are held in a *shared repository* (a DB2 shared database), and are available to all the queue managers in the queue-sharing group. These objects have a disposition of GROUP.

Private definition

If you want to create an object definition that is required by one queue manager only, or if your queue manager is not a member of a queue-sharing group, you can create object definitions that are not shared with other members of a queue-sharing group.

Private object definitions are held on page set zero of the defining queue manager. These objects have a disposition of QMGR.

You can create private definitions for all six types of MQSeries object (channels, namelists, process definitions, queues, queue managers, and storage class definitions), and global definitions for all types of object except queue managers.

MQSeries automatically copies the definition of a group object to page set zero of each queue manager that uses it. You can alter the copy of the definition temporarily if you want, and MQSeries allows you to refresh the page set copies from the repository copy if required. MQSeries always tries to refresh the page set copies from the repository copy on start up (for channel commands, this is done when the channel initiator restarts). This ensures that the page set copies reflect the version on the repository, including any changes that were made when the queue manager was inactive. There are circumstances under which the refresh is not performed, for example:

- If a copy of the queue is open, a refresh that changes the usage of the queue will fail.
- If a copy of a queue has messages on it, a refresh that deletes that queue will fail.

In these circumstances, the refresh is not performed on that copy, but is performed on the copies on all other queue managers.

If the queue manager is shut down and then restarted stand-alone, any local copies of objects are deleted, unless for example, the queue has associated messages.

There is a third object disposition that applies to local queues only. This allows you to create shared queues. The definition for a shared queue is held on the shared repository and is available to all the queue managers in the queue-sharing group. In addition, the messages on a shared queue are also available to all the queue managers in the queue sharing group. This is described in "Chapter 2. Shared queues and queue-sharing groups" on page 11. Shared queues have an object disposition of SHARED.

The following table summarizes the effect of the object disposition options for queue managers started stand-alone, and as a member of a queue-sharing group.

Disposition	Stand-alone queue manager	Member of a queue-sharing group
QMGR	Object definition held on page set zero.	Object definition held on page set zero.
GROUP	Not allowed.	Object definition held in the shared repository. Local copy held on page set zero of each queue manager in the group.
SHARED	Not allowed.	Queue definition held in the shared repository. Messages available to any queue manager in the group.

Manipulating global definitions

If you want to change the definition of an object that is held in the shared repository, you need to specify whether you want to change the version on the repository, or the local copy on page set zero. Use the object disposition as part of the command to do this.

Directing commands to different queue managers

You can choose to execute a command on the queue manager where it is entered, or on a different queue manager in the queue-sharing group. You can also choose to issue a particular command in parallel on all the queue managers in a queue-sharing group.

This is determined by the *command scope*. The command scope is used in conjunction with the object disposition to determine which version of an object you want to work with.

For example, you might want to alter some of the attributes of an object, the definition of which is held in the shared repository.

- You might want to change the version on one queue manager only, and not make any changes to the version on the repository or those in use by other queue managers.
- You might want to change the version in the shared repository for future users, but leave existing copies unchanged.
- You might want to change the version in the shared repository, but also want your changes to be reflected immediately on all the queue managers in the queue-sharing group that hold a copy of the object on their page set zero.

Use the command scope to specify whether the command is executed on this queue manager, another queue manager, or all queue managers. Use the object disposition to specify whether the object you are manipulating is in the shared repository (a group object), or is a local copy on page set zero (a queue manager object).

You do not have to specify the command scope and object disposition to work with a shared queue because every queue manager in the queue-sharing group sees the shared queue as a single queue.

Administrator commands

The following tables summarize the commands that are available on MQSeries for OS/390 to manage MQSeries objects.

Table 11. Summary of MQSeries administrator commands

	ALTER	CLEAR	DEFINE	DISPLAY	DELETE	MOVE
CHANNEL	✓		✓	✓	✓	
NAMELIST	✓		✓	✓	✓	
PROCESS	✓		✓	✓	✓	
QALIAS	✓		✓	✓	✓	
QCLUSTER				✓		
QLOCAL	✓	✓	✓	✓	✓	✓
QMGR	✓			✓		
QMODEL	✓		✓	✓	✓	
QREMOTE	✓		✓	✓	✓	
QUEUE				✓		
STGCLASS	✓		✓	✓	✓	

System control commands

You can use the system control commands to manage other MQSeries resources, such as page sets and buffer pools. Table 12 summarizes the MQSeries system control commands.

Table 12. System control commands

Resource	Task	Command
BSDS	Re-establish a dual bootstrap data set that had a data set error	RECOVER BSDS
Buffer pools	Define a buffer pool and the number of 4 KB buffers that it contains	DEFINE BUFFPOOL
Channels	Start a channel	START CHANNEL
	Stop a channel	STOP CHANNEL
	Test a channel	PING CHANNEL
	Reset channel sequence numbers	RESET CHANNEL
	Resolve in-doubt messages on a channel	RESOLVE CHANNEL
	Display channel status information	DISPLAY CHSTATUS
Channel initiators	Start a channel initiator	START CHINIT
	Stop a channel initiator	STOP CHINIT
	Display information about channel initiators	DISPLAY DQM
Channel listeners	Start a channel listener	START LISTENER
	Stop a channel listener	STOP LISTENER
Clusters	Refresh locally-held cluster information	REFRESH CLUSTER
	Perform special cluster operations	RESET CLUSTER
	Join a cluster	RESUME QMGR
	Leave a cluster	SUSPEND QMGR
	Display cluster information about the queue managers in a cluster	DISPLAY CLUSQMGR
Command servers	Start the command server	START CMDSERV
	Stop the command server	STOP CMDSERV
	Display command server attributes	DISPLAY CMDSERV
IMS Tpipes	Reset sequence numbers for an IMS transaction pipe	RESET TPIPE
Logs	Copy the current active log to an archive log	ARCHIVE LOG
	Set logging parameters	SET LOG
	Display logging parameters	DISPLAY LOG
Page sets	Define a page set and an associated buffer pool	DEFINE PSID
	Display the current state of a page set	DISPLAY USAGE
Queues	Display who is using a queue	DISPLAY QSTATUS
	Display and reset queue statistics	RESET QSTATS

Table 12. System control commands (continued)

Resource	Task	Command
Queue managers	Start a queue manager	START QMGR
	Stop a queue manager	STOP QMGR
	Define the maximum number of messages that a task can get or put within a single unit of recovery	DEFINE MAXSMSGS
	Display the setting of MAXSMSGS	DISPLAY MAXSMSGS
Queue-sharing groups	Display information about a queue-sharing group	DISPLAY GROUP
Security	Refresh in-storage ESM tables	REFRESH SECURITY
	Request security reverification for a user	RVERIFY SECURITY
	Change security settings	ALTER SECURITY
	Display security settings	DISPLAY SECURITY
Threads	Display information about threads	DISPLAY THREAD
	Resolve in-doubt units of recovery manually	RESOLVE INDOUBT
Traces	Start an MQSeries trace	START TRACE
	Stop an MQSeries trace	STOP TRACE
	Alter MQSeries trace settings	ALTER TRACE
	Display MQSeries trace settings	DISPLAY TRACE

Initialization commands

Commands in the initialization input data sets are processed when MQSeries is initialized on MQSeries startup. Three types of command can be issued from the initialization input data sets:

- Commands to define MQSeries entities that cannot be recovered (DEFINE BUFFPOOL and DEFINE PSID for example)

  These commands must reside in the data set identified by the DDname CSQINP1. They are processed before the restart phase of initialization. They cannot be issued through the console, operations and control panels, or an application program. The responses to these commands are written to the sequential data set that you refer to in the CSQOUT1 statement of the started task procedure.
- Commands to define MQSeries objects that are recoverable after restart. These definitions must be specified in the data set identified by the DDname CSQINP2. They are stored in page set zero. CSQINP2 is processed after the restart phase of initialization. The responses to these commands are written to the sequential data set that you refer to in the CSQOUT2 statement of the started task procedure.
- Commands to manipulate MQSeries objects. These commands must also be specified in the data set identified by the DDname CSQINP2. For example, the MQSeries-supplied data set CSQ4INP2 contains an ALTER QMGR command to specify a dead-letter queue for the subsystem. The response to these commands is written to the CSQOUT2 output data set.

Note: If MQSeries objects are defined in CSQINP2, MQSeries attempts to redefine them each time the MQSeries subsystem is started. If the queues already exist, the attempt to define them fails. If you need to define your objects in CSQINP2, you can avoid this problem by using the REPLACE parameter of the DEFINE commands, however, this will override any changes that were made during the previous run of the queue manager.

Sample initialization data set members are supplied with MQSeries for OS/390. They are described in “Sample definitions supplied with MQSeries” on page 45.

Initialization commands for distributed queuing

You can also use the CSQINP2 initialization data set for the START CHINIT command, and follow it with a series of other commands to define your distributed queuing environment (for example, defining your channels). If you stop and restart the channel initiator however, CSQINP2 is not reprocessed, so MQSeries provides a third initialization input data set, called CSQINPX, that you can choose to process as part of the channel initiator started task procedure.

The MQSC commands contained in the data set are executed at the end of channel initiator initialization, and output is written to the data set specified by the CSQOUTX DD statement. You might use the CSQINPX initialization data set to start listeners for example.

A sample channel initiator initialization data set member is supplied with MQSeries for OS/390. It is described in “Sample definitions supplied with MQSeries” on page 45.

The MQSeries for OS/390 utilities

MQSeries for OS/390 supplies a set of utility programs to help you perform various administrative tasks. These utilities:
- Perform backup, restoration, and reorganization tasks
- Issue commands and process object definitions
- Generate data-conversion exits
- Modify the bootstrap data set
- List information about the logs
- Print the logs
- Set up DB2 tables and other DB2 utilities
- Process messages on the dead-letter queue

The CSQUTIL utility

The CSQUTIL utility program is provided with MQSeries for OS/390 to help you perform backup, restoration, and reorganization tasks, and to issue commands and process object definitions. Through this utility program, you can invoke the following functions:

COMMAND
To issue MQSC commands, to record object definitions, and to make client-channel definition files.

COPY To read the contents of a named MQSeries for OS/390 message queue or the contents of all the queues of a named page set, and put them into a sequential file and retain the original queue.

COPYPAGE
To copy whole page sets to larger page sets.

EMPTY
To delete the contents of a named MQSeries for OS/390 message queue or the contents of all the queues of a named page set, retaining the definitions of the queues.

FORMAT
To format MQSeries for OS/390 page sets.

LOAD
To restore the contents of a named MQSeries for OS/390 message queue or the contents of all the queues of a named page set from a sequential file created by the COPY function.

RESETPAGE
To copy whole page sets to other page set data sets and reset the log information in the copy.

SCOPY
To copy the contents of a queue to a data set while the queue manager is offline.

SDEFS
To produce a set of define commands for objects while the queue manager is offline.

The data conversion exit utility

The MQSeries for OS/390 data conversion exit utility (CSQUCVX) runs as a stand-alone utility to create data conversion exit routines.

The change log inventory utility

The MQSeries for OS/390 change log inventory utility program (CSQJU003) runs as a stand-alone utility to change the bootstrap data set (BSDS). The utility can be used to:

- Add or delete active or archive log data sets
- Supply passwords for archive logs

The print log map utility

The MQSeries for OS/390 print log map utility program (CSQJU004) runs as a stand-alone utility to list the following information:

- Log data set name and log RBA association for both copies of all active and archive log data sets. If dual logging is not active, there is only one copy of the data sets.
- Active log data sets available for new log data.
- Contents of the queue of checkpoint records in the bootstrap data set (BSDS).
- Contents of the archive log command history record.
- System and utility time stamps.

The log print utility

The log print utility program (CSQ1LOGP) is run as a stand-alone utility. You can run the utility specifying:

- A bootstrap data set (BSDS)
- Active logs (with no BSDS)
- Archive logs (with no BSDS)

The queue-sharing group utility

The queue-sharing group utility program (CSQ5PQSG) runs as a stand-alone utility to set up DB2 tables and perform other DB2 tasks required for queue-sharing groups.

The dead-letter queue handler utility

The dead-letter queue handler utility program (CSQUDLQH) runs as a stand-alone utility. It checks messages that are on the dead-letter queue and processes them according to a set of rules that you supply to the utility.

Where to find more information

You can find more information about the topics discussed in this chapter from the following sources:

Table 13. Where to find more information about creating and managing objects

Topic	Where to look
Using initialization input data sets System parameter macros Installation verification program	*MQSeries for OS/390 System Setup Guide*
Writing administration programs Operations and control panels Utility programs	*MQSeries for OS/390 System Administration Guide*
Queue indexes MQSeries commands	*MQSeries MQSC Command Reference*
MQSeries clusters	*MQSeries Queue Manager Clusters*
MQSeries events	*MQSeries Event Monitoring*

Chapter 10. Monitoring and statistics

MQSeries supplies facilities for monitoring the system and collecting statistics. These are discussed in the following sections:

- "MQSeries trace"
- "Events"
- "Where to find more information" on page 84

MQSeries trace

MQSeries supplies a trace facility that can be used to gather the following information while MQSeries is running:

Performance statistics
The statistics trace gathers the following information to help you monitor performance and tune your system:

- Counts of different MQI requests (message manager statistics)
- Counts of different object requests (data manager statistics)
- Information about DB2 usage (DB2 manager statistics)
- Information about Coupling Facility usage (Coupling Facility manager statistics)
- Information about buffer pool usage (buffer manager statistics)
- Information about logging (log manager statistics)
- Information about storage usage (storage manager statistics)
- Information about lock requests (lock manager statistics)

Accounting data

- The accounting trace gathers information about the CPU time spent processing MQI calls and about the number of **MQPUT** and **MQGET** requests made by a particular user.
- MQSeries can also gather information about each task using MQSeries. This data is gathered as a thread-level accounting record. For each thread, MQSeries also gathers information about each queue used by that thread.

The data generated by the trace is sent to the System Management Facility (SMF) or the generalized trace facility (GTF).

Events

MQSeries events provide information about errors, warnings, and other significant occurrences in a queue manager. By incorporating these events into your own system management application, you can monitor the activities across many queue managers, for multiple MQSeries applications. In particular, you can monitor all the queue managers in your system from a single queue manager.

Events can be reported through a user-written reporting mechanism to an administration application that supports the presentation of the events to an operator. Events also enable applications acting as agents for other administration networks, for example NetView, to monitor reports and create the appropriate alerts.

Where to find more information

You can find more information about the topics discussed in this chapter from the following sources:

Table 14. Where to find more information about monitoring and statistics

Topic	**Where to look**
MQSeries trace	*MQSeries for OS/390 System Setup Guide*
Trace commands	*MQSeries MQSC Command Reference*
MQSeries events	*MQSeries Event Monitoring*

Part 3. MQSeries and other products

Chapter 11. MQSeries and CICS

This chapter discusses how MQSeries works with CICS. The CICS adapter and the CICS bridge allow you to connect your MQSeries subsystem to CICS.

- The CICS adapter enables CICS applications to use the MQI.
- The CICS bridge enables applications to run a CICS program or transaction that does not use the MQI. This means that you can use your legacy applications with MQSeries, without the need to rewrite them.

These topics are described in the following sections:

- "The CICS adapter"
- "The CICS bridge" on page 93
- "Where to find more information" on page 97

The CICS adapter

The CICS adapter connects a CICS subsystem to an MQSeries subsystem, enabling CICS application programs to use the MQI.

You can start and stop CICS and MQSeries independently, and you can establish or terminate a connection between them at any time. You can also allow CICS to connect to MQSeries automatically.

The CICS adapter provides two main facilities:

- A set of control functions for use by system programmers and administrators to manage the adapter.
- MQI support for CICS applications.

In a CICS multiregion operation or intersystem communication (ISC) environment, each CICS address space can have its own attachment to the queue manager subsystem. A single CICS address space can be connected to only one queue manager at a time. However, multiple CICS address spaces can connect to the same MQSeries subsystem.

You can use MQSeries with the CICS Extended Recovery Facility (XRF) to aid recovery from a CICS error.

The CICS adapter is supplied with MQSeries and runs as a CICS External Resource Manager. MQSeries also provides CICS transactions to manage the interface.

The CICS adapter uses standard CICS command-level services where required, for example, EXEC CICS WAIT and EXEC CICS ABEND. A portion of the CICS adapter runs under the control of the transaction issuing the messaging requests. Therefore, these calls for CICS services appear to be issued by the transaction.

Control functions

The CICS adapter's control functions (the CKQC transaction) let you manage the connections between CICS and MQSeries dynamically. These functions can be invoked using the CICS adapter panels, from the command line, or from a CICS application. You can use the adapter's control function to:

- Start or stop a connection to a queue manager.
- Modify the current connection. For example, you can reset the connection statistics, change the adapter's trace ID number, and enable or disable the API-crossing exit.
- Display the current status of a connection and the statistics associated with that connection.
- Start or stop an instance of the task initiator transaction, CKTI. ("Task initiator transaction" is CICS terminology; in MQSeries terminology, this is a trigger monitor.)
- Display details of the current instances of CKTI.
- Display details of the CICS tasks currently using the adapter.

These functions and the different methods of invoking them are described in the *MQSeries for OS/390 System Administration Guide*.

MQI support

The CICS adapter implements the MQI for use by CICS application programs. The MQI calls, and how they are used, are described in the *MQSeries Application Programming Guide*. The adapter also supports an *API-crossing exit*, (see "The API-crossing exit" on page 91), and a trace facility.

All application programs that run under CICS must have the supplied *API stub program* called CSQCSTUB link-edited with them if they are to access MQSeries, unless the program is using dynamic calls. This stub provides the application with access to all MQI calls. (For information about calling the CICS stub dynamically, see the *MQSeries Application Programming Guide*.)

For performance, the CICS adapter can handle up to eight MQI calls concurrently. For transaction integrity, the adapter supports syncpointing under the control of the CICS syncpoint manager, so that units of work can be committed or backed out as required. The adapter also supports security checking of MQSeries resources when used with an appropriate security management product, such as RACF. The adapter provides high availability with automatic reconnection after an MQSeries termination, and automatic resource resynchronization after a restart. It also features an alert monitor that responds to unscheduled events such as a shut down of the MQSeries subsystem.

Adapter components

Figure 15 on page 89 shows the relationship between CICS, the CICS adapter, and an MQSeries subsystem. CICS and the adapter share the same address space; the MQSeries for OS/390 is a separate OS/390 subsystem, executing in its own address space.

Part of the adapter is a CICS task-related user exit that communicates with the MQSeries message manager. CICS management modules call the exit directly; application programs call it through the supplied API stub program (CSQCSTUB). Task-related user exits and stub programs are described in the *CICS Customization Guide*.

Each CKTI transaction is normally in an **MQGET WAIT** state, ready to respond to any trigger messages that are placed on its initiation queue.

The adapter management interface provides the operation and control functions described in the *MQSeries for OS/390 System Administration Guide*.

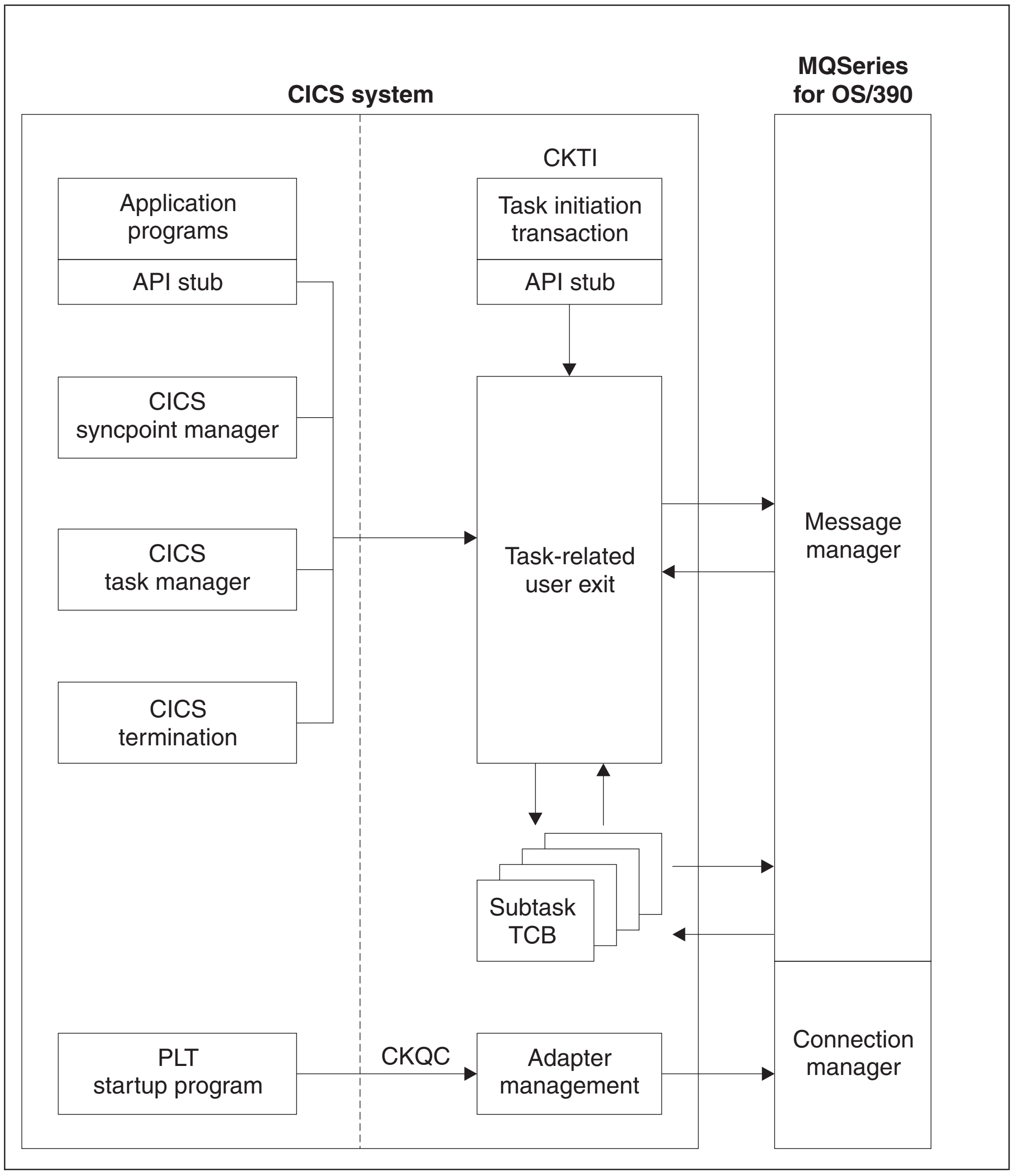


Figure 15. How CICS, the CICS adapter, and an MQSeries subsystem are related

Alert monitor

The *alert monitor* transaction, CKAM, handles unscheduled events—known as *pending events*—that occur as a result of connect requests to instances of MQSeries. The alert monitor generates messages that are sent to the system console.

