MQSeries

Programmable System Management

SC33-1482-06

MQSeries

Programmable System Management

SC33-1482-06

Note!

Before using this information and the product it supports, be sure to read the general information under Appendix D, “Notices” on page 489.

Seventh edition (February 1998)

This edition applies to the following products:

- MQSeries for AIX Version 5
- MQSeries for AS/400 Version 4 Release 2
- MQSeries for AT&T GIS UNIX Version 2 Release 2
- MQSeries for Digital OpenVMS Version 2 Release 2
- MQSeries for HP-UX Version 5
- MQSeries for MVS/ESA Version 1 Release 2
- MQSeries for OS/2 Warp Version 5
- MQSeries for SINIX and DC/OSx Version 2 Release 2
- MQSeries for SunOS Version 2 Release 2
- MQSeries for Sun Solaris Version 5
- MQSeries for Tandem NonStop Kernel Version 2 Release 2
- MQSeries for Windows NT Version 5
- MQSeries for Windows Version 2 Release 1

and to any subsequent releases and modifications until otherwise indicated in new editions.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

At the back of this publication is a page titled “Sending your comments to IBM”. If you want to make comments, but the methods described are not available to you, please address them to:

IBM United Kingdom Laboratories,
Information Development,
Mail Point 095,
Hursley Park,
Winchester,
Hampshire,
England,
SO21 2JN

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

Contents

Figures

Tables

Tables

About this book

This book describes the facilities available on MQSeries products for:

- Monitoring instrumentation events in a network of connected systems that use IBM MQSeries products in different operating system environments.
- Writing programs using the MQSeries Programmable Command Formats (PCFs) to administer IBM MQSeries systems either locally or remotely.
- Installable services which extend the facilities available to a queue manager.

This table shows which facilities are offered on different MQSeries platforms, together with the short name used in the book.

Table 1. MQSeries programmable system management

Platform MQSeries for	Short name	Event monitoring	PCF commands	Installable services
AS/400	OS/400	√	√	No
Digital OpenVMS** V2.2	OpenVMS	√	√	√
MVS/ESA	MVS/ESA	√	No	No
OS/2	OS/2	√	√	√
Tandem NonStop Kernel V2.2	Tandem NSK	√	√	√
UNIX systems **see Note below**	UNIX systems	√	√	√
Windows NT	Windows NT	√	√	√
Windows V2.0	16-bit Windows	No	No	No
Windows V2.1	32-bit Windows	√	√	No

Note: In this book references to MQSeries for "UNIX systems" include:

IBM MQSeries for AIX Version 5
IBM MQSeries for AT&T** GIS UNIX Version 2 Release 2[1]
IBM MQSeries for HP-UX** Version 5
IBM MQSeries for SINIX** and DC/OSx** Version 2.2
IBM MQSeries for SunOS** Version 2.2
IBM MQSeries for Sun Solaris** Version 5

The following table lists the MQSeries products available for Windows, and shows the Windows platforms on which each runs.

[1] This platform has become NCR UNIX SVR4 MP-RAS, R3.0

MQSeries product	Windows 3.1	Windows 95	Windows NT
MQSeries for Windows Client	Yes	Yes	Yes
MQSeries for Windows NT	No	No	Yes
MQSeries for Windows V2.0	Yes	Yes	No
MQSeries for Windows V2.1	No	Yes	Yes

MQSeries for Windows Version 2.1 support most of the features of the MQI described in this book. For information on this product, see the *MQSeries for Windows User's Guide*.

Who this book is for

Primarily, this book is intended for system programmers who write programs to monitor and administer MQSeries products. To do this they may need to use the event messages, the Programmable Command Formats, and the installable services that are described in this book.

What you need to know

You should have:

- Experience in writing systems management applications
- An understanding of the Message Queue Interface (MQI)
- Experience of MQSeries programs in general, or familiarity with the content of the other books in the MQSeries Library

How to use this book

There are three parts to this book:

- Part 1 – Event monitoring

  This part of the book describes how to monitor significant events in a network of connected systems that use IBM MQSeries products, in different operating system environments.