There are two kinds of pending events:

1. **Deferred connection**

   If CICS tries to connect to MQSeries before MQSeries is started, a *pending event* called a *deferred connection* is activated. When MQSeries is started, a connection request is issued by the CICS adapter, a connection is made, and the pending event is canceled.

   There can be multiple deferred connections, one of which will be connected when MQSeries is started.

2. **Termination notification**

   When a connection is successfully made to MQSeries, a pending event called *termination notification* is created. This pending event expires when:

   - MQSeries shuts down normally with MODE(QUIESCE). The alert monitor issues a quiesce request on the connection.
   - MQSeries shuts down with MODE(FORCE) or terminates abnormally. After an abnormal termination, the CICS adapter waits for ten seconds and then tries a connect call. This enables the CICS system to be automatically reconnected to the queue manager when the latter is restarted.
   - The connection is shut down from the CKQC transaction.

The maximum number of pending events that can be handled is 99. If this limit is reached, no more events can be created until at least one current event expires.

The alert monitor terminates itself when all pending events have expired. It is subsequently restarted automatically by any new connect request.

Auto-reconnect

When CICS is connected to MQSeries and MQSeries terminates, the CICS adapter tries to issue a connect request ten seconds after the stoppage has been detected. This request uses the same connect parameters that were used in the previous connect request. If MQSeries has not been restarted within the ten seconds, the connect request is deferred until MQSeries is restarted later.

Task initiator

CKTI is an MQSeries-supplied CICS transaction that starts a CICS transaction when an MQSeries trigger message is read, for example when a message is put onto a specific queue.

When a message is put onto an application message queue, a trigger is generated if the trigger conditions are met. The queue manager then writes a message, containing user-defined data, known as a *trigger message*, to the initiation queue that has been specified for that message queue. In a CICS environment, an instance of CKTI can be set up to monitor an initiation queue and to retrieve the trigger messages from it as they arrive. CKTI starts another CICS transaction, (specified using the DEFINE PROCESS command), which typically reads the message from the application message queue and then processes it. The process must be named on the application queue definition, not the initiation queue.

Each copy of CKTI services a single initiation queue. To start or stop a copy of CKTI, you must supply the name of the queue that this CKTI is to serve, or is serving. You cannot start more than one instance of CKTI against the same initiation queue from a single CICS subsystem.

At CICS system initialization or at connect time, you can define a default initiation queue. If you issue a STARTCKTI or a STOPCKTI command without specifying an initiation queue, these commands are automatically interpreted as referring to the default initiation queue.

Notes:

1. If you are using version 4.1 of CICS, any transaction entries processed by CKTI, for example EXEC CICS START, are locked by the CKTI task until it terminates. Any attempt to CEDA INSTALL such entries after altering them will fail: CEDA rejects the install request because the transaction entry is being used by another task.

   In this situation, you must stop the CKTI task using the CICS adapter, and restart it after the CEDA install.
2. This restriction also applies to intersystem connection (ISC) and multi-region operation (MRO) links. For example, if CKTI has started a remote transaction, a connection cannot be reinstalled until CKTI has been stopped.

Multi-tasking

The CICS adapter optimizes the performance of a CICS to MQSeries connection by exploiting multi-processors and by removing work from the main CICS task control block (TCB), allowing multiple MQI calls to be handled concurrently.

The adapter enables some MQI calls to be executed under subtasks, rather than under the main CICS TCB that runs the application code. All the CICS adapter administration code, including connection and disconnection from MQSeries, runs under the main CICS TCB.

The adapter tries to attach up to eight OS/390 subtasks (TCBs) to be used by this CICS system. *You cannot modify this number*. Each subtask makes a connect call to MQSeries. Each CICS system connected takes up nine of the connections specified on the CTHREAD system parameter. This means that you must increase the value specified for CTHREAD by nine for each CICS system connected. MQI calls can flow over those connections. When the main connection is terminated, the subtasks are disconnected and terminated automatically.

The API-crossing exit

MQSeries provides an API-crossing exit for use with the CICS adapter; it runs in the CICS address space. You can use this exit to intercept MQI calls as they are being run, for monitoring, testing, maintenance, or security purposes.

Using the API-crossing exit degrades MQSeries performance. You should plan your use of it carefully.

CICS adapter conventions

There are a number of conventions that must be observed in applications using the adapter.

Temporary storage queue names

The CICS adapter display function uses two temporary storage queues (MAIN) per invoking task to store the output data for browsing. The names of the queues are *tttt*CKRT and *tttt*CKDP, where *tttt* is the terminal identifier of the terminal from which the display function is requested.

Do not try to access these queues.

MQGET

When the CICS adapter puts a task on a CICS wait because the WAIT option was used with the **MQGET** call and there was no message available, the RESOURCE NAME used is GETWAIT and the RESOURCE_TYPE is MQSeries.

When the CICS adapter puts a task on a CICS wait because of a need to perform task switching the RESOURCE NAME used is TASKSWCH and the RESOURCE_TYPE is MQSeries.

ENQUEUE names

The CICS adapter uses the name:

```
CSQ.genericapplid(8).QMGR
```

to issue CICS ENQ and CICS DEQ calls during processing, for example, starting and stopping the connection.

You should not use similar names for CICS ENQ or DEQ purposes.

The CICS bridge

The MQSeries-CICS bridge is the component of MQSeries for OS/390 that allows direct access from MQSeries applications to applications on your CICS system. In bridge applications there are no MQSeries calls within the CICS application (the bridge enables *implicit MQI support*). This means that you can re-engineer legacy applications that were controlled by 3270-connected terminals to be controlled by MQSeries messages, without having to rewrite, recompile, or re-link them.

MQSeries applications use the CICS header (the MQCIH structure) in the message data to ensure that the applications can execute as they did when driven by nonprogrammable terminals.

The bridge enables an application that is not running in a CICS environment to run a *program* or *transaction* on CICS and get a response back. This non-CICS application can be run from any environment that has access to an MQSeries network that encompasses MQSeries for OS/390.

A *program* is a CICS program that can be invoked using the EXEC CICS LINK command. It must conform to the DPL subset of the CICS API, that is, it must not use CICS terminal or syncpoint facilities.

A *transaction* is a CICS transaction designed to run on a 3270 terminal. This transaction can use BMS or TC commands. It can be conversational or part of a pseudoconversation. It is permitted to issue syncpoints. For information about the transactions that can be run, see the *CICS Internet and External Interfaces Guide*.

When to use the CICS bridge

The CICS bridge allows an application to run a single CICS program or a 'set' of CICS programs (often referred to as a unit of work). It caters for the application that waits for a response to come back before it runs the next CICS program (synchronous processing) and for the application that requests one or more CICS programs to run, but doesn't wait for a response (asynchronous processing).

The CICS bridge also allows an application to run a 3270-based CICS transaction, without knowledge of the 3270 data stream.

The CICS bridge uses standard CICS and MQSeries security features and can be configured to authenticate, trust, or ignore the requestor's user ID.

Given this flexibility, there any many instances where the CICS bridge can be used. For example, when you want:

- To write a new MQSeries application that needs access to logic or data (or both) that reside on your CICS server.
- To be able to run CICS programs from a Lotus Notes application.
- To be able to access your CICS applications from
  - Your MQSeries Classes for Java client application
  - A web browser using the MQSeries Internet gateway

System configuration for the CICS bridge

When you are setting your system up, you should ensure that:

- Both MQSeries and CICS are running in the same OS/390 image.
- The MQSeries request queue is local to the CICS bridge, however the response queue can be local or remote.

- The CICS bridge tasks run in the same CICS as the bridge monitor. The user programs can be in the same or a different CICS system.
- The MQSeries-CICS adapter is enabled.

Running CICS DPL programs

Data necessary to run the program is provided in the MQSeries message. The bridge builds a COMMAREA from this data, and runs the program using EXEC CICS LINK. Figure 16 shows the step sequence taken to process a single message to run a CICS DPL program:

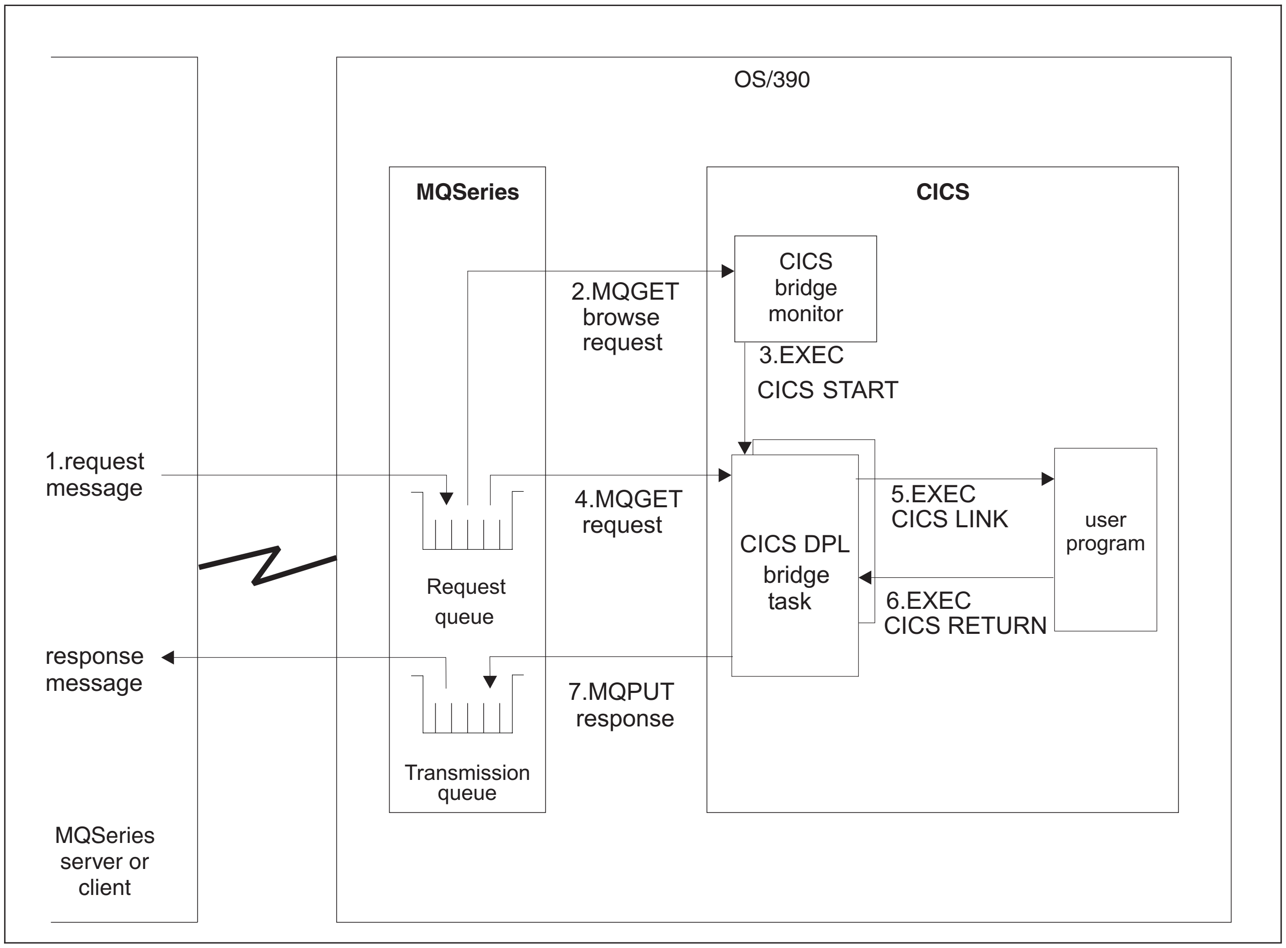


Figure 16. Components and data flow to run a CICS DPL program

The following takes each step in turn, and explains what takes place:

1. A message, with a request to run a CICS program, is put on the request queue.
2. The CICS bridge monitor task, which is constantly browsing the queue, recognizes that a 'start unit of work' message is waiting (`CorrelId`=MQCI_NEW_SESSION).
3. Relevant authentication checks are made, and a CICS DPL bridge task is started with the appropriate authority.
4. The CICS DPL bridge task removes the message from the request queue.
5. The CICS DPL bridge task builds a COMMAREA from the data in the message and issues an EXEC CICS LINK for the program requested in the message.
6. The program returns the response in the COMMAREA used by the request.

7. The CICS DPL bridge task reads the COMMAREA, creates a message, and puts it on the reply-to queue specified in the request message. All response messages (normal and error, requests and replies) are put to the reply-to queue with default context.
8. The CICS DPL bridge task ends.

A unit of work can be just a single user program, or it can be multiple user programs. There is no limit to the number of messages you can send to make up a unit of work.

In this scenario, a unit of work made up of many messages works in the same way, with the exception that the CICS bridge task waits for the next request message in the final step unless it is the last message in the unit of work.

Running CICS 3270 transactions

Data necessary to run the transaction is provided in the MQSeries message. The CICS transaction runs as if it has a real 3270 terminal, but instead uses one or more MQ messages to communicate between the CICS transaction and the MQSeries application

Unlike traditional 3270 emulators, the bridge does not work by replacing the VTAM flows with MQSeries messages. Instead, the message consists of a number of parts called vectors, each of which corresponds to an EXEC CICS request. Therefore the application is talking directly to the CICS transaction, rather than via an emulator, using the actual data used by the transaction (known as application data structures or ADSs).

Figure 17 on page 96 shows the step sequence taken to process a single message to run a CICS 3270 transaction.

The following takes each step in turn, and explains what takes place:

1. A message, with a request to run a CICS transaction, is put on the request queue.
2. The CICS bridge monitor task, which is constantly browsing the queue, recognizes that a ‘start unit of work’ message is waiting (`CorrelId`=MQCI_NEW_SESSION).
3. Relevant authentication checks are made, and a CICS 3270 bridge task is started with the appropriate authority.
4. The MQ-CICS bridge exit removes the message from the queue and changes task to run a user transaction
5. Vectors in the message provide data to answer all terminal related input EXEC CICS requests in the transaction.
6. Terminal related output EXEC CICS requests result in output vectors being built.
7. The MQ-CICS bridge exit builds all the output vectors into a single message and puts this on the reply-to queue.
8. The CICS 3270 bridge task ends.

Note: The CICS bridge exit is an MQSeries supplied CICS exit associated with the bridge transaction.

MQSeries and CICS

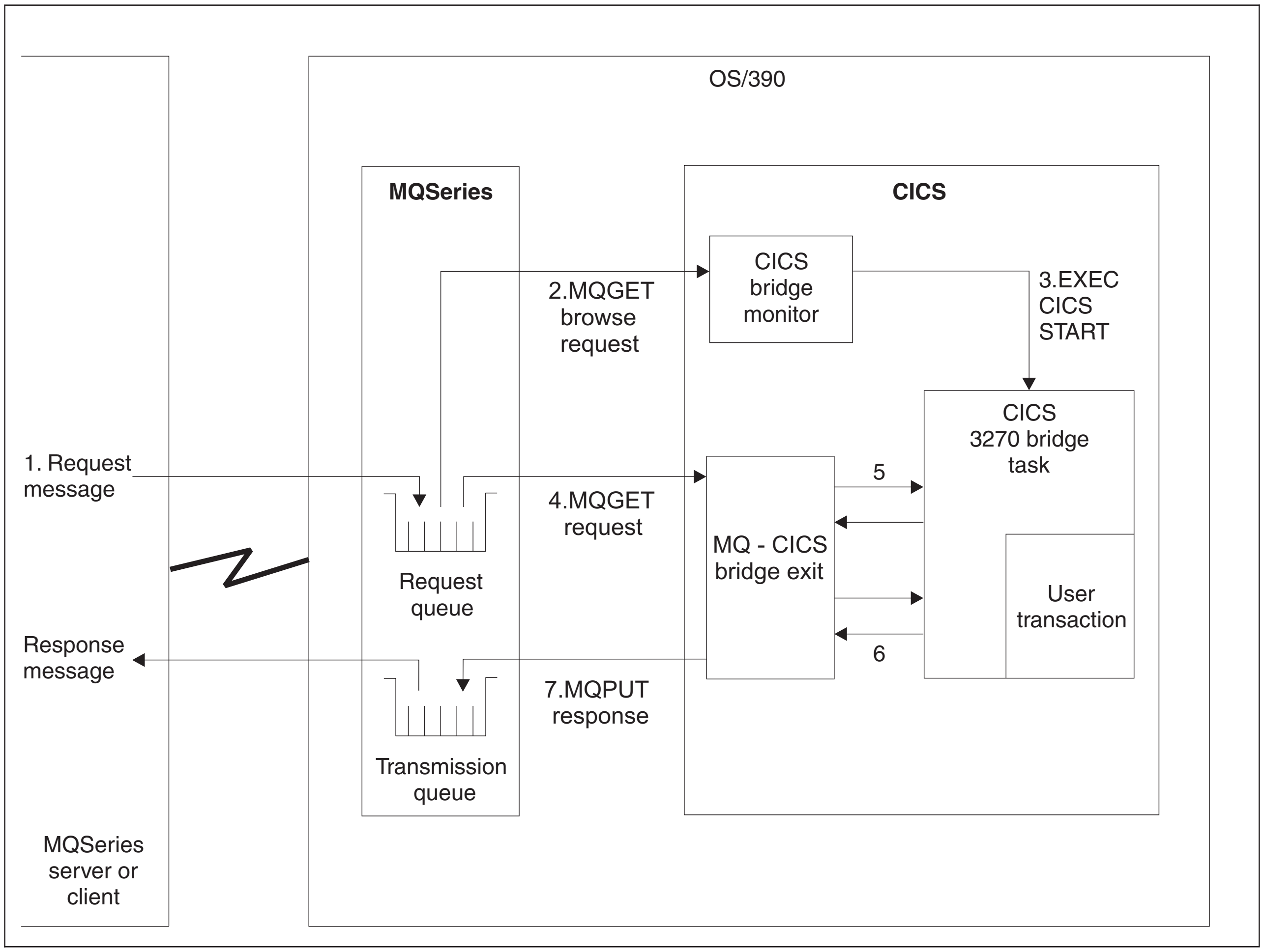


Figure 17. Components and data flow to run a CICS 3270 transaction

A traditional CICS application usually consists of one or more transactions linked together as a pseudoconversation. In general, each transaction is started by the 3270 terminal user entering data onto the screen and pressing an AID key. This model of application can be emulated by an MQSeries application. A message is built for the first transaction, containing information about the transaction, and input vectors. This is put on the queue. The reply message will consist of the output vectors, the name of the next transaction to be run, and a token that is used to represent the pseudoconversation. The MQSeries application builds a new input message, with the transaction name set to the next transaction and the facility token set to the value returned on the previous message. Vectors for this second transaction are added to the message, and the message put on the queue. This process is continued until the application ends.

An alternative approach to writing CICS applications is the conversational model. In this model, the original message might not contain all the data to run the transaction. If the transaction issues a request that cannot be answered by any of the vectors in the message, a message is put onto the reply-to queue requesting more data. The MQSeries application gets this message and puts a new message back to the queue with a vector to satisfy the request.

For more information about this, see the *CICS Internet and External Interfaces Guide*.

Where to find more information

You can find more information about the topics discussed in this chapter from the following sources:

Table 15. Where to find more information about using CICS with MQSeries

Topic	Where to look
System parameters Setting up the CICS adapter Setting up the CICS bridge	*MQSeries for OS/390 System Setup Guide*
Operating the CICS adapter Operating the CICS bridge	*MQSeries for OS/390 System Administration Guide*
Console messages	*MQSeries for OS/390 Messages and Codes*
Writing CICS applications API-crossing exit	*MQSeries Application Programming Guide*
Writing CICS bridge applications	*CICS Internet and External Interfaces Guide*

Chapter 12. MQSeries and IMS

This chapter discusses how MQSeries works with IMS. The IMS adapter and the IMS bridge allow you to connect your MQSeries subsystem to IMS.

- The IMS adapter enables IMS applications to use the MQI.
- The IMS bridge enables applications to run an IMS application that does not use the MQI. This means that you can use your legacy applications with MQSeries, without the need to rewrite them.

These topics are described in the following sections:

The IMS adapter

The MQSeries adapters enable different application environments to send and receive messages through a message queuing network. The IMS adapter is the interface between IMS application programs and an MQSeries subsystem. It makes it possible for IMS application programs to use the MQI.

The IMS adapter receives and interprets requests for access to MQSeries using the External Subsystem Attach Facility (ESAF) provided by IMS. This facility is described in the *IMS Customization Guide*. Usually, IMS connects to MQSeries automatically without operator intervention.

The IMS adapter provides access to MQSeries resources for programs running in:

- Task (TCB) mode
- Problem state
- Non-cross-memory mode
- Non-access register mode

The adapter provides a connection thread from an application task control block (TCB) to MQSeries.

The adapter supports a two-phase commit protocol for changes made to resources owned by MQSeries with IMS acting as the syncpoint coordinator.

The adapter also provides a trigger monitor transaction (CSQQTRMN). This is described in "The IMS trigger monitor" on page 100.

You can use MQSeries with the IMS Extended Recovery Facility (XRF) to aid recovery from a IMS error. For more information about XRF, see the *IMS Administration Guide: System* manual.

Using the adapter

The application programs and the IMS adapter run in the same address space. MQSeries for OS/390 is a separate OS/390 subsystem, in its own address space.

Each program that issues one or more MQI calls must be link-edited to a suitable IMS language interface module, and, unless it uses dynamic MQI calls, the MQSeries-supplied API stub program, CSQQSTUB. When the application issues an MQI call, the stub transfers control to the adapter through the IMS external subsystem interface, which manages the processing of the request by the message queue manager.

System administration and operation with IMS

An authorized IMS terminal operator can issue IMS commands to control and monitor the connection to MQSeries. However, the IMS terminal operator has no control over the MQSeries address space. For example, the operator cannot shut down MQSeries from an IMS address space.

The IMS trigger monitor

The IMS trigger monitor (CSQQTRMN) is an MQSeries-supplied IMS application that starts an IMS transaction when an MQSeries event occurs, for example, when a message is put onto a specific queue.

How it works

When a message is put onto an application message queue, a trigger is generated if the trigger conditions are met. The queue manager then writes a message (containing some user-defined data), known as a *trigger message*, to the initiation queue that has been specified for that message queue. In an IMS environment, an instance of CSQQTRMN can be started to monitor an initiation queue and to retrieve the trigger messages from it as they arrive. Typically, CSQQTRMN schedules another IMS transaction by an INSERT (ISRT) to the IMS message queue. The started IMS application reads the message from the application message queue and then processes it. CSQQTRMN must run as a non-message BMP.

Each copy of CSQQTRMN services a single initiation queue. Once started, the trigger monitor runs until MQSeries or IMS ends.

The APPLCTN macro for CSQQTRMN must specify SCHDTYP=PARALLEL.

Because the trigger monitor is a batch-oriented BMP, IMS transactions started by the trigger monitor will contain:
- Blanks in the LTERM field of the IOPCB
- The PSB name of the trigger monitor BMP in the Userid field of the IOPCB

If the target IMS transaction is RACF protected, you might need to define CSQQTRMN as a user ID to RACF.

The IMS bridge

The MQSeries-IMS bridge is the component of MQSeries for OS/390 that allows direct access from MQSeries applications to applications on your IMS system (the bridge enables *implicit MQI support*). This means that you can re-engineer legacy applications that were controlled by 3270-connected terminals to be controlled by MQSeries messages, without having to rewrite, recompile, or re-link them. The bridge is an IMS *Open Transaction Manager Access* (OTMA) client.

In bridge applications there are no MQSeries calls within the IMS application. The application gets its input using a GET UNIQUE (GU) to the IOPCB and sends its output using an ISRT to the IOPCB. MQSeries applications use the IMS header (the MQIIH structure) in the message data to ensure that the applications can execute as they did when driven by nonprogrammable terminals. If you are using an IMS application that processes multi-segment messages, note that all segments should be contained within one MQSeries message.

The IMS bridge is illustrated in Figure 18.

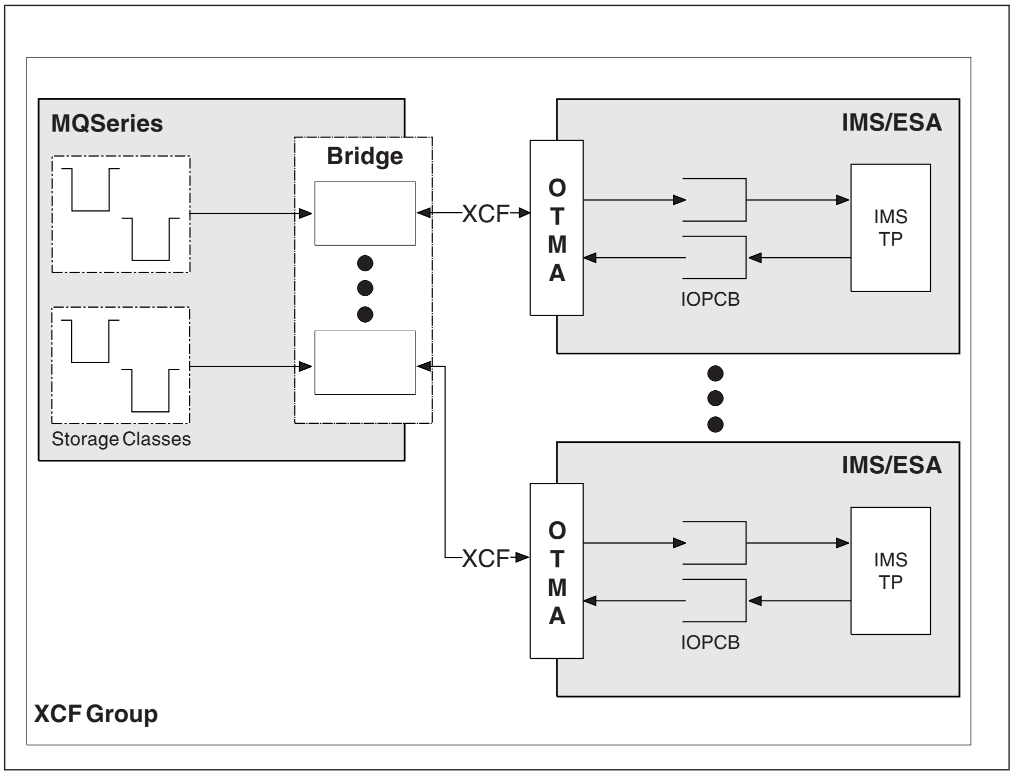


Figure 18. The MQSeries-IMS bridge

A queue manager can connect to one or more IMS systems, and more than one queue manager can connect to one IMS system. The only restriction is that they must all belong to the same XCF group and must all be in the same sysplex.

What is OTMA?

The IMS OTMA facility is a transaction-based connectionless client/server protocol that runs on IMS Version 5.1 or later. It functions as an interface for host-based communications servers accessing IMS TM applications through the OS/390 *Cross Systems Coupling Facility* (XCF).

OTMA enables clients to connect to IMS in a high performance manner enabling the client to support interactions with IMS for a large network or large number of sessions. OTMA is implemented in an OS/390 sysplex environment. Therefore, the domain of OTMA is restricted to the domain of XCF.

Submitting IMS transactions from MQSeries

To submit an IMS transaction that uses the bridge, applications put messages on an MQSeries queue as usual. The messages contain IMS transaction data; they can have an IMS header (the MQIIH structure) or allow the MQSeries-IMS bridge to make assumptions about the data in the message.

MQSeries then puts the message to an IMS queue (it is queued in MQSeries first to enable the use of syncpoints to assure data integrity). The storage class of the MQSeries queue determines whether the queue is an *OTMA queue* (that is, a queue used to transmit messages to the MQSeries-IMS bridge) and the particular IMS partner to which the message data is sent.

Remote queue managers can also start IMS transactions by writing to these OTMA queues on MQSeries for OS/390.

Data returned from the IMS system is written directly to the MQSeries reply-to queue specified in the message descriptor structure (MQMD). (This might be a transmission queue to the queue manager specified in the *ReplyToQMgr* field of the MQMD.)

Where to find more information

You can find more information about the topics discussed in this chapter from the following sources:

Table 16. Where to find more information about using IMS with MQSeries

Topic	Where to look
Setting up the IMS adapter Setting up the IMS bridge	*MQSeries for OS/390 System Setup Guide*
Operating the IMS adapter Operating the IMS bridge	*MQSeries for OS/390 System Administration Guide*
Console messages	*MQSeries for OS/390 Messages and Codes*
Writing IMS applications	*MQSeries Application Programming Guide*
IMS Open Transaction Manager Access (OTMA)	*IMS/ESA Open Transaction Manager Access Guide*
MQIIH structure	*MQSeries Application Programming Reference*

Chapter 13. MQSeries and OS/390 Batch and TSO

This chapter discusses how MQSeries works with OS/390 Batch and TSO. It contains the following sections:

Introduction to the Batch adapters

The Batch/TSO adapters are the interface between OS/390 application programs running under JES, TSO, or UNIX System Services and an MQSeries subsystem. They enable OS/390 application programs to use the MQI.

The adapters provide access to MQSeries resources for programs running in:

- Task (TCB) mode
- Problem or supervisor state
- Non-cross-memory mode
- Non-access register mode

There is one MQSeries address space and there are *allied address spaces* for each environment in which the applications run. Each TSO user and each batch program has its own allied address space.

Connections between application programs and MQSeries are at the task level. The adapters provide a connection thread from an application task control block (TCB) to MQSeries.

The Batch/TSO adapter supports a single-phase commit protocol for changes made to resources owned by MQSeries. It does not support multi-phase commit protocols. The RRS adapter enables MQSeries applications to participate in two-phase commit protocols with other RRS-enabled products, coordinated by OS/390 Resource Recovery Services (RRS).

The adapters use the OS/390 STIMERM service to schedule an asynchronous event every second. This event runs an interrupt request block (IRB) that does not involve any waiting by the batch application's task. This IRB checks to see if the MQSeries termination ECB has been posted. If the termination ECB has been posted, the IRB posts any application ECBs that are waiting on an event in the MQSeries subsystem (for example, a signal or a wait).

The Batch/TSO adapter

The MQSeries Batch/TSO adapter provides MQSeries support for OS/390 Batch and TSO applications. All application programs that run under OS/390 Batch or TSO must have the API stub program CSQBSTUB link-edited with them. The stub provides the application with access to all MQI calls. You use single-phase commit and backout for applications by issuing the MQI calls **MQCMIT** and **MQBACK**.

The RRS adapter

Resource Recovery Services (RRS) is a subcomponent of OS/390 that provides a system-wide service for coordinating two-phase commit across OS/390 products. The MQSeries Batch/TSO RRS adapter (the RRS adapter) provides MQSeries support for OS/390 Batch and TSO applications that want to use these services. The RRS adapter enables MQSeries to become a full participant in RRS coordination. Applications can participate in two-phase commit processing with other products that support RRS (for example, DB2).

The RRS adapter provides two stubs; application programs that want to use RRS must be link-edited with one of these stubs.

CSQBRSTB
This stub allows you to use two-phase commit and backout for applications by using the RRS callable resource recovery services instead of the MQI calls **MQCMIT** and **MQBACK**.

You must also link-edit module ATRSCSS from library SYS1.CSSLIB with your application. If you use the MQI calls **MQCMIT** and **MQBACK**, you will receive return code `MQRC_ENVIRONMENT_ERROR`.

CSQBRSSI
This stub allows you to use MQI calls **MQCMIT** and **MQBACK**; MQSeries actually implements these calls as the **SRRCMIT** and **SRRBACK** RRS calls.

For information about building application programs that use the RRS adapter, see the *MQSeries Application Programming Guide*.

Where to find more information

You can find more information about the topics discussed in this chapter from the following sources:

Table 17. Where to find more information about using OS/390 Batch with MQSeries

Topic	Where to look
Setting up the Batch adapters	*MQSeries for OS/390 System Setup Guide*
Writing applications to use the Batch adapter	*MQSeries Application Programming Guide*
RRS callable resource recovery services	*MVS Programming: Callable Services for High Level Languages*

Part 4. Planning your MQSeries environment

Chapter 14. Planning your storage requirements

This chapter discusses the storage requirements for MQSeries for OS/390. It contains the following sections:

Address space storage

Each MQSeries for OS/390 subsystem has the following approximate storage requirements:

CSA	4 KB
ECSA	800 KB, plus the size of the trace table specified in the TRACTBL parameter of the CSQ6SYSP system parameter macro. This macro is described in the *MQSeries for OS/390 System Setup Guide*.

In addition, each concurrent MQSeries task requires about 5 KB of ECSA. When a task ends, this storage can be reused by other MQSeries tasks. MQSeries does not release the storage until the queue manager is shut down, so the maximum amount of ECSA required can be calculated by multiplying the maximum number of concurrent tasks by 5 KB.

Concurrent tasks consist of the following:

- The number of Batch, TSO or IMS regions that have connected to MQSeries, but not disconnected
- The number of CICS transactions that have issued an MQSeries request, but have not terminated

The channel initiator typically requires ECSA usage of up to 160 KB, plus about 2.4 KB per channel. The channel initiator does not use any CSA.

Private region storage usage

If you are using OS/390 Version 2.6, every channel uses the following private region virtual storage below the 16 MB line in the channel initiator address space:

- 1200 bytes if LE/370 APAR PQ03507 has been applied
- None if LE/370 APAR PQ06157 has also been applied
- 8 KB otherwise

These APARs have been incorporated into later versions of OS/390 so MQSeries channels do not require any private region storage if you are using OS/390 Version 2.7 or later.

Planning your storage requirements

Every channel uses approximately 170 KB of extended private region in the channel initiator address space. Storage is increased by message size if messages larger than 32 KB are transmitted.

This increased storage is freed when:

- A sending or client channel requires less than half the current buffer size for 10 consecutive sends
- A heartbeat is sent or received

The storage is freed for re-use within the LE environment, but is not seen as free by the OS/390 virtual storage manager.

This means that the upper limit for the number of channels is dependent on message size and arrival patterns as well as individual user system limitations on extended private region size. The upper limit on the number of channels is likely to be approximately 9000 on many systems as the extended region size is unlikely to exceed 1.6 GB. The use of message sizes larger than 32 KB reduces the maximum number of channels in the system. For example, if messages that are 100 MB long are transmitted, and an extended region size of 1.6 GB is assumed, the maximum number of channels is 15.

Region sizes

The following table shows suggested values for region sizes. Two sets of values are given; one set is suitable for a test system, the other for a production system or a system that will become a production system eventually.

Table 18. Suggested definitions for JCL region sizes

Definition setting	Test system	Production system
Queue manager	REGION=7168K	REGION=0M
Channel initiator	REGION=7168K	REGION=0M

The region sizes suggested for the test system (REGION=7168K for both the queue manager region and channel initiator region) allow the queue manager and channel initiator to allocate 7 MB of private virtual storage below the 16 MB line (PVT) and up to 32 MB of extended private virtual storage above the 16 MB line (extended PVT).

These values are insufficient for a production system with large buffer pools and many active channels. The production region sizes chosen (REGION=0M) allow all available private storage above and below the 16 MB line to be used, although MQSeries uses very little storage below the 16 MB line.

Note: You can use the MVS exit IEALIMIT to override the region limits below the 16 MB line and IEFUSI to override the region limits above and below the 16 MB line.

Logs and archive storage

Active log data sets record significant events and data changes. They are periodically off-loaded to the archive log. Consequently, the space requirements for your active log data sets depend on the volume of messages that your MQSeries for OS/390 handles and how often the active logs are off-loaded to your archive data sets. MQSeries for OS/390 provides optional support for dual logging; if you use this your log storage requirement will be doubled.

If you decide to place the archive data sets on direct access storage devices (DASD), you need to reserve enough space on the devices. Space should also be reserved for the bootstrap data sets (BSDS). A typical size for each BSDS might be 500 KB. These are all separate data sets and should be allocated space on different volumes and strings if possible to minimize DASD contention and problems caused by any defects on the physical devices.

Because each change to the system is logged, the size of storage required can be estimated from the size and expected throughput of persistent messages (nonpersistent messages are not logged). You must add to this a small overhead for the header information in the data sets.

To arrive at the size of the log extents, an algorithm can be developed which depends on various factors including the message rate and size of persistent messages and how frequently you want to switch the log.

Figure 19 shows an approximate calculation for the number of records to specify in the cluster for the log data set.

```
Number of records = (a * log switch interval) / 4096

where a = (Number of puts/sec*(average persistent message size+500))
         + (Number of gets/sec*110)
         + (Number of units of recovery started*120)
         + (Number of syncpoints per second*240)
 and
       log switch interval = time period between successive log
                             switches required in seconds
```

Figure 19. Calculating the number of records to specify in the cluster for the log data set

Each log data set should have the same number of records specified and should not have secondary extents. Other than for a very small number of records, AMS rounds up the number of records so that a whole number of cylinders is allocated. The number or records actually allocated is:

```
   c = (INT (number of log records / b) + 1) * b

   Where b is the number of 4096-byte blocks per cylinder (180 for a
   3390 device) and INT means round down to an integer
```

DB2 storage

If you are using queue-sharing groups, you need to set up a set of DB2 tables that are used to hold MQSeries data. These are automatically defined for you by MQSeries when you set up your DB2 environment (as described in the *MQSeries for OS/390 System Setup Guide*).

For most installations, the amount of DB2 storage required is about 20 or 30 cylinders on a 3390 device. However, if you want to calculate your storage requirement, the following table gives some information to help you determine how much storage DB2 will require for the MQSeries data. The table describes the length of each DB2 row, and when each row is added to or deleted from the relevant DB2 table. Use this information together with the information about calculating the space requirements for the DB2 tables and their indexes in the *DB2 for OS/390 Installation Guide*.

Table 19. Planning your DB2 storage requirements

DB2 table name	Length of row	A row is added when:	A row is deleted when:
CSQ.ADMIN_B_QSG	223 bytes	A queue-sharing group is added to the table with the ADD QSG function of the CSQ5PQSG utility.	A queue-sharing group is removed from the table with the REMOVE QSG function of the CSQ5PQSG utility. (All rows relating to this queue-sharing group are deleted automatically from all the other DB2 tables when the queue-sharing group record is deleted.)
CSQ.ADMIN_B_QMGR	Up to 3281 bytes	A queue manager is added to the table with the ADD QMGR function of the CSQ5PQSG utility.	A queue manager is removed from the table with the REMOVE QMGR function of the CSQ5PQSG utility.
CSQ.ADMIN_B_STRUCTURE	242 bytes	The first local queue definition, specifying the QSGDISP(SHARED) attribute, that names a previously unknown structure within the queue-sharing group is defined.	The last local queue definition, specifying the QSGDISP(SHARED) attribute, that names a structure within the queue-sharing group is deleted.
CSQ.ADMIN_B_SCST	291 bytes	A shared channel is started.	A shared channel becomes inactive.
CSQ.ADMIN_B_SSKT	240 bytes	A shared channel that has the NPMSPEED(NORMAL) attribute is started.	A shared channel that has the NPMSPEED(NORMAL) attribute becomes inactive.
CSQ.OBJ_B_QUEUE	Up to 3744 bytes	• A queue with the QSGDISP(GROUP) attribute is defined. • A queue with the QSGDISP(SHARED) attribute is defined. • A model queue with the DEFTYPE(SHAREDYN) attribute is opened.	• A queue with the QSGDISP(GROUP) attribute is deleted. • A queue with the QSGDISP(SHARED) attribute is deleted. • A dynamic queue with the DEFTYPE(SHAREDYN) attribute is closed with the DELETE option.
CSQ.OBJ_B_NAMELIST	Up to 15096 bytes	A namelist with the QSGDISP(GROUP) attribute is defined.	A namelist with the QSGDISP(GROUP) attribute is deleted.
CSQ.OBJ_B_CHANNEL	Up to 3646 bytes	A channel with the QSGDISP(GROUP) attribute is defined.	A channel with the QSGDISP(GROUP) attribute is deleted.

Table 19. Planning your DB2 storage requirements (continued)

DB2 table name	Length of row	A row is added when:	A row is deleted when:
CSQ.OBJ_B_STGCLASS	Up to 2836 bytes	A storage class with the QSGDISP(GROUP) attribute is defined.	A storage class with the QSGDISP(GROUP) attribute class is deleted.
CSQ.OBJ_B_PROCESS	Up to 3326 bytes	A process with the QSGDISP(GROUP) attribute is defined.	A process with the QSGDISP(GROUP) attribute is deleted.

Coupling Facility storage

"Planning the size of your structures" on page 124 describes how to determine how large to make your Coupling Facility structures.

Page set and message storage

The amount of storage needed for page data sets depends on the sizes of the messages that your applications will exchange, on the numbers of these messages, and on the rate at which they are created or exchanged. You can calculate the amount of storage needed for page data sets using the algorithm given in "Chapter 15. Planning your page sets and buffer pools" on page 113.

Library storage

You need to allocate storage for the product libraries. The exact figures depend on your configuration, but an estimate of the space required by the distribution libraries is 80 MB. The target libraries require about 72 MB. Additionally, you require space for the SMP/E libraries.

You should refer to the program directory supplied with MQSeries for OS/390 for information about the required libraries and their sizes.

Where to find more information

You can find more information about the topics discussed in this chapter from the following sources:

Table 20. Where to find more information about storage requirements

Topic	Were to look
System parameters	*MQSeries for OS/390 System Setup Guide*
Storage required to install MQSeries	*MQSeries for OS/390 Program Directory*
IEALIMIT and IEFUSI exits	*MVS Installation Exits*
Latest information	MQSeries SupportPac™ Web site http://www.ibm.com/software/mqseries/

Chapter 15. Planning your page sets and buffer pools

This chapter contains the following sections:

Planning your page sets

When deciding on the most appropriate settings for page set definitions, there are a number of factors that should be considered.

Page set usage

In the case of short-lived messages, few pages are normally used on the page set and there is little or no I/O to the data sets except at startup, during a checkpoint, or at shutdown.

In the case of long-lived messages, those pages containing messages are normally written out to disk. This is performed by the queue manager in order to reduce restart time.

You should separate short-lived messages from long-lived messages by placing them on different page sets and in different buffer pools.

Number of page sets

Using several large page sets can make the role of the MQSeries administrator easier because it means that you need fewer page sets, making the mapping of queues to page sets simpler.

Using multiple, smaller page sets has a number of advantages. For example, they take less time to back up, and I/O can be carried out in parallel during backup and restart. However, consider that this adds a significant overhead to the role of the MQSeries administrator, who will be required to map each queue to one of a much greater number of page sets.

You are recommended to define at least five page sets, as follows:

- A page set reserved for object definitions (page set zero)
- A page set for system related messages
- A page set for performance-critical long-lived messages
- A page set for performance-critical short-lived messages
- A page set for all other messages

"Defining your buffer pools" on page 120 explains the performance advantages of distributing your messages on page sets in this way.

Size of page sets

You should allow enough space in your page sets for the expected peak message capacity. You should also specify a secondary extent to allow for any unexpected peak capacity, such as when a build up of messages develops because a queue server program is not running.

The size of the page set also determines the time taken to recover a page set when restoring from a backup, because a large page set takes longer to restore.

Note: Recovery of a page set also depends on the time the queue manager takes to process the log records written since the backup was taken; this is determined by the backup frequency. This is discussed in "Recovery procedures" on page 131.

Calculating the size of your page sets

For queue manager object definitions (for example, queues and processes), it is simple to calculate the storage requirement because these objects are of fixed size and are permanent. For messages however, the calculation is more complex for the following reasons:

- Messages vary in size.
- Messages are transitory.
- Space occupied by messages that have been retrieved is reclaimed periodically by an asynchronous process.

Page set zero

For page set zero, the storage required is:

```
   (maximum number of local queue definitions x 1010)
      (excluding shared queues)
+  (maximum number of model queue definitions x 746)
+  (maximum number of alias queue definitions x 338)
+  (maximum number of remote queue definitions x 434)
+  (maximum number of permanent dynamic queue definitions x 1010)
+  (maximum number of process definitions x 674)
+  (maximum number of namelist definitions x 12320)
+  (maximum number of message channel definitions x 1010)
+  (maximum number of client-connection channel definitions x 1714)
+  (maximum number of server-connection channel definitions x 1010)
+  (maximum number of cluster-receiver channel definitions x 1010)
+  (maximum number of cluster-sender channel definitions x 1010)
+  (maximum number of storage class definitions x 266)
```

Divide this value by 4096 to determine the number of records to specify in the cluster for the page set data set.

You do not need to allow for objects that are stored in the shared repository, but you do have to allow for objects that are stored or copied to page set zero (objects with a disposition of GROUP or QMGR).

The total number of objects that can be created is limited by the capacity of page set zero. There is an implementation limit on the number of local queues that can be defined, which is 524 287.

Page sets 01 to 99

For page sets 01 to 99, the storage required for each page set is determined by the number and size of the messages stored on that page set. (Messages on shared queues are not stored on page sets.)

Divide this value by 4096 to determine the number of records to specify in the cluster for the page set data set.

Calculating the storage requirement for messages

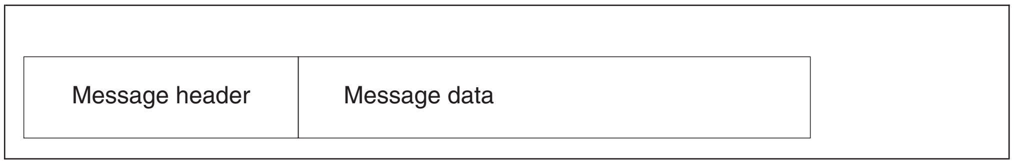
This section describes how messages are stored on pages. Understanding this will help you calculate how much page set storage you need to define for your messages. To calculate the approximate space required for all messages on a page set you must consider maximum queue depth of all the queues that map to the page set and the average size of messages on those queues.

You must allow for the possibility that message “gets” might be delayed for reasons outside the control of MQSeries (for example, because of a problem with your communications protocol). In this case, the “put” rate of messages might far exceed the “get” rate. This could lead to a large increase in the number of messages stored in the page sets and a consequent increase in the storage size demanded.

Each page in the page set is 4096 bytes long. Allowing for fixed header information, each page has 4057 bytes of space available for storing messages.

When calculating the space required for each message, the first thing you need to consider is whether the message will fit on one page (a *short message*) or whether it needs to be split over two or more pages (a *long message*). When messages are split in this way, you need to allow for additional control information in your space calculations.

For the purposes of space calculation, a message can be represented like this:



The message header section contains the message descriptor (352 bytes) and other control information, the size of which varies depending on the size of the message. The message data section contains all the actual message data, and any other headers (for example, the transmission header or the IMS bridge header).

A minimum of two pages are required for page set control information. This is typically less than 1% of the total space required for messages.

Short messages: A short message is defined as a message that will fit on one page.

For a short message the control information is 20 bytes long. When this is added to the length of the message header, the usable space remaining on the page is 3685 bytes. If the size of the message data is 3685 bytes or less, MQSeries stores the messages in the next available space on the page, or if there is not enough space available, on the next page, as shown in Figure 20:

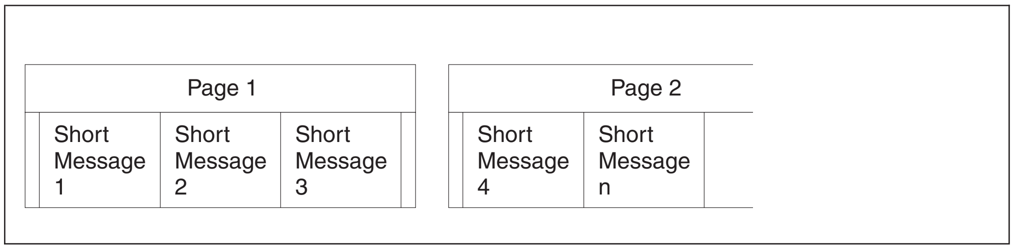


Figure 20. How MQSeries stores short messages on page sets

If there is sufficient space remaining on the page, the next message is also stored on this page, if not, the remaining space on the page is left unused.

Long messages: If the size of the message data is greater than 3685 bytes, but not greater than 4 MB, the message is classed as a long message. When presented with a long message, MQSeries stores the message on a series of pages, and stores control information that points to these pages in the same way that it would store a short message. This is shown in Figure 21:

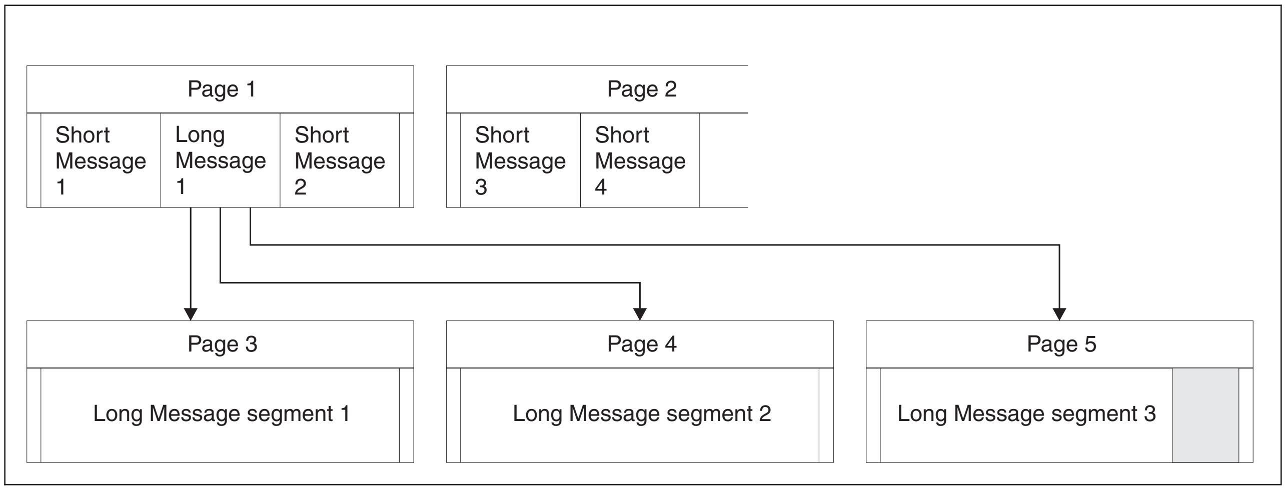


Figure 21. How MQSeries stores long messages on page sets

Each segment of the long message is preceded by 8 bytes of control information, and the first segment also includes the message header portion of 352 bytes. This means that the first page contains 3697 bytes of the message data. The remaining message data is placed on subsequent pages, in 4049-byte segments. If this does not fill an exact number of pages, the remaining space in the last page is left unused.

The number of pages (n) used for a long message is calculated as follows:

```
     message data length + 352
n = -------------------------
               4049

rounded up to the nearest page
```

In addition to this, you need to allow space for the control information that points to the pages. The length of this (c) depends on the length of the message, and is calculated as follows:

```
c = 20 + (3n) bytes

(where n is the number of pages calculated above)
```

This means that the total page set space required for a long message is:

```
(n * 4096) + c bytes
```

Very long messages: Very long messages are messages with a size greater than 4 MB. These are stored so that each 4 MB uses 1037 pages. Any remainder is stored in the same was as a long message, as described above.

Enabling dynamic page set expansion

Page sets can be extended dynamically while MQSeries is running. A page set can have up to 123 extents, which can exist on multiple disk volumes.

Note: The maximum number of extents for a page set cataloged in an ICF catalog is between 119 and 123, depending upon the number of extents (1-5) allocated by direct access storage data management (DADSM) per allocate/extend request.

In order to use this facility, your page sets must be allocated with secondary extent values defined. If you have existing page set definitions, they cannot be altered to add secondary extent definitions. You will have to re-allocate each of your page sets with secondary extents, and then use the COPYPAGE function of CSQUTIL to copy the old versions of the page sets to the new ones.

Sample thlqual.SCSQPROC(CSQ4PAGE) shows how to define the secondary extents. CSQUTIL is described in the *MQSeries for OS/390 System Administration Guide*.

How to determine an appropriate secondary extent value

You might decide the secondary extent value by considering how many times the page set should exceed its original value. For instance, if your page set primary allocation is 1000 units (records/pages, tracks, cylinders, KB, MB), and you want it to grow to be at most four times that size, then determine the secondary extent size from:

```
              ( maximum size   -  original size )
              -----------------------------------
                              119

 In this case, 4000 - 1000  = 3000  = 25 or 26
               -----------    ----
                   119         119
```

You might not be able to use this much disk space, as described in “Number of extents available”.

Note: If you define the size of your extent in records, IDCAMS rounds this up to map onto a physical boundary. The queue manager uses all this space for the secondary extent.

Number of extents available

The Data Facility Product (DFP) uses up to five non-contiguous areas of disk to satisfy the total space requirements of a primary or secondary extent. This means, in the worst case of badly fragmented disk space, that you might only get around 22 times the secondary space allocated before you reach the maximum extent limit.

Multivolume data sets

If the definition of a page set allows it to utilize multiple volumes, the primary space is wholly contained on the first volume, and secondary extents are first allocated on the same volume, while space is available, and thereafter the next volume is used, while space is available, and so on. The process stops when you have used all the secondary extents, or no more disk space is available.

This behavior differs if the page set is a data set managed by Storage Management Subsystem (SMS), and you use a storage class that uses the GUARANTEED SPACE attribute. In this case, a primary extent is allocated on each volume when the page set is defined. Thereafter, secondary extents are allocated, as before, except that when the services of a new volume are required, the pre-allocated secondary extent is used.

Defining your buffer pools

The following table shows suggested values for buffer pool definitions that affect the performance of queue manager operation, recovery, and restart. Two sets of values are given; one set is suitable for a test system, the other for a production system or a system that will become a production system eventually.

Table 21. Suggested definitions for buffer pool settings

Definition setting	Test system	Production system
BUFFPOOL 0	1 050 buffers	50 000 buffers
BUFFPOOL 1	1 050 buffers	20 000 buffers
BUFFPOOL 2	1 050 buffers	50 000 buffers
BUFFPOOL 3	1 050 buffers	20 000 buffers

You are recommended to reserve buffer pool 0 for object definitions (in page set 0) and performance critical, system related message queues, such as the SYSTEM.CHANNEL.SYNCQ queue and the SYSTEM.CLUSTER.* queues. The remaining three buffer pools can be used for user messages, for example:

- Buffer pool 1 might be used for important long-lived messages.

  Long-lived messages are those that remain in the system for longer than two checkpoints, at which time they are written out to the page set. If you have a large number of long-lived messages, this buffer pool should be relatively small, so that page set I/O is evenly distributed (older messages are written out to DASD each time the buffer pool becomes 85% full).

  If the buffer pool is too large, page set I/O is delayed until checkpoint processing. This might affect response times throughout the system.

  If you expect a small number of long-lived messages only, this buffer pool should be defined so that it is sufficiently large to hold all of these messages.
- Buffer pool 2 might be used for performance-critical, short-lived messages.

  There will normally be a high degree of buffer reuse, using a small number of buffers; however, you are recommended to make this buffer pool large in order to allow for unexpected message accumulation, for example, when a server application fails.
- Buffer pool 3 might be used for all other (usually performance non-critical) messages. Queues such as the dead-letter queue, SYSTEM.COMMAND.* queues and SYSTEM.ADMIN.* queues can also be mapped to buffer pool 3.

  Where virtual storage constraints exist and buffer pools need to be smaller, buffer pool 3 is the first candidate for size reduction.

Initially, you are recommended to define all buffer pools as shown in the table. Usage of buffer pools can be monitored by analysis of buffer pool performance statistics. In particular, you should ensure that the buffer pools are large enough so that the values of QPSTSOS, QPSTSTLA and QPSTNBUF remain at zero. (These performance statistics are described in the *MQSeries for OS/390 System Setup Guide*.)

Buffer pool 0 and the buffer pool for short-lived messages (buffer pool 2) should be tuned so that the 15% free threshold is never exceeded (that is, QPSTCBSL divided by QPSTNBUF is always greater than 15%). If more than 15% of buffers remain free, I/O to the page sets using these buffer pools can be largely avoided during normal operation, although messages older than two checkpoints will be written to page sets.

Note: The optimum value for these parameters is dependent on the characteristics of the individual system. The values given are only intended as a guideline and might not be appropriate for your system.

MQSeries SupportPac *Capacity planning and tuning for MQSeries for OS/390* (MP16) gives more information about tuning buffer pools.

Chapter 16. Planning your Coupling Facility and DB2 environment

This chapter discusses the following topics:

- "Defining Coupling Facility resources"
- "Planning your DB2 environment" on page 126

Defining Coupling Facility resources

If you intend to use shared queues, you must define the Coupling Facility structures that MQSeries will use in your CFRM policy. To do this you must first update your CFRM policy with information about the structures, and then activate the policy.

Your installation probably has an existing CFRM policy that describes the Coupling Facilities available. The IXCMIAPU OS/390 utility is used to modify the contents of the policy based on textual statements you provide. The utility is described in the *MVS Setting up a Sysplex* manual. You must add statements to the policy that define the names of the new structures, the Coupling Facilities that they are to be defined in, and what size the structures will be.

Planning your structures

A queue-sharing group requires a minimum of two structures to be defined. The first structure, known as the administrative structure, is used to coordinate MQSeries internal activity across the queue-sharing group. No user data is held in this structure. It has a fixed name of *qsg-name*CSQ_ADMIN (where *qsg-name* is the name of your queue-sharing group). Subsequent structures are used to hold the messages on MQSeries shared queues. Each structure is able to hold up to 512 shared queues.

Using multiple structures

A queue-sharing group can connect to up to 64 Coupling Facility structures. One of these structures must be the administration structure, but you can use up to 63 structures for MQSeries data. You might choose to use multiple structure because:

- You have some queues that are likely to hold a very large number of messages and so will require all the resources of an entire Coupling Facility.
- You have a requirement for a very large number of shared queues, so they must be split across multiple structures because each structure can only contain 512 queues.
- RMF™ reports on the usage characteristic of a structure suggest that you should distribute the queues it contains across a number of Coupling Facilities.
- You want some queue data to held in a physically different Coupling Facility from other queue data for data isolation reasons.

When choosing which Coupling Facilities the structures should be allocated in, you should consider the following points:

- Your data isolation requirements.
- The volatility of the Coupling Facility (that is, its ability to preserve data through a power outage).

- Failure independence between the accessing systems and the Coupling Facility, or between Coupling Facilities.
- The level of Coupling Facility Control Code (CFCC) installed on the Coupling Facility (MQSeries requires Level 9 or higher).

Planning the size of your structures

The size of the administrative structure (*qsg-name*CSQ_ADMIN) must be at least 10 MB.

The size of the structures required to hold MQSeries messages depends on the likely number and size of the messages to be held on a structure concurrently, together with an estimate of the likely number of concurrent units of work.

The graph in Figure 22 shows how large you should make your CF structures to hold the messages on your shared queues. To calculate the allocation size you need to know

- The average size of messages on your queues
- The total number of messages likely to be stored in the structure

Find the value of log10 of the number of messages, (for example, 200 000 messages gives a value of 5.3) along the horizontal axis. Select the curve that corresponds to your message size and determine the required value from the vertical axis. For example, for 200 000 messages of length 1 KB gives a value of about 29. This means you need to a Coupling Facility structure with a size of 2^{28} = 256 MB.

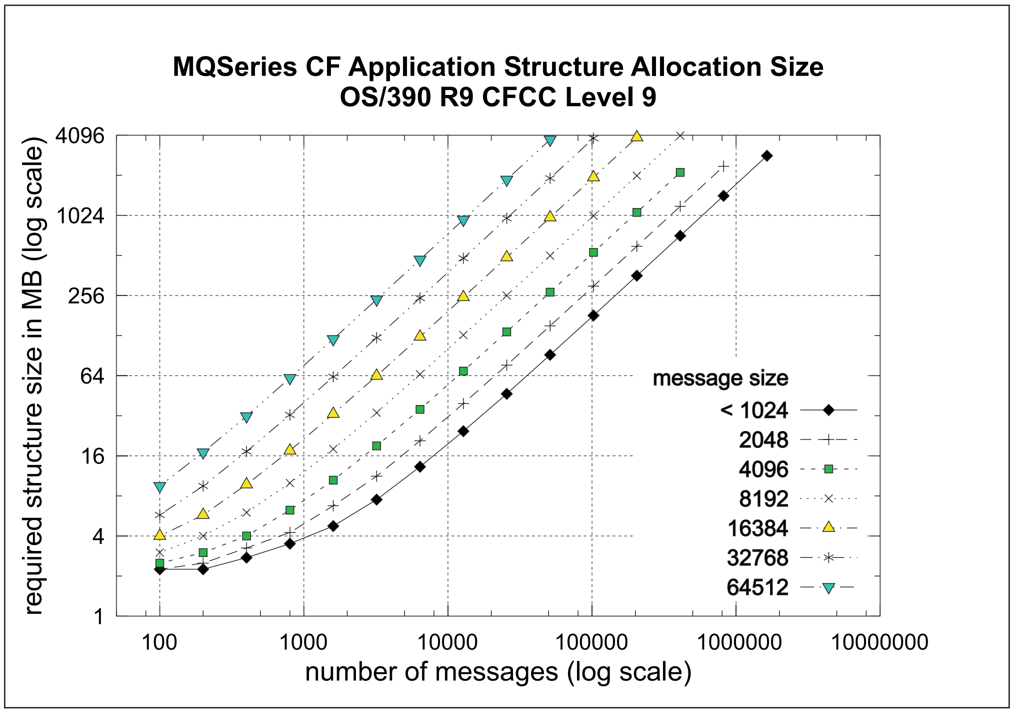


Figure 22. Calculating the size of a Coupling Facility structure

Your CFRM policy should include the following statements:

```
CFNAME(structure-name)
INITSIZE(value from graph, in KB)
MAXSIZE(something larger)
FULLTHRESHOLD(85)
```

`INITSIZE` is the size in KB that XES will allocate to the structure when the first connector connects to it. `MAXSIZE` is the maximum size that the structure can attain. `FULLTHRESHOLD` sets the percentage value of the threshold at which XES issues message IXC585E to indicate that the structure is getting full.

For example, with the figures determined above, you might include the following statements:

```
CFNAME(QSG1APPLICATION1)
INITSIZE(272144) /* 256 MB */
MAXSIZE(524288)  /* 512 MB */
FULLTHRESHOLD(85)
```

If the structure utilization reaches the threshold where warning messages are issued, intervention is required. You might use MQSeries to inhibit **MQPUT** operations to some of the queues in the structure to prevent applications from writing more messages, start more applications to get messages from the queues, or quiesce some of the applications that are putting messages to the queue.

Alternatively XES facilities can be used to alter the structure size in place. The following OS/390 command:

```
SETXCF START,ALTER,CFNAME=structure-name,SIZE=newsize
```

alters the size of the structure to *newsize*, where *newsize* is a value that is less than the value of MAXSIZE specified on the CFRM policy for the structure, but greater than the current Coupling Facility size.

You can monitor the utilization of a Coupling Facility structure with the MQSeries DISPLAY GROUP command.

If no action is taken and a queue structure fills up, an MQRC_STORAGE_MEDIUM_FULL return code is returned to the application. If the administration structure becomes full, the exact symptoms depend on which processes experience the error, but they might include the following problems:

- No responses to commands.
- Queue manager failure as a result of problems during commit processing.

Mapping shared queues to structures

The CFSTRUCT attribute of the queue definition is used to map the queue to a structure.

MQSeries adds the name of the queue-sharing group to the beginning of the CFSTRUCT attribute. For a structure defined in the CFRM policy with name *qsg-name*SHAREDQ01, the definition of a queue that uses this structure will be:

```
DEFINE QLOCAL(myqueue) QSGDISP(SHARED) CFSTRUCT(SHAREDQ01)
```

Planning your DB2 environment

If you are using queue-sharing groups, MQSeries needs to attach to a DB2 subsystem that is a member of a data-sharing group. MQSeries needs to know the name of the data-sharing group that it is to connect to, and the name of a DB2 subsystem (or DB2 group) to connect to in order to reach this data-sharing group. These names are specified in the QSGDATA parameter of the CSQ6SYSP system parameter macro (described in the *MQSeries for OS/390 System Setup Guide*).

MQSeries uses the RRS Attach facility of DB2. This means that you can specify the name of a DB2 group that you want to connect to. The advantage of connecting to a DB2 group attach name (rather than a specific DB2 subsystem), is that MQSeries can connect (or reconnect) to any available DB2 subsystem on the OS/390 image that is a member of that group. There must be a DB2 subsystem that is a member of the data-sharing group active on each OS/390 image where you are going to run a queue-sharing MQSeries subsystem, and RRS must be active.

Chapter 17. Planning your logging environment

The MQSeries logging environment is established using the system parameter macros to specify options, such as whether to have single or dual active logs, what media to use for the archive log volumes, and how many log buffers to have. These macros are described in the *MQSeries for OS/390 System Setup Guide*.

This chapter discusses the following topics:
- "Planning your logs"
- "Planning your archive storage" on page 129

Planning your logs

You are recommended to have at least three log data sets, and to use dual logging and log archiving. The following table shows suggested values for log and bootstrap data set definitions that affect the performance of queue manager operation, recovery, and restart. Two sets of values are given; one set is suitable for a test system, the other for a production system or a system that will become a production system.

Table 22. Suggested definitions for log and bootstrap data sets

Definition setting	Test system	Production system
Log data set size[1]	10 000 records (40 MB approx.) Access Method Services rounds to nearest cylinder and allocates 10 080 records.	180 000 records (700 MB or 1 000 cylinders of 3390)
Number of active logs[1]	3	6
BSDS size	60 records (240 KB approx.)	120 records (480 KB approx.) A primary extent of this size should be sufficient for the maximum number of archive logs that can be recorded in the BSDS (1 000).

Note:
1. The optimum value for this definition is dependent on the characteristics of the individual system. The values given are intended as a guideline only and might not be appropriate for your system.

Log data set definitions

Before setting up the log data sets, review the following section in order to decide on the most appropriate configuration for your system.

Should your installation use single or dual logging?

If your DASD type is 3390 or similar, you are recommended to use dual logging in order to ensure that you have an alternative backup source in the event of losing a data set. You should also use dual BSDSs and dual archiving to ensure adequate provision for data recovery.

If you use devices with in-built data redundancy (for example, Redundant Array of Independent Disks (RAID) devices) you might consider using single active logging. If you use persistent messages, single logging can increase maximum capacity by 10-30% and can also improve response times.

Dual active logging adds a small performance overhead. You might want to specify it on a test system used for benchmarking, in addition to a production system.

How many active log data sets do you need?

You should have sufficient active logs to ensure that your system is not impacted in the event of an archive being delayed.

In practice, you should have at least three active log data sets but it is preferable to define more. For example, if the time taken to fill a log is likely to approach the time taken to archive a log during peak load, you should define more logs. You are also recommended to define more logs to offset possible delays in log archiving. If you use archive logs on tape, allow for the time required to mount the tape.

How large should the active logs be?

Your logs should be large enough so that it takes at least 30 minutes to fill a single log during the expected peak persistent message load. If you are archiving to tape, you are advised to make the logs large enough to fill one tape cartridge, or a number of tape cartridges. (For example, a log size of 1000 cylinders on 3390 DASD will fit onto a 3490E non-compacted tape with space to spare.)

Note: When archiving to tape, a copy of the BSDS is also written to the tape. When archiving to DASD, a separate data set is created for the BSDS copy.

If the logs are small (for example, 10 cylinders) it is likely that they will fill up frequently, which could result in performance degradation. In addition, you might find that the large number of archive logs required is difficult to manage.

If the logs are very large, and you are archiving to DASD, you will need a corresponding amount of spare space reserved on DASD for SMS retrieval of migrated archive logs, which might cause space management problems. In addition, the time taken to restart might increase because one or more of the logs has to be read sequentially at startup.

Active log placement

Ideally, each of the active logs should be allocated on separate, low-usage DASD volumes. As a minimum, no two adjacent logs should be on the same volume.

When an active log fills, the next log in the ring is used and the previous log data set is copied to the archive data set. If these two active data sets are on the same volume, contention will result, because one data set is read while the other is written to. For example, if you have three active logs and use dual logging, you will need six DASD volumes because each log is adjacent to both of the two other logs. Alternatively, if you have four active logs and you want to conserve DASD space, by allocating logs 1 and 3 on one volume and logs 2 and 4 on another, you will require four DASD volumes only.

In addition, you should ensure that primary and secondary logs are on separate physical units. If you use 3390 DASD, be aware that each head disk assembly contains two logical volumes. The physical layout of other DASD subsystems such as RAMAC® arrays should also be taken into account.

Planning your archive storage

This section describes the different ways of maintaining your archive log data sets.

Archive log data sets can be placed on standard-label tapes, or DASD, and can be managed by data facility hierarchical storage manager (DFHSM). Each OS/390 logical record in an archive log data set is a VSAM control interval from the active log data set. The block size is a multiple of 4 KB.

Archive log data sets are dynamically allocated, with names chosen by MQSeries. The data set name prefix, block size, unit name, and DASD sizes needed for such allocations are specified in the system parameter module. You can also choose, at installation time, to have MQSeries add a date and time to the archive log data set name.

It is not possible to choose specific volumes for new archive logs. If allocation errors occur, off-loading is postponed until the next time off-loading is triggered.

If you specify dual archive logs at installation time, each log control interval retrieved from the active log is written to two archive log data sets. The log records that are contained in the pair of archive log data sets are identical, but the end-of-volume points are not synchronized for multi-volume data sets.

Should your archive logs reside on tape or DASD?

When deciding whether to use tape or DASD for your archive logs, there are a number of factors that you should consider:

1. Review your operating procedures before making decisions about tape or disk. For example, if you choose to archive to tape, operators must be available to mount the appropriate tapes when they are required.
2. During recovery, archive logs on tape are available as soon as the tape is mounted. If DASD archives have been used, and the data sets migrated to tape using hierarchical storage manager (HSM), there will be a delay while HSM recalls each data set to disk. You can recall the data sets before the archive log is used. However, it is not always possible to predict the correct order in which they will be required.
3. When using archive logs on DASD, if many logs are required (which might be the case when recovering a page set after restoring from a backup) you might require a significant quantity of DASD in order to hold all the archive logs.
4. In a low usage system or test system, it might be more convenient to have archive logs on DASD in order to eliminate the need for tape mounts.

Archiving to DASD offers faster recoverability but is more expensive than archiving to tape. If you use dual logging, you can specify that the primary copy of the archive log go to DASD and the secondary copy go to tape. This increases recovery speed without using as much DASD, and the tape can be used as a backup.

Archiving to tape

If you choose to archive to a tape device, MQSeries can extend to a maximum of twenty volumes.

If you choose to off-load to tape, you should consider adjusting the size of your active log data sets so that each nearly fills a tape volume. This minimizes tape handling and volume mounts, and maximizes the use of tape resources. However, such an adjustment is not essential.

If you are considering changing the size of the active log data set so that the set fits on one tape volume, you must bear in mind that a copy of the BSDS is placed on the same tape volume as the copy of the active log data set. Adjust the size of the active log data set downward to offset the space required for the BSDS on the tape volume.

If you use dual archive logs on tape, it is typical for one copy to be held locally, and the other copy to be held off-site for use in disaster recovery.

Archiving to DASD volumes

MQSeries requires that all archive log data sets allocated on non-tape devices (DASD) be cataloged. If you choose to archive to DASD, the CATALOG parameter of the CSQ6ARVP macro must be YES. (This macro is described in the *MQSeries for OS/390 System Setup Guide*.) If this parameter is NO, and you decide to place archive log data sets on DASD, you receive message CSQJ072E each time an archive log data set is allocated, although the MQSeries subsystem still catalogs the data set.

If the archive log data set is held on DASD, the archive log data sets cannot extend to another volume.

If you choose to use DASD, make sure that the primary space allocation (both quantity and block size) is large enough to contain either the data coming from the active log data set, or that from the corresponding BSDS, whichever is the larger of the two. This minimizes the possibility of unwanted OS/390 B37 or E37 abends during the off-load process. The primary space allocation is set with the PRIQTY (primary quantity) parameter of the CSQ6ARVP macro.

Using SMS with archive log data sets

If you have MVS/DFP™ storage management subsystem (DFSMS) installed, you can write an Automatic Class Selection (ACS) user-exit filter for your archive log data sets, which helps you convert them for the SMS environment. Such a filter, for example, can route your output to a DASD data set, which in turn can be managed by DFSMS™. You must exercise caution if you use an ACS filter in this manner. Because SMS requires DASD data sets to be cataloged, you must make sure the CATALOG DATA field of the CSQ6ARVP macro contains YES. If it does not, message CSQJ072E is returned; however, the data set is still cataloged by MQSeries.

For more information about ACS filters, see the *DFP Storage Administration Reference* manual, and the *SMS Migration Planning Guide*.

Chapter 18. Planning for backup and recovery

Developing backup and recovery procedures at your site is vital to avoid costly and time-consuming losses of data. MQSeries provides means for recovering both queues and messages to their current state after a system failure.

This chapter contains the following sections:

Recovery procedures

You should develop the following procedures for MQSeries:
- Creating a point of recovery
- Backing up page sets
- Recovering page sets
- Recovering from out-of-space conditions (MQSeries logs and page sets)

See the *MQSeries for OS/390 System Administration Guide* for information about these.

You should also be familiar with the procedures used at your site for the following:
- Recovering from a hardware or power failure
- Recovering from an OS/390 component failure
- Recovering from a site interruption, using off-site recovery

General tips for backup and recovery

This section introduces some backup and recovery tasks. The MQSeries restart process recovers your data to a consistent state by applying log information to the page sets. If your page sets are damaged or unavailable, you can resolve the problem using your backup copies of your page sets (provided that all the logs are available). If your log data sets are damaged or unavailable, it might not be possible to recover completely.

Periodically take backup copies

A *point of recovery* is the term used to describe a set of backup copies of MQSeries page sets and the corresponding log data sets required to recover these page sets. These backup copies provide a potential restart point in the event of page set loss (for example, page set I/O error). If MQSeries were to be restarted using these backup copies, the data in MQSeries will be consistent up to the point that these copies were taken. Providing that all logs are available from this point, MQSeries can be recovered to the point of failure.

The more recent your backup copies, the quicker MQSeries can recover the data in the page sets. The recovery of the page sets are dependent on all the necessary log data sets being available.

In planning for recovery, you need to determine how often to take backup copies and how many complete backup cycles to keep. These values tell you how long you must keep your log data sets and backup copies of page sets for MQSeries recovery.

In deciding how often to take backup copies, consider the time needed to recover a page set. It is determined by:
- The amount of log to traverse
- The time it takes an operator to mount and remove archive tape volumes
- The time it takes to read the part of the log needed for recovery
- The time needed to reprocess changed pages
- The storage medium used for the backup copies
- The method used to make and restore backup copies

In general, the more frequently you make backup copies, the less time recovery takes, but the more time is spent making copies.

For each queue manager, you should take backup copies of:
- The archive log data sets
- The BSDS copies created at the time of the archive
- The page sets
- Your object definitions

To reduce the risk of your backup copies being lost of damaged, you should consider:
- Storing the backup copies on different storage volumes to the original copies.
- Storing the backup copies at a different site to the original copies.
- Making at least two copies of each backup of your page sets and, if you are using single logging or a single BSDS, two copies of your archive logs and BSDS. If you are using dual logging or BSDS, a single copy of both archive logs or BSDS will suffice.

Before moving MQSeries to a production environment you should have tested and documented your backup procedures.

Backing up your object definitions

You should also create backup copies of your object definitions. To do this, use the MAKEDEF feature of the COMMAND function of the utility program (described in the *MQSeries for OS/390 System Administration Guide*).

You should do this whenever you take backup copies of your queue manager data sets, and keep the most current version.

Do not discard archive logs you might need

MQSeries might need to use archive logs during restart. You must keep sufficient archive logs so the system can be fully restored. MQSeries might use an archive log to recover a page set from a restored backup copy. If you have discarded that archive log, MQSeries is not able to restore the page set to its current state. When and how you should discard archive logs is described in the *MQSeries for OS/390 System Administration Guide*.

Do not change the DDname to page set association

MQSeries associates page set number 00 with DDname CSQP0000, page set number 01 with DDname CSQP0001, and so on up to CSQP0099. MQSeries writes recovery log records for a page set based on the DDname that the page set is associated with. For this reason, you must not move page sets that have already been associated with a PSID DDname.

Recovering page sets

A key factor in recovery strategy concerns the period of time for which you can tolerate a queue manager outage. The total outage time might include the time taken to recover a page set from a backup, or to restart the queue manager after an abnormal termination. Factors affecting restart time include how frequently you back up your page sets, and how much data is written to the log between checkpoints.

In order to minimize the restart time after an abnormal termination, keep units of work short so that, at most, two active logs are used when the system restarts. For example, if you are designing an MQSeries application, avoid placing an **MQGET** call that has a long wait interval between the first in-syncpoint MQI call and the commit point because this might result in a unit of work that has a long duration. Other common causes of long units of work are batch intervals of more than 5 minutes for the channel initiator, and in-doubt channels in the CICS mover.

You can use the DISPLAY THREAD command to display the RBA of units of work and to help resolve the old ones. For information about the DISPLAY THREAD command, see the *MQSeries MQSC Command Reference* manual.

How often should a page set be backed up?

Frequent page set backup is essential if a reasonably short recovery time is required. This applies even when a page set is very small or there is a small amount of activity on queues in that page set.

If you use persistent messages in a page set, the backup frequency should be in the order of hours rather than days. This is also the case for page set zero.

In order to calculate an approximate backup frequency, start by determining the target total recovery time. This will consist of:

1. The time taken to react to the problem.
2. The time taken to restore the page set backup copy.

   (For example, you can restore approximately 60 cylinders of 3390 data per minute from and to RAMAC Virtual Array 2 Turbo 82 (RVA2-T82) DASD using Access Method Services REPRO.)

   If you use SnapShot backup/restore, the time taken to perform this task is of the order of a few seconds. For information about SnapShot, see the *DFSMSdss Storage Administration Guide*.
3. The time the queue manager requires to restart, including the additional time needed to recover the page set.

   This depends most significantly on the amount of log data that must be read from active and archive logs since that page set was last backed up. All such log data must be read, in addition to that directly associated with the damaged page set.

   Note: When using “fuzzy” backup, it might be necessary to read up to three additional checkpoints, and this might result in the need to read one or more additional logs.

When deciding on how long to allow for the recovery of the page set, the factors that you need to consider are:

- The rate at which data is written to the active logs during normal processing:
  - Approximately 1.3 KB of extra data is required on the log for a persistent message.
  - Approximately 2.5 KB of data is required on the log for each batch of messages sent on a channel.
  - Approximately 1.4 KB of data is required on the log for each batch of messages received on a channel.
  - Nonpersistent messages require no log data. NPMSPEED(FAST) channels require no log data for batches consisting entirely of nonpersistent messages.

  The rate at which data is written to the log depends on how messages arrive in your system, in addition to the message rate. Messages received or sent over a channel result in more data logging than messages generated and retrieved locally.
- The rate at which data can be read from the archive and active logs.

  When reading the logs, the achievable data rate depends on the devices used and the overall load on your particular DASD subsystem. (For example, data rates of approximately 2.7 MB per second have been observed using active and archive logs on RVA2-T82 DASD.)

  With most tape units, it is possible to achieve higher data rates for archived logs with a large block size. However, if an archive log is required for recovery, all the data on the active logs must be read also.

Achieving specific recovery targets

If you have specific recovery targets to achieve, for example, completion of the queue manager recovery and restart processing in addition to the normal startup time within *xx* seconds, you can use the following calculation to estimate your backup frequency (in hours):

```
Formula (A)
                    Required restart time * System recovery log read rate
                          (in secs)                (in MB/sec)
Backup frequency  = -----------------------------------------------------
   (in hours)              Application log write rate (in MB/hour)
```

Note: The examples given below are intended to highlight the need to back up your page sets on a frequent basis. The calculations assume that the majority of log activity is derived from a large number of persistent messages. However, there are situations where the amount of log activity is not easily calculated. For example, in a queue-sharing group environment, a unit of work in which shared queues are updated in addition to other resources might result in UOW records being written to the MQSeries log. For this reason, the ’Application log write rate’ in Formula (A) can only be derived accurately from the observed rate at which the MQSeries logs fill.

For example, consider a system in which MQSeries clients generate an overall load of 100 persistent messages per second. In this case, all messages are generated locally.

If each message is of user length 1 KB, the amount of data logged per hour is of the order:

```
100 * (1 + 1.3) KB * 3600 = approximately 800 MB

where
      100           = the message rate per second
      (1 + 1.3) KB  = the amount of data logged for
                      each 1 KB of persistent messages
```

Consider an overall target recovery time of 75 minutes. If you have allowed 15 minutes to react to the problem and restore the page set backup copy, queue manager recovery and restart must then complete within 60 minutes (3600 seconds) applying formula (A). Assuming that all required log data is on RVA2-T82 DASD, which has a recovery rate of approximately 2.7 MB per second, this necessitates a page set backup frequency of at least every:

```
  3600 seconds * 2.7 MB per second / 800 MB per hour  =  12.15 hours
```

If your MQSeries application day lasts approximately 12 hours, one backup each day is appropriate. However, if the application day lasts 24 hours, two backups each day is more appropriate.

Another example might be a production system in which all the messages are for request-reply applications (that is, a persistent message is received on a receiver channel and a persistent reply message is generated and sent down a sender channel).

In this example, the achieved batch size is one, and so there is one batch for every message. If there are 50 request replies per second, the overall load is 100 persistent messages per second. If each message is 1 KB in length, the amount of data logged per hour is of the order:

```
50((2 * (1+1.3) KB) + 1.4 KB + 2.5 KB) * 3600 = approximately 1500 MB

where:
 50                 = the message pair rate per second
 (2 * (1 + 1.3) KB) = the amount of data logged for each message pair
 1.4 KB             = the overhead for each batch of messages
                      received by each channel
 2.5 KB             = the overhead for each batch of messages sent
                      by each channel
```

In order to achieve the queue manager recovery and restart within 30 minutes (1800 seconds), again assuming that all required log data is on RVA2-T82 DASD, this requires that page set backup is carried out at least every:

```
    1800 seconds * 2.7 MB per second / 1500 MB per hour = 3.24 hours
```

Periodic review of backup frequency

You are recommended to monitor your MQSeries log usage in terms of MB per hour. You should perform this check periodically and amend your page set backup frequency if necessary.

Backup and recovery with DFHSM

The data facility hierarchical storage manager (DFHSM) does automatic space- and data-availability management among storage devices in your system. If you use it, you need to know that it moves data to and from the MQSeries storage automatically.

DFHSM manages your DASD space efficiently by moving data sets that have not been used recently to alternate storage. It also makes your data available for recovery by automatically copying new or changed data sets to tape or DASD backup volumes. It can delete data sets, or move them to another device. Its operations occur daily, at a specified time, and allow for keeping a data set for a predetermined period before deleting or moving it.

All DFHSM operations can also be performed manually. The *Data Facility Hierarchical Storage Manager User's Guide* explains how to use the DFHSM commands. If you use DFHSM with MQSeries, note that DFHSM:
- Uses cataloged data sets
- Operates on page sets and logs
- Supports VSAM data sets

MQSeries recovery and CICS

The recovery of CICS resources is not affected by the presence of MQSeries. CICS recognizes MQSeries as a non-CICS resource (or external resource manager), and includes MQSeries as a participant in any syncpoint coordination requests using the CICS resource manager interface (RMI). For more information about CICS recovery, see the *CICS Recovery and Restart Guide*. For information about the CICS resource manager interface, see the *CICS Customization Guide*.

MQSeries recovery and IMS

IMS recognizes MQSeries as an external subsystem and as a participant in syncpoint coordination. IMS recovery for external subsystem resources is described in the *IMS Customization Guide*.

Preparing for recovery on an alternative site

In the case of a total loss of an MQSeries computing center, you can recover on another MQSeries system at a recovery site. To be able to do this, you must regularly back up the page sets and the logs. As with all data recovery operations, the objectives of disaster recovery are to lose as little data, workload processing (updates), and time as possible.

At the recovery site:

- The recovery MQSeries queue manager **must** have the same name as the lost queue manager.
- The system parameter module used on the recovery queue manager should contain the same parameters as the lost queue manager.

The process for disaster recovery is described in the *MQSeries for OS/390 System Administration Guide*.

Part 5. Planning to install MQSeries

Chapter 19. MQSeries Prerequisites

This chapter discusses the prerequisite products that you need to install before you can install and use MQSeries for OS/390.

Machine requirements

MQSeries runs on any IBM® System/390® processor that is capable of running the required level of OS/390, and which has enough storage to meet the combined requirements of the programming prerequisites, MQSeries, the access methods, and the application programs.

Software requirements

This section lists the software requirements for MQSeries. We recommend that you use one of the packaged offerings of OS/390 from IBM (for example, ServerPac). This includes most of the products that you will need to run MQSeries on OS/390, and the integrated products have been verified by IBM.

The list of elements included in OS/390, and the recommended levels of nonexclusive elements are described in the *OS/390 Planning For Installation* manual.

MQSeries for OS/390 Version 5.2 requires OS/390 Version 2.6 or later, together with the products included in OS/390. These include:
- C/C++
- DFSMS™/dfp
- High Level Assembler
- ICSS
- ISPF
- JES
- Language Environment® (previously known as LE/370)
- Security Server (previously known as RACF®)
- SMP/E
- TCP/IP
- TSO/E
- UNIX Services (previously known as OpenEdition®)
- VTAM®

The following list gives the minimum levels required for the optional products that are not included in OS/390. You might not need all of these products.
- CICS, Version 4.1
- COBOL:
  - IBM SAA AD/Cycle® COBOL/370™
  - VS COBOL 2, Version 4
  - IBM COBOL for OS/390 and VM, Version 2.1
  - IBM COBOL for MVS and VM, Version 2.1
- IMS, Version 5.1
- IBM AD/Cycle PL/I for MVS and VM, Version 1.1

Additional requirements for some features

Some of the features of MQSeries for OS/390 Version 5.2 have additional requirements. These are listed below:

Queue-sharing groups
- Coupling Facility, Level 9
- DB2, Version 5.1
- OS/390, Version 2.9 or later

CICS bridge (3270 transactions)
- CICS Transaction Server, Release 2 or later. Release 2 requires APAR PQ32659, Release 3 requires APAR PQ23961.

Internet Gateway
- Internet Connection Secure Server, Version 2.2
- Java, Version 1.1.1
- Web browser that supports HTML 3.2 or later

MQSeries Workflow
- MQSeries Workflow, Version 3.1

Non-IBM products

If you choose to use SOLVE:TCPaccess from Computer Associates in place of IBM's TCP/IP, the recommended minimum level is Version 4.1.

Clients

For MQSeries for OS/390 to support clients you need to install the client/server support code that is provided by the Client Attachment Feature of MQSeries for OS/390.

Delivery

MQSeries is supplied on either 6250 tape or 3480 cartridge. (It is also available on 4-mm DAT tape for PC/390.) One tape or cartridge contains the product code together with four language features – U.S. English (mixed case), U.S. English (uppercase), Japanese, and Simplified Chinese – and the optional CICS mover and Internet Gateway features. The optional Client Attachment feature is supplied on a separate tape or cartridge.

Chapter 20. Making MQSeries available

This chapter gives an overview of what you need to do to make MQSeries available to application programmers, and their applications. It contains the following sections:
- "Installing MQSeries for OS/390"
- "Customizing MQSeries and its adapters" on page 147
- "Verifying your installation of MQSeries for OS/390" on page 147

If you are migrating from a previous version of MQSeries for OS/390, see "Chapter 21. Migrating from previous versions" on page 149.

Installing MQSeries for OS/390

MQSeries for OS/390 uses the standard OS/390 installation procedure. It is supplied with a Program Directory that contains specific instructions for installing the program on an OS/390 system. You must follow the instructions in the *MQSeries for OS/390 Program Directory.* They include not only details of the installation process, but also information about the prerequisite products and their service or maintenance levels.

SMP/E, used for installation on the OS/390 platform, validates the service levels and prerequisite and corequisite products, and maintains the SMP/E history records to record the installation of MQSeries for OS/390. It loads the MQSeries for OS/390 libraries and checks that the loads have been successful. You then have to customize the product to your own requirements.

Before you install MQSeries for OS/390, you must decide the following:
- Whether you are going to install one of the optional national language features.
- Which communications protocol and distributed queuing facility you are going to use.
- What your naming convention for MQSeries objects will be.
- What command prefix string (CPF) you are going to use for each queue manager.

You also need to plan how much storage you require in your OS/390 system to accommodate MQSeries; "Chapter 14. Planning your storage requirements" on page 107 helps you plan the amount of storage required.

National language support

You can choose one of the following national languages for the MQSeries operator messages and the MQSeries operations and control panels (including the character sets used). Each language is identified by a language letter:

C	Simplified Chinese
E	U.S. English (mixed case)
K	Japanese
U	U.S. English (uppercase)

The samples, MQSeries commands, and utility control statements are available only in mixed case U.S. English.

Communications protocol and distributed queuing

The distributed queuing facility provided with the base product feature of MQSeries uses native OS/390 communications (APPC or TCP/IP). It can either use APPC (LU 6.2), TCP/IP from IBM, or SOLVE:TCPaccess. The distributed queuing facility is also known as the channel initiator and the mover.

Alternatively, you can use CICS ISC for distributed queuing; this facility is also known as the CICS mover. You must install the CICS mover feature in order to use it. (This feature is retained for compatibility with previous releases but there will be no further enhancements to this function. Therefore, you are recommended to use the channel initiator for distributed queuing.)

You can enable both facilities and use them simultaneously on the same MQSeries instance. However, the two types will have no knowledge of each other or each other's channels, and you must ensure that the channel names they use are distinct.

If you want to use clients, the client attachment feature is needed as well (but you can administer clients without it).

Whichever mover you choose, you must perform the following tasks to enable distributed queuing:

- Choose which communications interface to use. This can be:
  - APPC (LU 6.2)
  - IBM TCP/IP OpenEdition sockets
  - SOLVE:TCPaccess (native, IUCV, or OpenEdition sockets)
- Customize the distributed queuing facility and define the MQSeries objects required.
- Define access security.
- Set up your communications. This includes setting up your TCPIP.DATA data set if you are using TCP/IP, LU names and side information if you are using APPC, and CICS definitions if you are using CICS. This is described in the *MQSeries Intercommunication* manual.

Naming conventions

It is advisable to establish a set of naming conventions when planning your MQSeries systems. The names you choose will probably be used on different platforms, so you should follow the convention for MQSeries, not for the particular platform.

MQSeries allows both uppercase and lowercase letters in names, and the names are case sensitive. However, some OS/390 consoles fold names to uppercase so do not use lowercase characters for names unless you are sure that this will not happen.

You can also use numeric characters and the period (.), forward slash (/), underscore (_) and percent (%) characters. The percent sign is a special character to RACF, so you should not use it in names if you are using RACF as your External Security Manager. You should not use leading or trailing underscore characters if you are planning to use the Operations and Control panels.

Rules for naming MQSeries objects are described in the *MQSeries MQSC Command Reference* manual.

Choosing names for queue managers and queue-sharing groups

Each queue manager and queue-sharing group within a network must have a unique name. On OS/390 the names of queue managers and queue-sharing groups can be up to four characters long. Each DB2 system and data-sharing group within the network must also have a unique name.

The queue manager name is the same as the OS/390 subsystem name. You could identify each subsystem as a queue manager by giving it the name **QM***xx* (where *xx* is a distinguishing identifier), or you could choose a naming convention similar to **ADDX**, where **A** signifies the geographic area, **DD** signifies the company division, and **X** is a distinguishing identifier.

You might want to use your naming convention to distinguish between queue managers and queue-sharing groups. For example, you could identify each queue-sharing group by giving it the name **QG***xx* (where *xx* is the distinguishing identifier).

Choosing names for objects

Queues, processes, namelists, and clusters can have names up to 48 characters long. Channels can have names up to 20 characters long and storage classes can have names up to 8 characters long.

If possible, choose meaningful names within any constraints of your local conventions. Any structure or hierarchy within names is ignored by MQSeries, however, hierarchical names can be useful for system management. You can also specify a description of the object when you define it to give more information about its purpose.

Each object must have a unique name within its object type. However, each object type has a separate name space, so you can define objects of different types with the same name. For example, if a queue has an associated process definition, it is a good idea to give the queue and the process the same name. It is also a good idea to give a transmission queue the same name as its destination queue manager.

You could also use the naming convention to identify whether the object definition is private or a global. For example, you could call a namelist **project_group.global** to indicate that the definition is stored on the shared repository.

Application queues: Choosing names that describe the function of each queue helps you to manage these queues more easily. For example, you could call a queue for inquiries about the company payroll **payroll_inquiry**. The reply-to queue for responses to the inquiries could be called **payroll_inquiry_reply**.

You can use a prefix to group related queues. This means that you can specify groups of queues for administration tasks like managing security and using the dead-letter queue handler. For example, all the queues that belong to the payroll application could be prefixed by **payroll_**. You can then define a single security profile to protect all queues with names beginning with this prefix.

You can also use your naming convention to indicate that a queue is a shared queue. For example, if the payroll inquiry queue mentioned above was a shared queue, you could call it **payroll_inquiry.shared**.

Storage classes and Coupling Facility structures: The character set you can use when naming storage classes and Coupling Facility structures is limited to uppercase alphabetic and numeric characters. You should be systematic when choosing names for these objects.

Storage class names can be up to 8 characters long, and must begin with an alphabetic character. You will probably not define many storage classes, so a simple name is sufficient. For example, a storage class for IMS bridge queues could be called **`IMS`**.

Coupling Facility structure names can be up to 16 characters long. The first four characters are the name of the queue-sharing group. The subsequent characters can be alphabetic or numeric. You could use the name to indicate something about the shared queues associated with the Coupling Facility structure (that they all belong to one suite of applications for example). Alternatively, you could use a system similar to those suggested for queue manager names.

Choosing names for channels

To help you manage channels, it is a good idea if the channel name includes the names of the source and target queue managers. For example, a channel transmitting messages from a queue manager called QM27 to a queue manager called QM11 might be called **`QM27/QM11`**.

If your network supports both TCP and SNA, you might also want to include the transport type in the channel name, for example **`QM27/QM11_TCP`**. You could also indicate whether the channel is a shared channel, for example **`QM27/QM11_TCP.shared`**.

Remember that channel names cannot be longer than 20 characters. If you are communicating with a queue manager on a different platform, where the name of the queue manager might contain more than 4 characters, you might not be able to include the whole name in the name of the channel.

Using command prefix strings

Each instance of MQSeries that you install must have its own *command prefix* string (CPF). You use the CPF to identify the OS/390 subsystem that commands are intended for. It also identifies the OS/390 subsystem from which messages sent to the console originate.

You can issue all MQSeries commands from an authorized console by inserting the CPF before the command. If you enter commands via the system command input queue (for example, using CSQUTIL), or use the MQSeries operations and control panels, you do not use the CPF.

To start a subsystem called CSQ1 whose CPF is '`+CSQ1`', issue the command `+CSQ1 START QMGR` from the operator console (the space between the CPF and the command is optional).

The CPF also identifies the subsystem that is returning operator messages. The following example shows `+CSQ1` as the CPF between the message number and the message text.

```
CSQ9022I +CSQ1 CSQNCDSP ' DISPLAY CMDSERV' NORMAL COMPLETION
```

See the *MQSeries for OS/390 System Setup Guide* for information about defining command prefix strings.

Customizing MQSeries and its adapters

MQSeries requires some customization after installation in order to meet the individual and special requirements of your system, and to use your system resources in the most effective way. These are the tasks that you must perform when you customize your system:

1. Identify the OS/390 system parameters
2. Review the number of system LXs
3. Define the MQSeries subsystem to OS/390
4. Update the OS/390 link list and LPA
5. APF authorize the MQSeries load libraries
6. Update the OS/390 program properties table
7. Create procedures for the MQSeries subsystem
8. Create procedures for the channel initiator
9. Set up the DB2 environment
10. Set up the Coupling Facility
11. Implement your ESM security controls
12. Customize the initialization input data sets
13. Create the bootstrap and log data sets
14. Define your page sets
15. Run the queue-sharing group utility
16. Tailor your system parameter module
17. Tailor the channel initiator parameter module
18. Set up Batch, TSO, and RRS adapters
19. Set up the operations and control panels
20. Include the MQSeries dump formatting member
21. Suppress information messages

These tasks are described in detail in the *MQSeries for OS/390 System Setup Guide*.

Verifying your installation of MQSeries for OS/390

After the installation and customization has been completed, you can use the installation verification programs (IVPs) supplied with MQSeries to verify that the installation has been completed successfully. The IVPs supplied are assembler language programs and should be run after MQSeries has been customized to suit your needs. They are described in the *MQSeries for OS/390 System Setup Guide*.

Chapter 21. Migrating from previous versions

This chapter is for people who are planning to migrate from a previous version of MQSeries for OS/390. It contains the following sections:
- "What's new for this release"
- "What to do when you migrate from a previous version" on page 153

What's new for this release

This section describes the new function that has been added for this release of MQSeries for OS/390.

Queue-sharing groups

- You can group your queue managers into a queue-sharing group. These queue managers can access the same set of shared queues. This is described in "Chapter 2. Shared queues and queue-sharing groups" on page 11.
- A new system parameter (QSGDATA) sets the queue-sharing group parameters for your queue manager. This is described in the *MQSeries for OS/390 System Setup Guide*.
- You can now define an object once on one queue manager and then use the object definition on other queue managers in the queue-sharing group. This is described in "Private and global definitions" on page 73.
- You can now send commands to all queue managers in a queue-sharing group by issuing the command on one member of the group. This is described in "Directing commands to different queue managers" on page 75.
- You can now use intra-group queuing to send messages between queue managers in a queue-sharing group, without setting up channels. This is described in "Intra-group queuing" on page 20.

Channel initiator

- You can now use shared channels. This is described in "Distributed queuing and queue-sharing groups" on page 18.
- You can now stop a receiver channel automatically and start a new one in its place when a request to start a duplicate receiver channel is received by using the ADOPTMCA and ADOPTCHK parameters of the CSQ6CHIP channel initiator parameter macro. This is described in the *MQSeries Intercommunication* manual; CSQ6CHIP is described in the *MQSeries for OS/390 System Setup Guide*. (This function was available as a PTF for previous releases.)
- You can now specify a range of port numbers to be used when binding outbound channels by using the OPORTMIN and OPORTMAX parameters of the CSQ6CHIP channel initiator parameter macro. This is described in the *MQSeries for OS/390 System Setup Guide*. (This function was available as a PTF for previous releases.)
- You can now specify a single IP address upon which the TCP/IP listener accepts inbound connection requests. This is described in the *MQSeries MQSC Command Reference*.
- You can now specify that a TCP listener should register with Workload Manager for Dynamic Domain Name Services by using the DNSWLM and DNSGROUP parameters of the CSQ6CHIP channel initiator parameter macro. This is described in the *MQSeries for OS/390 System Setup Guide*.

- You can now specify an LU name that is defined as part of a generic resource for an LU 6.2 listener by using the LUGROUP parameter of the CSQ6CHIP channel initiator parameter macro. This is described in the *MQSeries for OS/390 System Setup Guide*.

Commands

- The following commands have been added:

CLEAR QLOCAL	Clear messages from a local queue.
DISPLAY GROUP	Display information about the queue-sharing group to which the queue manager is connected.
DISPLAY QSTATUS	Display queue status information.
MOVE QLOCAL	Move messages from one queue to another.
RESET QSTATS	Display information about how many messages have been put to and retrieved from a queue.

  These commands are described in the *MQSeries MQSC Command Reference*.
- The QSGDISP attribute has been added to many commands to specify the object disposition (described in "Private and global definitions" on page 73).
- The CHLDISP attribute has been added to the channel commands to specify the channel disposition (described in the *MQSeries MQSC Command Reference*).
- The CMDSCOPE attribute has been added to most commands to specify the command scope (described in "Directing commands to different queue managers" on page 75).
- The responses to many commands have changed.

System parameters

- The following new system parameters have been added (they are all described elsewhere in this chapter):

CSQ6SYSP	QSGDATA, RESAUDIT
CSQ6LOGP	DEALLCT, MAXRTU
CSQ6ARVP	UNIT2

- The default settings for the following system parameters have changed:

CSQ6SYSP	The default for LOGLOAD is now 500 000.
CSQ6LOGP	The default for INBUFF is now 60 KB and the default for OUTBUFF is now 4 000 KB.

  These are described in the *MQSeries for OS/390 System Setup Guide*.
- The MAXALLC parameter of CSQ6LOGP is no longer used.

System object samples

- A new system object sample (CSQ4INSS) that defines the objects required for queue-sharing groups has been added (described in "CSQ4INSS system object sample" on page 45).
- The default buffer pool, storage class and page set definitions have been changed. These are explained in the CSQ4INP1 and CSQ4INYG sample data sets (described in "Sample definitions supplied with MQSeries" on page 45).
- The sample object data set for the basic IVP has been renamed to CSQ4IVPQ, and a new sample object data set called CSQ4IVPG has been added for the queue-sharing group IVP.
- The default region sizes have been increased to 0M for the queue manager and channel initiator address spaces.

Logs

- The default settings for log placement and size have been changed. These changes are described in the CSQ4BSDS sample data set.
- You can now specify that you want both copies of the archive log to be written to different device types by using the UNIT and UNIT2 parameters of the CSQ6ARVP system parameter macro. This is described in the *MQSeries for OS/390 System Setup Guide*.
- You can now restart the log archive process after a failure by using the ARCHIVE LOG CANCEL OFFLOAD command. This command is described in the *MQSeries MQSC Command Reference*.
- You can now optimize archive log reading from tape devices using the MAXRTU and DEALLCT parameters of the CSQ6LOGP system parameter macro. You can display and reset these parameters using the DISPLAY LOG and SET LOG commands. CSQ6LOGP is described in the *MQSeries for OS/390 System Setup Guide*; the commands are described in the *MQSeries MQSC Command Reference*.

Security

- You can now use one set of security profiles to control security for all the queue managers in a queue-sharing group. This is described in "Queue manager or queue-sharing group level checking" on page 62.
- You can now control whether RACF audit records are written for RESLEVEL security checks using the RESAUDIT parameter of the CSQ6SYSP system parameter module. This is described in the *MQSeries for OS/390 System Setup Guide*.

Statistics and accounting

- You can now use the SMF data collection broadcast to gather MQSeries statistics and accounting records. This is described in the *MQSeries for OS/390 System Setup Guide*.
- You can now gather performance statistics for the Coupling Facility manager and DB2 manager. This is described in the *MQSeries for OS/390 System Setup Guide*.
- You can now gather accounting data at queue and thread level. This is described in the *MQSeries for OS/390 System Setup Guide*.

Operations and control panels

- The operations and control panels have been changed extensively to include the new function. For example, you are now asked to enter the disposition of objects that you are working with, and this information is included when you use the panels to display an object. This is described in the *MQSeries for OS/390 System Administration Guide* and in the help supplied with the operations and control panels.
- The DEFINE action has been changed to DEFINE LIKE.
- You can now get and collate information for the whole queue-sharing group.
- You can now display queue status information.

Dead-letter queue

There is a new utility program (CSQUDLQH) which processes messages on the dead-letter queue. This is described in the *MQSeries for OS/390 System Administration Guide*.

Application programming

- MQSeries messages can now be up to 100 MB in length. (This function was available as a PTF for previous releases.)
- The Application Messaging Interface (AMI) provides a simple interface to the Message Queue Interface (MQI). Application programmers can use this interface to write applications without needing to understand all the functions available in the MQI. The functions that are required in a particular installation are defined by a system administrator. This function is described in the *MQSeries Application Messaging Interface* manual.
- The MQRC_STORAGE_MEDIUM_FULL return code has been added. This is described in the *MQSeries Application Programming Reference*.
- New code pages have been added, including one for UNICODE. These are described in the *MQSeries Application Programming Reference*.
- You can now specify options when you connect to a queue manager using the **MQCONNX** call. This is described in the *MQSeries Application Programming Reference*.

What to do when you migrate from a previous version

When you migrate from a previous version of MQSeries for OS/390, you can continue to use your existing subsystems with the new version, including their page sets, log data sets, object definitions, and initialization input data sets. You can continue to use your existing queues, including system queues such as the SYSTEM.CHANNEL.SYNCQ. **You should not cold start your queue managers when migrating from a previous version because you do not need to. If you do, you will lose all your messages and other information such as channel state.**

However, there are some tasks that you need to perform when migrating from a previous version. Whether you need to perform each task depends on which of the new features you want to use, and which level of MQSeries you are migrating from. Generally, you need to perform more tasks if you are migrating from a version of MQSeries that was not the previous version (that is, Version 1.2 or earlier); however, you do not need to install the intervening versions.

The following list introduces the tasks that you might have to perform when migrating from Version 2.1 to Version 5.2. These tasks are described in the *MQSeries for OS/390 System Setup Guide*.

- Check levels of prerequisite and corequisite software.
- Check and update your system parameter macros if you want to use some of the new functions.
- Check and update your channel initiator parameter macro if you want to use some of the new functions.
- Customize and include new sample initialization data set CSQ4INSS if you want to use queue-sharing groups.
- Review sample initialization data set CSQ4INYG to see if you want to use the new default buffer pool, storage class, and page set definitions.
- Review sample data set CSQ4BSDS to see if you want to use the new default settings for log placement and size.
- Customize and include new sample data set CSQ4IVPG if you want to run the IVP for queue-sharing groups. (Note that the sample data set for the basic IVP has been renamed to CSQ4IVPQ.)
- Check and change names for renamed MQSeries libraries in STEPLIBs.
- Migrate queues and queue definitions to shared queues, if required.
- Check and change RBA values if you are using the NEWLOG function of the change log utility.
- Use MQRC_STORAGE_MEDIUM_FULL in place of MQRC_PAGESET_FULL.
- If you are using data conversion exits written for MQSeries for OS/390 Version 2.1, they will continue to function correctly with Version 5.2. However, they are unable to convert messages containing text using the Unicode UCS-2 coded character sets (1200, 13488, 17584) and need to be updated to do so. This is described in the *MQSeries for OS/390 System Setup Guide*.

See the *MQSeries for OS/390 System Setup Guide* for information about the additional tasks you might need to perform if you are migrating from Version 1.2 or Version 1.1.4.

Reverting to a previous version

After you have migrated to Version 5.2 of MQSeries for OS/390, it is possible to revert to using Version 2.1 or Version 1.2 if exceptional circumstances so dictate. However, to do this you need to apply a PTF to the previous version. This is described in the *MQSeries for OS/390 System Setup Guide*.

Part 6. Appendixes

Appendix A. Macros intended for customer use

The macros identified in this appendix are provided as programming interfaces for customers by MQSeries.

Note: Do not use as programming interfaces any MQSeries macros other than those identified in this appendix.

General-use programming interface macros

The following macros are provided to enable you to write programs that use the services of MQSeries. The macros are supplied in library thlqual.SCSQMACS.

CMQA	CMQCXPA	CMQRMHA	CMQXCALA
CMQCDA	CMQDLHA	CMQTMA	CMQXCFBA
CMQCFA	CMQDXPA	CMQTMC2A	CMQXCFCA
CMQCFHA	CMQGMOA	CMQWCRA	CMQXCDFA
CMQCFILA	CMQIIHA	CMQWDRA	CMQXCINA
CMQCFINA	CMQMDA	CMQWIHA	CMQXCVCA
CMQCFSLA	CMQMDEA	CMQWPRA	CMQXPA
CMQCFSTA	CMQODA	CMQWXPA	CMQXQHA
CMQCIHA	CMQPMOA	CMQXA	CMQXWDA

Product-sensitive programming interface macros

The following macros are provided to enable you to write programs that use the services of MQSeries. The macros are supplied in library thlqual.SCSQMACS.

CSQBDEF	CSQDQEST	CSQDQIST	CSQDQJST
CSQDQLST	CSQDQMAC	CSQDQMST	CSQDQPST
CSQDQSST	CSQDQWHC	CSQDQWHS	CSQDQ5ST
CSQDWQ	CSQDWTAS	CSQQDEFX	CSQQLITX

General-use programming interface copy files

The following COBOL copy files are provided to enable you to write programs that use the services of MQSeries. The copy files are supplied in library thlqual.SCSQCOBC.

CMQCDL	CMQCFSTV	CMQIIHV	CMQTML
CMQCDV	CMQCFV	CMQMDEL	CMQTMV
CMQCFHL	CMQCIHL	CMQMDEV	CMQTMC2L
CMQCFHV	CMQCIHV	CMQMDL	CMQTMC2V
CMQCFILL	CMQCXPL	CMQMDV	CMQWIHL
CMQCFILV	CMQCXPV	CMQODL	CMQWIHV
CMQCFINL	CMQDLHL	CMQODV	CMQV
CMQCFINV	CMQDLHV	CMQPMOL	CMQXV
CMQCFSLL	CMQGMOL	CMQPMOV	CMQXQHL
CMQCFSLV	CMQGMOV	CMQRMHL	CMQXQHV
CMQCFSTL	CMQIIHL	CMQRMHV	

General-use programming interface include files

The following C include files are provided to enable you to write programs that use the services of MQSeries. The files are supplied in library thlqual.SCSQC370.

CMQC	CMQXC	CMQCFC

The following PL/I include files are provided to enable you to write programs that use the services of MQSeries. The files are supplied in library thlqual.SCSQPLIC.

CMQP	CMQEPP	CMQXP	CMQCFP

Appendix B. Measured usage license charges with MQSeries for OS/390

Measured Usage License Charges (MULC) is a particular way of charging you for an IBM product that runs on an OS/390 system, based on how much use you make of the product. To determine the product usage, the OS/390 system records the amount of processor time that is used by the product when it executes.

OS/390 can measure how much processing time is spent in doing work on behalf of the MQSeries queue manager which is handling MQI calls, executing MQSeries commands, or performing some other action to support the messaging and queuing functions used by your application programs. The amount of processing time is recorded in a file at hourly intervals, and the hourly records are totalled at the end of a month. In this way, the total amount of time that has been used by the MQSeries for OS/390 product on your behalf is computed, and used to determine how much you should pay for your use of the MQSeries for OS/390 product that month.

MULC is implemented as follows:

- When MQSeries for OS/390 is installed, it identifies itself to OS/390, and requests that the *System Management Facilities (SMF)* mechanism within OS/390 is to automatically measure how much processor time is used by the MQSeries for OS/390 product.
- When enabled, the OS/390 usage measurement facility collects usage figures for each hour of the day, and generates usage records that are added to a report file on disk.
- At the end of one full month, these usage records are collected by a program, which generates a report of product usage for the month. This report is used to determine the charge for the MQSeries for OS/390 product.

More details on MULC can be found in the *MVS Support for Measured License Charges* manual.

Appendix C. 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

Trademarks

Glossary of terms and abbreviations

This glossary defines MQSeries terms and abbreviations used in this book. If you do not find the term you are looking for, see the Index or the *IBM Dictionary of Computing*, New York: McGraw-Hill, 1994.

This glossary includes terms and definitions from the *American National Dictionary for Information Systems*, ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies may be purchased from the American National Standards Institute, 11 West 42 Street, New York, New York 10036. Definitions are identified by the symbol (A) after the definition.

A

abend reason code. A 4-byte hexadecimal code that uniquely identifies a problem with MQSeries for OS/390. A complete list of MQSeries for OS/390 abend reason codes and their explanations is contained in the *MQSeries for OS/390 Messages and Codes* manual.

active log. See *recovery log*.

adapter. An interface between MQSeries for OS/390 and TSO, IMS, CICS, or batch address spaces. An adapter is an attachment facility that enables applications to access MQSeries services.

address space. The area of virtual storage available for a particular job.

address space identifier (ASID). A unique, system-assigned identifier for an address space.

administrator commands. MQSeries commands used to manage MQSeries objects, such as queues, processes, and namelists.

affinity. An association between objects that have some relationship or dependency upon each other.

alert. A message sent to a management services focal point in a network to identify a problem or an impending problem.

alert monitor. In MQSeries for OS/390, a component of the CICS adapter that handles unscheduled events occurring as a result of connection requests to MQSeries for OS/390.

alias queue object. An MQSeries object, the name of which is an alias for a base queue defined to the local queue manager. When an application or a queue manager uses an alias queue, the alias name is resolved and the requested operation is performed on the associated base queue.

allied address space. See *ally*.

ally. An OS/390 address space that is connected to MQSeries for OS/390.

alternate user security. A security feature in which the authority of one user ID can be used by another user ID; for example, to open an MQSeries object.

APAR. Authorized program analysis report.

application environment. The software facilities that are accessible by an application program. On the OS/390 platform, CICS and IMS are examples of application environments.

application queue. A queue used by an application.

archive log. See *recovery log*.

ARM. Automatic Restart Management

ASID. Address space identifier.

asynchronous messaging. A method of communication between programs in which programs place messages on message queues. With asynchronous messaging, the sending program proceeds with its own processing without waiting for a reply to its message. Contrast with *synchronous messaging*.

attribute. One of a set of properties that defines the characteristics of an MQSeries object.

authorization checks. Security checks that are performed when a user tries to issue administration commands against an object, for example to open a queue or connect to a queue manager.

authorized program analysis report (APAR). A report of a problem caused by a suspected defect in a current, unaltered release of a program.

Automatic Restart Management (ARM). An OS/390 recovery function that can improve the availability of specific batch jobs or started tasks, and therefore result in faster resumption of productive work.

B

backout. An operation that reverses all the changes made during the current unit of recovery or unit of work. After the operation is complete, a new unit of recovery or unit of work begins. Contrast with *commit*.

Glossary

basic mapping support (BMS). An interface between CICS and application programs that formats input and output display data and routes multiple-page output messages without regard for control characters used by various terminals.

BMS. Basic mapping support.

bootstrap data set (BSDS). A VSAM data set that contains:

- An inventory of all active and archived log data sets known to MQSeries for OS/390
- A wrap-around inventory of all recent MQSeries for OS/390 activity

The BSDS is required if the MQSeries for OS/390 subsystem has to be restarted.

browse. In message queuing, to use the MQGET call to copy a message without removing it from the queue. See also *get*.

browse cursor. In message queuing, an indicator used when browsing a queue to identify the message that is next in sequence.

BSDS. Bootstrap data set.

buffer pool. An area of main storage used for MQSeries for OS/390 queues, messages, and object definitions. See also *page set*.

C

call back. In MQSeries, a requester message channel initiates a transfer from a sender channel by first calling the sender, then closing down and awaiting a call back.

CCF. Channel control function.

CCSID. Coded character set identifier.

CDF. Channel definition file.

channel. See *message channel*.

channel control function (CCF). In MQSeries, a program to move messages from a transmission queue to a communication link, and from a communication link to a local queue, together with an operator panel interface to allow the setup and control of channels.

channel definition file (CDF). In MQSeries, a file containing communication channel definitions that associate transmission queues with communication links.

channel event. An event indicating that a channel instance has become available or unavailable. Channel events are generated on the queue managers at both ends of the channel.

checkpoint. A time when significant information is written on the log. Contrast with *syncpoint*.

CI. Control interval.

CL. Control Language.

client. A run-time component that provides access to queuing services on a server for local user applications. The queues used by the applications reside on the server. See also *MQSeries client*.

client application. An application, running on a workstation and linked to a client, that gives the application access to queuing services on a server.

client connection channel type. The type of MQI channel definition associated with an MQSeries client. See also *server connection channel type*.

cluster. A network of queue managers that are logically associated in some way.

cluster queue. A queue that is hosted by a cluster queue manager and made available to other queue managers in the cluster.

cluster queue manager. A queue manager that is a member of a cluster. A queue manager may be a member of more than one cluster.

cluster transmission queue. A transmission queue that transmits all messages from a queue manager to any other queue manager that is in the same cluster. The queue is called SYSTEM.CLUSTER.TRANSMIT.QUEUE.

coded character set identifier (CCSID). The name of a coded set of characters and their code point assignments.

command. In MQSeries, an administration instruction that can be carried out by the queue manager.

command prefix (CPF). In MQSeries for OS/390, a character string that identifies the queue manager to which MQSeries for OS/390 commands are directed, and from which MQSeries for OS/390 operator messages are received.

command processor. The MQSeries component that processes commands.

command server. The MQSeries component that reads commands from the system-command input queue, verifies them, and passes valid commands to the command processor.

commit. An operation that applies all the changes made during the current unit of recovery or unit of work. After the operation is complete, a new unit of recovery or unit of work begins. Contrast with *backout*.

completion code. A return code indicating how an MQI call has ended.

connect. To provide a queue manager connection handle, which an application uses on subsequent MQI calls. The connection is made either by the MQCONN or MQCONNX call, or automatically by the MQOPEN call.

connection handle. The identifier or token by which a program accesses the queue manager to which it is connected.

context. Information about the origin of a message.

context security. In MQSeries, a method of allowing security to be handled such that messages are obliged to carry details of their origins in the message descriptor.

control interval (CI). A fixed-length area of direct access storage in which VSAM stores records and creates distributed free spaces. The control interval is the unit of information that VSAM transmits to or from direct access storage.

controlled shutdown. See *quiesced shutdown*.

CPF. Command prefix.

Cross Systems Coupling Facility (XCF). Provides the OS/390 coupling services that allow authorized programs in a multisystem environment to communicate with programs on the same or different OS/390 systems.

coupling facility. On OS/390, a special logical partition that provides high-speed caching, list processing, and locking functions in a parallel sysplex.

D

datagram. The simplest message that MQSeries supports. This type of message does not require a reply.

DCI. Data conversion interface.

dead-letter queue (DLQ). A queue to which a queue manager or application sends messages that it cannot deliver to their correct destination.

default object. A definition of an object (for example, a queue) with all attributes defined. If a user defines an object but does not specify all possible attributes for that object, the queue manager uses default attributes in place of any that were not specified.

deferred connection. A pending event that is activated when a CICS subsystem tries to connect to MQSeries for OS/390 before MQSeries for OS/390 has been started.

dequeue. To remove a message from a queue. Contrast with *enqueue*.

distributed application. In message queuing, a set of application programs that can each be connected to a different queue manager, but that collectively constitute a single application.

distributed queue management (DQM). In message queuing, the setup and control of message channels to queue managers on other systems.

DLQ. Dead-letter queue.

DQM. Distributed queue management.

dual logging. A method of recording MQSeries for OS/390 activity, where each change is recorded on two data sets, so that if a restart is necessary and one data set is unreadable, the other can be used. Contrast with *single logging*.

dual mode. See *dual logging*.

dynamic queue. A local queue created when a program opens a model queue object. See also *permanent dynamic queue* and *temporary dynamic queue*.

E

enqueue. To put a message on a queue. Contrast with *dequeue*.

environment. See *application environment*.

ESM. External security manager.

event. See *channel event*, *instrumentation event*, *performance event*, and *queue manager event*.

event data. In an event message, the part of the message data that contains information about the event (such as the queue manager name, and the application that gave rise to the event). See also *event header*.

event header. In an event message, the part of the message data that identifies the event type of the reason code for the event.

event message. Contains information (such as the category of event, the name of the application that caused the event, and queue manager statistics) relating to the origin of an instrumentation event in a network of MQSeries systems.

event queue. The queue onto which the queue manager puts an event message after it detects an event. Each category of event (queue manager, performance, or channel event) has its own event queue.

external security manager (ESM). A security product that is invoked by the OS/390 System Authorization Facility. RACF is an example of an ESM.

F

FIFO. First-in-first-out.

first-in-first-out (FIFO). A queuing technique in which the next item to be retrieved is the item that has been in the queue for the longest time. (A)

forced shutdown. A type of shutdown of the CICS adapter where the adapter immediately disconnects from MQSeries for OS/390, regardless of the state of any currently active tasks. Contrast with *quiesced shutdown*.

G

GCPC. Generalized command preprocessor.

generalized command preprocessor (GCPC). An MQSeries for OS/390 component that processes MQSeries commands and runs them.

Generalized Trace Facility (GTF). An OS/390 service program that records significant system events, such as supervisor calls and start I/O operations, for the purpose of problem determination.

get. In message queuing, to use the MQGET call to remove a message from a queue.

global trace. An MQSeries for OS/390 trace option where the trace data comes from the entire MQSeries for OS/390 subsystem.

globally-defined object. On OS/390, an object whose definition is stored in the shared repository. The object is available to all queue managers in the queue-sharing group. See also *locally-defined object*.

GTF. Generalized Trace Facility.

H

handle. See *connection handle* and *object handle*.

hardened message. A message that is written to auxiliary (disk) storage so that the message will not be lost in the event of a system failure. See also *persistent message*.

I

immediate shutdown. In MQSeries, a shutdown of a queue manager that does not wait for applications to disconnect. Current MQI calls are allowed to complete, but new MQI calls fail after an immediate shutdown has been requested. Contrast with *quiesced shutdown* and *preemptive shutdown*.

inbound channel. A channel that listens for and receives messages from another queue manager. See also *shared inbound channel*.

in-doubt unit of recovery. In MQSeries, the status of a unit of recovery for which a syncpoint has been requested but not yet confirmed.

initialization input data sets. Data sets used by MQSeries for OS/390 when it starts up.

initiation queue. A local queue on which the queue manager puts trigger messages.

input/output parameter. A parameter of an MQI call in which you supply information when you make the call, and in which the queue manager changes the information when the call completes or fails.

input parameter. A parameter of an MQI call in which you supply information when you make the call.

instrumentation event. A facility that can be used to monitor the operation of queue managers in a network of MQSeries systems. MQSeries provides instrumentation events for monitoring queue manager resource definitions, performance conditions, and channel conditions. Instrumentation events can be used by a user-written reporting mechanism in an administration application that displays the events to a system operator. They also allow applications acting as agents for other administration networks to monitor reports and create the appropriate alerts.

Interactive Problem Control System (IPCS). A component of OS/390 that permits online problem management, interactive problem diagnosis, online debugging for disk-resident abend dumps, problem tracking, and problem reporting.

Interactive System Productivity Facility (ISPF). An IBM licensed program that serves as a full-screen editor and dialog manager. It is used for writing application programs, and provides a means of generating standard screen panels and interactive dialogues between the application programmer and terminal user.

IPCS. Interactive Problem Control System.

ISPF. Interactive System Productivity Facility.

L

listener. In MQSeries distributed queuing, a program that monitors for incoming network connections.

local definition. An MQSeries object belonging to a local queue manager.

local definition of a remote queue. An MQSeries object belonging to a local queue manager. This object defines the attributes of a queue that is owned by another queue manager. In addition, it is used for queue-manager aliasing and reply-to-queue aliasing.

local queue. A queue that belongs to the local queue manager. A local queue can contain a list of messages waiting to be processed. Contrast with *remote queue*.

local queue manager. The queue manager to which a program is connected and that provides message queuing services to the program. Queue managers to which a program is not connected are called *remote queue managers*, even if they are running on the same system as the program.

locally-defined object. On OS/390, an object whose definition is stored on page set zero. The definition can be accessed only by the queue manager that defined it. Also known as a *privately-defined object*.

log. In MQSeries, a file recording the work done by queue managers while they receive, transmit, and deliver messages, to enable them to recover in the event of failure.

logical unit of work (LUW). See *unit of work*.

M

machine check interrupt. An interruption that occurs as a result of an equipment malfunction or error. A machine check interrupt can be either hardware recoverable, software recoverable, or nonrecoverable.

MCA. Message channel agent.

MCI. Message channel interface.

message. In message queuing applications, a communication sent between programs. In system programming, information intended for the terminal operator or system administrator.

message channel. In distributed message queuing, a mechanism for moving messages from one queue manager to another. A message channel comprises two message channel agents (a sender at one end and a receiver at the other end) and a communication link. Contrast with *MQI channel*.

message channel agent (MCA). A program that transmits prepared messages from a transmission queue to a communication link, or from a communication link to a destination queue. See also *message queue interface*.

message channel interface (MCI). The MQSeries interface to which customer- or vendor-written programs that transmit messages between an MQSeries queue manager and another messaging system must conform. A part of the MQSeries Framework.

message descriptor. Control information describing the message format and presentation that is carried as part of an MQSeries message. The format of the message descriptor is defined by the MQMD structure.

message priority. In MQSeries, an attribute of a message that can affect the order in which messages on a queue are retrieved, and whether a trigger event is generated.

message queue. Synonym for *queue*.

message queue interface (MQI). The programming interface provided by the MQSeries queue managers. This programming interface allows application programs to access message queuing services.

message queuing. A programming technique in which each program within an application communicates with the other programs by putting messages on queues.

message sequence numbering. A programming technique in which messages are given unique numbers during transmission over a communication link. This enables the receiving process to check whether all messages are received, to place them in a queue in the original order, and to discard duplicate messages.

messaging. See *synchronous messaging* and *asynchronous messaging*.

model queue object. A set of queue attributes that act as a template when a program creates a dynamic queue.

MQI. Message queue interface.

MQI channel. Connects an MQSeries client to a queue manager on a server system, and transfers only MQI calls and responses in a bidirectional manner. Contrast with *message channel*.

MQSC. MQSeries commands.

MQSeries. A family of IBM licensed programs that provides message queuing services.

N

namelist. An MQSeries object that contains a list of names, for example, queue names.

nonpersistent message. A message that does not survive a restart of the queue manager. Contrast with *persistent message*.

null character. The character that is represented by X'00'.

O

object. In MQSeries, an object is a queue manager, a queue, a process definition, a channel, a namelist, or a storage class (OS/390 only).

object descriptor. A data structure that identifies a particular MQSeries object. Included in the descriptor are the name of the object and the object type.

object handle. The identifier or token by which a program accesses the MQSeries object with which it is working.

off-loading. In MQSeries for OS/390, an automatic process whereby a queue manager's active log is transferred to its archive log.

Open Transaction Manager Access (OTMA). A transaction-based, connectionless client/server protocol. It functions as an interface for host-based communications servers accessing IMS TM applications through the OS/390 Cross Systems Coupling Facility (XCF). OTMA is implemented in an OS/390 sysplex environment. Therefore, the domain of OTMA is restricted to the domain of XCF.

OTMA. Open Transaction Manager Access.

outbound channel. A channel that takes messages from a transmission queue and sends them to another queue manager. See also *shared outbound channel*.

output log-buffer. In MQSeries for OS/390, a buffer that holds recovery log records before they are written to the archive log.

output parameter. A parameter of an MQI call in which the queue manager returns information when the call completes or fails.

P

page set. A VSAM data set used when MQSeries for OS/390 moves data (for example, queues and messages) from buffers in main storage to permanent backing storage (DASD).

pending event. An unscheduled event that occurs as a result of a connect request from a CICS adapter.

percolation. In error recovery, the passing along a preestablished path of control from a recovery routine to a higher-level recovery routine.

performance event. A category of event indicating that a limit condition has occurred.

performance trace. An MQSeries trace option where the trace data is to be used for performance analysis and tuning.

permanent dynamic queue. A dynamic queue that is deleted when it is closed only if deletion is explicitly requested. Permanent dynamic queues are recovered if the queue manager fails, so they can contain persistent messages. Contrast with *temporary dynamic queue*.

persistent message. A message that survives a restart of the queue manager. Contrast with *nonpersistent message*.

ping. In distributed queuing, a diagnostic aid that uses the exchange of a test message to confirm that a message channel or a TCP/IP connection is functioning.

platform. In MQSeries, the operating system under which a queue manager is running.

point of recovery. In MQSeries for OS/390, the term used to describe a set of backup copies of MQSeries for OS/390 page sets and the corresponding log data sets required to recover these page sets. These backup copies provide a potential restart point in the event of page set loss (for example, page set I/O error).

preemptive shutdown. In MQSeries, a shutdown of a queue manager that does not wait for connected applications to disconnect, nor for current MQI calls to complete. Contrast with *immediate shutdown* and *quiesced shutdown*.

privately-defined object. In OS/390, an object whose definition is stored on page set zero. The definition can be accessed only by the queue manager that defined it. Also known as a *locally-defined object*.

process definition object. An MQSeries object that contains the definition of an MQSeries application. For example, a queue manager uses the definition when it works with trigger messages.

Q

queue. An MQSeries object. Message queuing applications can put messages on, and get messages from, a queue. A queue is owned and maintained by a queue manager. Local queues can contain a list of messages waiting to be processed. Queues of other types cannot contain messages—they point to other queues, or can be used as models for dynamic queues.

queue manager. A system program that provides queuing services to applications. It provides an application programming interface so that programs can access messages on the queues that the queue manager owns. See also *local queue manager* and *remote queue manager*. An MQSeries object that defines the attributes of a particular queue manager.

queue manager event. An event that indicates:

- An error condition has occurred in relation to the resources used by a queue manager. For example, a queue is unavailable.
- A significant change has occurred in the queue manager. For example, a queue manager has stopped or started.

queue-sharing group. In MQSeries for OS/390, a group of queue managers in the same sysplex that can access a single set of object definitions stored in the shared repository, and a single set of shared queues stored in the coupling facility. See also *shared queue*.

queuing. See *message queuing*.

quiesced shutdown. In MQSeries, a shutdown of a queue manager that allows all connected applications to disconnect. Contrast with *immediate shutdown* and *preemptive shutdown*. A type of shutdown of the CICS adapter where the adapter disconnects from MQSeries, but only after all the currently active tasks have been completed. Contrast with *forced shutdown*.

quiescing. In MQSeries, the state of a queue manager prior to it being stopped. In this state, programs are allowed to finish processing, but no new programs are allowed to start.

R

RBA. Relative byte address.

reason code. A return code that describes the reason for the failure or partial success of an MQI call.

receiver channel. In message queuing, a channel that responds to a sender channel, takes messages from a communication link, and puts them on a local queue.

recovery log. In MQSeries for OS/390, data sets containing information needed to recover messages, queues, and the MQSeries subsystem. MQSeries for OS/390 writes each record to a data set called the *active log*. When the active log is full, its contents are off-loaded to a DASD or tape data set called the *archive log*. Synonymous with *log*.

relative byte address (RBA). The displacement in bytes of a stored record or control interval from the beginning of the storage space allocated to the data set to which it belongs.

remote queue. A queue belonging to a remote queue manager. Programs can put messages on remote queues, but they cannot get messages from remote queues. Contrast with *local queue*.

remote queue manager. To a program, a queue manager that is not the one to which the program is connected.

remote queue object. See *local definition of a remote queue*.

remote queuing. In message queuing, the provision of services to enable applications to put messages on queues belonging to other queue managers.

reply message. A type of message used for replies to request messages. Contrast with *request message* and *report message*.

reply-to queue. The name of a queue to which the program that issued an MQPUT call wants a reply message or report message sent.

report message. A type of message that gives information about another message. A report message can indicate that a message has been delivered, has arrived at its destination, has expired, or could not be processed for some reason. Contrast with *reply message* and *request message*.

repository. A collection of information about the queue managers that are members of a cluster. This information includes queue manager names, their locations, their channels, what queues they host, and so on.

repository queue manager. A queue manager that hosts the *full repository* of information about a cluster.

requester channel. In message queuing, a channel that may be started remotely by a sender channel. The requester channel accepts messages from the sender channel over a communication link and puts the messages on the local queue designated in the message. See also *server channel*.

request message. A type of message used to request a reply from another program. Contrast with *reply message* and *report message*.

RESLEVEL. In MQSeries for OS/390, an option that controls the number of CICS user IDs checked for API-resource security in MQSeries for OS/390.

resolution path. The set of queues that are opened when an application specifies an alias or a remote queue on input to an MQOPEN call.

resource. Any facility of the computing system or operating system required by a job or task. In MQSeries for OS/390, examples of resources are buffer pools, page sets, log data sets, queues, and messages.

resource manager. An application, program, or transaction that manages and controls access to shared resources such as memory buffers and data sets. MQSeries, CICS, and IMS are resource managers.

Resource Recovery Services (RRS). An OS/390 facility that provides 2-phase syncpoint support across participating resource managers.

responder. In distributed queuing, a program that replies to network connection requests from another system.

resynch. In MQSeries, an option to direct a channel to start up and resolve any in-doubt status messages, but without restarting message transfer.

return codes. The collective name for completion codes and reason codes.

rollback. Synonym for *back out*.

RRS. Resource Recovery Services.

S

SAF. System Authorization Facility.

security enabling interface (SEI). The MQSeries interface to which customer- or vendor-written programs that check authorization, supply a user identifier, or perform authentication must conform. A part of the MQSeries Framework.

SEI. Security enabling interface.

sender channel. In message queuing, a channel that initiates transfers, removes messages from a transmission queue, and moves them over a communication link to a receiver or requester channel.

sequential delivery. In MQSeries, a method of transmitting messages with a sequence number so that the receiving channel can reestablish the message sequence when storing the messages. This is required where messages must be delivered only once, and in the correct order.

sequential number wrap value. In MQSeries, a method of ensuring that both ends of a communication link reset their current message sequence numbers at the same time. Transmitting messages with a sequence number ensures that the receiving channel can reestablish the message sequence when storing the messages.

server. (1) In MQSeries, a queue manager that provides queue services to client applications running on a remote workstation. (2) The program that responds to requests for information in the particular two-program, information-flow model of client/server. See also *client*.

server channel. In message queuing, a channel that responds to a requester channel, removes messages from a transmission queue, and moves them over a communication link to the requester channel.

server connection channel type. The type of MQI channel definition associated with the server that runs a queue manager. See also *client connection channel type*.

service interval. A time interval, against which the elapsed time between a put or a get and a subsequent get is compared by the queue manager in deciding whether the conditions for a service interval event have been met. The service interval for a queue is specified by a queue attribute.

service interval event. An event related to the service interval.

session ID. In MQSeries for OS/390, the CICS-unique identifier that defines the communication link to be used by a message channel agent when moving messages from a transmission queue to a link.

shared inbound channel. In MQSeries for OS/390, a channel that was started by a listener using the group port. The channel definition of a shared channel can be stored either on page set zero (private) or in the shared repository (global).

shared outbound channel. In MQSeries for OS/390, a channel that moves messages from a shared transmission queue. The channel definition of a shared channel can be stored either on page set zero (private) or in the shared repository (global).

shared queue. In MQSeries for OS/390, a type of local queue. The messages on the queue are stored in the *coupling facility* and can be accessed by one or more queue managers in a *queue-sharing group*. The definition of the queue is stored in the *shared repository*.

shared repository. In MQSeries for OS/390, a shared DB2 database that is used to hold object definitions that have been defined globally.

shutdown. See *immediate shutdown*, *preemptive shutdown*, and *quiesced shutdown*.

signaling. In MQSeries for OS/390 and MQSeries for Windows® 2.1, a feature that allows the operating system to notify a program when an expected message arrives on a queue.

single logging. A method of recording MQSeries for OS/390 activity where each change is recorded on one data set only. Contrast with *dual logging*.

single-phase backout. A method in which an action in progress must not be allowed to finish, and all changes that are part of that action must be undone.

single-phase commit. A method in which a program can commit updates to a queue without coordinating those updates with updates the program has made to resources controlled by another resource manager. Contrast with *two-phase commit*.

SIT. System initialization table.

storage class. In MQSeries for OS/390, a storage class defines the page set that is to hold the messages for a particular queue. The storage class is specified when the queue is defined.

store and forward. The temporary storing of packets, messages, or frames in a data network before they are retransmitted toward their destination.

subsystem. In OS/390, a group of modules that provides function that is dependent on OS/390. For example, MQSeries for OS/390 is an OS/390 subsystem.

supervisor call (SVC). An OS/390 instruction that interrupts a running program and passes control to the supervisor so that it can perform the specific service indicated by the instruction.

SVC. Supervisor call.

switch profile. In MQSeries for OS/390, a RACF profile used when MQSeries starts up or when a refresh security command is issued. Each switch profile that MQSeries detects turns off checking for the specified resource.

synchronous messaging. A method of communication between programs in which programs place messages on message queues. With synchronous messaging, the sending program waits for a reply to its message before resuming its own processing. Contrast with *asynchronous messaging*.

syncpoint. An intermediate or end point during processing of a transaction at which the transaction's protected resources are consistent. At a syncpoint, changes to the resources can safely be committed, or they can be backed out to the previous syncpoint.

sysplex. A multiple OS/390-system environment that allows multiple-console support (MCS) consoles to receive console messages and send operator commands across systems.

System Authorization Facility (SAF). An OS/390 facility through which MQSeries for OS/390 communicates with an external security manager such as RACF.

system.command.input queue. A local queue on which application programs can put MQSeries commands. The commands are retrieved from the queue by the command server, which validates them and passes them to the command processor to be run.

system control commands. Commands used to manipulate platform-specific entities such as buffer pools, storage classes, and page sets.

system initialization table (SIT). A table containing parameters used by CICS on start up.

T

target library high-level qualifier (thlqual). High-level qualifier for OS/390 target data set names.

task control block (TCB). An OS/390 control block used to communicate information about tasks within an address space that are connected to an OS/390 subsystem such as MQSeries for OS/390 or CICS.

task switching. The overlapping of I/O operations and processing between several tasks. In MQSeries for OS/390, the task switcher optimizes performance by allowing some MQI calls to be executed under subtasks rather than under the main CICS TCB.

TCB. Task control block.

temporary dynamic queue. A dynamic queue that is deleted when it is closed. Temporary dynamic queues are not recovered if the queue manager fails, so they can contain nonpersistent messages only. Contrast with *permanent dynamic queue*.

termination notification. A pending event that is activated when a CICS subsystem successfully connects to MQSeries for OS/390.

thlqual. Target library high-level qualifier.

thread. In MQSeries, the lowest level of parallel execution available on an operating system platform.

time-independent messaging. See *asynchronous messaging*.

TMI. Trigger monitor interface.

trace. In MQSeries, a facility for recording MQSeries activity. The destinations for trace entries can include GTF and the system management facility (SMF).

tranid. See *transaction identifier*.

transaction identifier. In CICS, a name that is specified when the transaction is defined, and that is used to invoke the transaction.

transmission program. See *message channel agent*.

transmission queue. A local queue on which prepared messages destined for a remote queue manager are temporarily stored.

trigger event. An event (such as a message arriving on a queue) that causes a queue manager to create a trigger message on an initiation queue.

triggering. In MQSeries, a facility allowing a queue manager to start an application automatically when predetermined conditions on a queue are satisfied.

trigger message. A message containing information about the program that a trigger monitor is to start.

trigger monitor. A continuously-running application serving one or more initiation queues. When a trigger message arrives on an initiation queue, the trigger monitor retrieves the message. It uses the information in the trigger message to start a process that serves the queue on which a trigger event occurred.

trigger monitor interface (TMI). The MQSeries interface to which customer- or vendor-written trigger monitor programs must conform. A part of the MQSeries Framework.

two-phase commit. A protocol for the coordination of changes to recoverable resources when more than one resource manager is used by a single transaction. Contrast with *single-phase commit*.

U

undo/redo record. A log record used in recovery. The redo part of the record describes a change to be made to an MQSeries object. The undo part describes how to back out the change if the work is not committed.

unit of recovery. A recoverable sequence of operations within a single resource manager. Contrast with *unit of work*.

unit of work. A recoverable sequence of operations performed by an application between two points of consistency. A unit of work begins when a transaction starts or after a user-requested syncpoint. It ends either at a user-requested syncpoint or at the end of a transaction. Contrast with *unit of recovery*.

utility. In MQSeries, a supplied set of programs that provide the system operator or system administrator with facilities in addition to those provided by the MQSeries commands. Some utilities invoke more than one function.

X

XCF. Cross Systems Coupling Facility.

Bibliography

This section describes the documentation available for all current MQSeries products.

MQSeries cross-platform publications

Most of these publications, which are sometimes referred to as the MQSeries "family" books, apply to all MQSeries Level 2 products. The latest MQSeries Level 2 products are:

- MQSeries for AIX, V5.1
- MQSeries for AS/400, V5.1
- MQSeries for AT&T GIS UNIX V2.2
- MQSeries for Compaq (DIGITAL) OpenVMS, V2.2.1.1
- MQSeries for Compaq Tru64 UNIX, V5.1
- MQSeries for HP-UX, V5.1
- MQSeries for OS/2 Warp, V5.1
- MQSeries for OS/390, V5.2
- MQSeries for SINIX and DC/OSx, V2.2
- MQSeries for Sun Solaris, V5.1
- MQSeries for Sun Solaris, Intel Platform Edition, V5.1
- MQSeries for Tandem NonStop Kernel, V2.2.0.1
- MQSeries for VSE/ESA V2.1
- MQSeries for Windows V2.0
- MQSeries for Windows V2.1
- MQSeries for Windows NT, V5.1

The MQSeries cross-platform publications are:

- *MQSeries Brochure*, G511-1908
- *An Introduction to Messaging and Queuing*, GC33-0805
- *MQSeries Intercommunication*, SC33-1872
- *MQSeries Queue Manager Clusters*, SC34-5349
- *MQSeries Clients*, GC33-1632
- *MQSeries System Administration*, SC33-1873
- *MQSeries MQSC Command Reference*, SC33-1369
- *MQSeries Event Monitoring*, SC34-5760
- *MQSeries Programmable System Management*, SC33-1482
- *MQSeries Administration Interface Programming Guide and Reference*, SC34-5390
- *MQSeries Messages*, GC33-1876
- *MQSeries Application Programming Guide*, SC33-0807
- *MQSeries Application Programming Reference*, SC33-1673
- *MQSeries Programming Interfaces Reference Summary*, SX33-6095
- *MQSeries Using C++*, SC33-1877
- *MQSeries Using Java™*, SC34-5456
- *MQSeries Application Messaging Interface*, SC34-5604

MQSeries platform-specific publications

Each MQSeries product is documented in at least one platform-specific publication, in addition to the MQSeries family books.

MQSeries for AIX, V5.1

MQSeries for AIX Quick Beginnings, GC33-1867

MQSeries for AS/400, V5.1

MQSeries for AS/400® Quick Beginnings, GC34-5557

MQSeries for AS/400 System Administration, SC34-5558

MQSeries for AS/400 Application Programming Reference (ILE RPG), SC34-5559

MQSeries for AT&T GIS UNIX V2.2

MQSeries for AT&T GIS UNIX® System Management Guide, SC33-1642

MQSeries for Compaq (DIGITAL) OpenVMS, V2.2.1.1

MQSeries for Digital OpenVMS System Management Guide, GC33-1791

MQSeries for Compaq Tru64 UNIX, V5.1

MQSeries for Compaq Tru64 UNIX Quick Beginnings, GC34-5684

MQSeries for HP-UX, V5.1

MQSeries for HP-UX Quick Beginnings, GC33-1869

MQSeries for OS/2 Warp, V5.1

MQSeries for OS/2 Warp Quick Beginnings, GC33-1868

MQSeries for OS/390, V5.2

MQSeries for OS/390 Concepts and Planning Guide, GC34-5650

MQSeries for OS/390 System Setup Guide, SC34-5651

MQSeries for OS/390 System Administration Guide, SC34-5652

MQSeries for OS/390 Problem Determination Guide, GC34-5892

MQSeries for OS/390 Messages and Codes, GC34-5891

MQSeries for OS/390 Licensed Program Specifications, GC34-5893

MQSeries for OS/390 Program Directory

MQSeries link for R/3 Version 1.2

MQSeries link for R/3 User's Guide, GC33-1934

MQSeries for SINIX and DC/OSx, V2.2

MQSeries for SINIX and DC/OSx System Management Guide, GC33-1768

MQSeries for Sun Solaris, V5.1

MQSeries for Sun Solaris Quick Beginnings, GC33-1870

MQSeries for Sun Solaris, Intel Platform Edition, V5.1

MQSeries for Sun Solaris, Intel Platform Edition Quick Beginnings, GC34-5851

MQSeries for Tandem NonStop Kernel, V2.2.0.1

MQSeries for Tandem NonStop Kernel System Management Guide, GC33-1893

MQSeries for VSE/ESA V2.1

MQSeries for VSE/ESA Version 2 Release 1 Licensed Program Specifications, GC34-5365

MQSeries for VSE/ESA™ System Management Guide, GC34-5364

MQSeries for Windows V2.0

MQSeries for Windows User's Guide, GC33-1822

MQSeries for Windows V2.1

MQSeries for Windows User's Guide, GC33-1965

MQSeries for Windows NT, V5.1

MQSeries for Windows NT Quick Beginnings, GC34-5389

MQSeries for Windows NT Using the Component Object Model Interface, SC34-5387

MQSeries LotusScript Extension, SC34-5404

Softcopy books

Most of the MQSeries books are supplied in both hardcopy and softcopy formats.

HTML format

Relevant MQSeries documentation is provided in HTML format with these MQSeries products:
- MQSeries for AIX, V5.1
- MQSeries for AS/400, V5.1
- MQSeries for Compaq Tru64 UNIX, V5.1
- MQSeries for HP-UX, V5.1
- MQSeries for OS/2 Warp, V5.1
- MQSeries for OS/390, V5.2
- MQSeries for Sun Solaris, V5.1
- MQSeries for Windows NT, V5.1 (compiled HTML)
- MQSeries link for R/3 V1.2

The MQSeries books are also available in HTML format from the MQSeries product family Web site at:

`http://www.ibm.com/software/mqseries/`

Portable Document Format (PDF)

PDF files can be viewed and printed using the Adobe Acrobat Reader.

If you need to obtain the Adobe Acrobat Reader, or would like up-to-date information about the platforms on which the Acrobat Reader is supported, visit the Adobe Systems Inc. Web site at:

`http://www.adobe.com/`

PDF versions of relevant MQSeries books are supplied with these MQSeries products:
- MQSeries for AIX, V5.1
- MQSeries for AS/400, V5.1
- MQSeries for Compaq Tru64 UNIX, V5.1
- MQSeries for HP-UX, V5.1
- MQSeries for OS/2 Warp, V5.1
- MQSeries for OS/390, V5.2
- MQSeries for Sun Solaris, V5.1
- MQSeries for Windows NT, V5.1
- MQSeries link for R/3 V1.2

PDF versions of all current MQSeries books are also available from the MQSeries product family Web site at:

```
http://www.ibm.com/software/mqseries/
```

BookManager® format

The MQSeries library is supplied in IBM BookManager format on a variety of online library collection kits, including the *Transaction Processing and Data* collection kit, SK2T-0730. You can view the softcopy books in IBM BookManager format using the following IBM licensed programs:

BookManager READ/2
BookManager READ/6000
BookManager READ/DOS
BookManager READ/MVS
BookManager READ/VM
BookManager READ for Windows

PostScript format

The MQSeries library is provided in PostScript (.PS) format with many MQSeries Version 2 products. Books in PostScript format can be printed on a PostScript printer or viewed with a suitable viewer.

Windows Help format

The *MQSeries for Windows User's Guide* is provided in Windows Help format with MQSeries for Windows Version 2.0 and MQSeries for Windows Version 2.1.

MQSeries information available on the Internet

The MQSeries product family Web site is at:

```
http://www.ibm.com/software/mqseries/
```

By following links from this Web site you can:

- Obtain latest information about the MQSeries product family.
- Access the MQSeries books in HTML and PDF formats.
- Download MQSeries SupportPacs.

Related publications

For information about other products that are referred to in this book, see the following books:

OS/390

- *OS/390 Planning For Installation*, GC28-1726
- *MVS Installation Exits*, GC28-1753
- *MVS Setting up a Sysplex*, GC28-1779
- *MVS Programming: Callable Services for High Level Languages*, GC28-1768
- *MVS Support for Measured License Charges*, GC28-1098

CICS Transaction Server for OS/390

- *XRF Guide*, SC33-0661
- *Customization Guide*, SC33-1683
- *Intercommunication Guide*, SC33-1695
- *Recovery and Restart Guide*, SC33-1698
- *Internet and External Interfaces Guide*, SC33-1944

CICS for MVS/ESA Version 4

- *XRF Guide*, SC33-0661
- *Customization Guide*, SC33-1165
- *Intercommunication Guide*, SC33-1181
- *Recovery and Restart Guide*, SC33-1182

IMS

- *Customization Guide*, SC26-8020
- *Administration Guide: System*, SC26-8013
- *Open Transaction Manager Access Guide*, SC26-8026

DFSMS/MVS

- *Access Method Services for VSAM*, SC26-4905
- *Access Method Services for the Integrated Catalog Facility*, SC26-4906

Other products

- *DFP Storage Administration Reference*, SC26-4566
- *DB2 for OS/390 Installation Guide*, GC26-8970
- *MQSeries Workflow for OS/390: Customization and Administration*, SC33-7030
- *MQSeries Workflow: Concepts and Architecture*, GH12-6285
- *Data Facility Hierarchical Storage Manager User's Guide*, SH35-0093

Related publications

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 44–1962–870229
  - From within the U.K., use 01962–870229
- 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 the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

GC34-5650-00