- Part 2 – Programmable Command Formats (PCFs)

  This part of the book describes the MQSeries (PCFs). PCFs are the formats of command and response messages that are sent between an MQSeries systems management application, or other program, and an MQSeries queue manager.

- Part 3 – Installable services

  This part of the book describes the MQSeries installable services. It includes full reference material for the interface to the installable services.

Go to the part that you are interested in; there is an introduction and discussion of each topic before the reference material.

Event monitoring

The first three chapters contain a description of the different types of event, and provide guidance on their use.

Chapter 4, “Event message reference” on page 35 contains the reference material for the event messages. Chapter 5, “Example of using instrumentation events” on page 111 contains a fragment of a C program to illustrate the use of events.

Programmable Command Formats

The first two chapters (chapters 6 and 7) contain introduction and guidance material. If you are using PCFs, you are advised to read all of this part.

Chapter 8, “Definitions of the Programmable Command Formats” on page 135 and Chapter 9, “Structures used for commands and responses” on page 333 contain the reference material. See Chapter 10, “Example of using PCFs” on page 353 for an example of how PCFs could be used.

Installable services

The first chapter (chapter 11) contains a description of the available installable services. You must read this chapter if you are going to use any of the installable services. Read the following chapters as necessary, according to the services that you are going to install. Three services are described:

- Chapter 12, “Authorization service” on page 375.
- Chapter 13, “Name service” on page 383.
- Chapter 14, “User identifier service” on page 389.

Chapter 15, “Installable services interface” on page 395 describes the interface for each service.

Appendixes

The error codes that apply to PCF commands and responses are listed in Appendix A, “Error codes” on page 455.

The values of constants for events, commands, responses and installable services are given in Appendix B, “Constants” on page 473.

The various header, COPY, and INCLUDE files that are provided to assist applications with the processing of event messages, PCF commands, and installable services are given in Appendix C, “Header, COPY, and INCLUDE files” on page 485.

MQSeries publications

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 V5.0
- MQSeries for AS/400 V4R2
- MQSeries for AT&T GIS UNIX V2.2
- MQSeries for Digital OpenVMS V2.2
- MQSeries for HP-UX V5.0
- MQSeries for MVS/ESA V1.2
- MQSeries for OS/2 Warp V5.0
- MQSeries for SINIX and DC/OSx V2.2
- MQSeries for SunOS V2.2
- MQSeries for Sun Solaris V5.0
- MQSeries for Tandem NonStop Kernel V2.2
- MQSeries Three Tier
- MQSeries for Windows V2.0
- MQSeries for Windows V2.1
- MQSeries for Windows NT V5.0

Any exceptions to this general rule are indicated. (Publications that support the MQSeries Level 1 products are listed in "MQSeries Level 1 product publications" on page xiii. For a functional comparison of the Level 1 and Level 2 MQSeries products, see the *MQSeries Planning Guide*.)

MQSeries Brochure
The *MQSeries Brochure*, G511-1908, gives a brief introduction to the benefits of MQSeries. It is intended to support the purchasing decision, and describes some authentic customer use of MQSeries.

MQSeries: An Introduction to Messaging and Queuing
MQSeries: An Introduction to Messaging and Queuing, GC33-0805, describes briefly what MQSeries is, how it works, and how it can solve some classic interoperability problems. This book is intended for a more technical audience than the *MQSeries Brochure*.

MQSeries Planning Guide
The *MQSeries Planning Guide*, GC33-1349, describes some key MQSeries concepts, identifies items that need to be considered before MQSeries is installed, including storage requirements, backup and recovery, security, and migration from earlier releases, and specifies hardware and software requirements for every MQSeries platform.

MQSeries Intercommunication
The *MQSeries Intercommunication* book, SC33-1872, defines the concepts of distributed queuing and explains how to set up a distributed queuing network in a variety of MQSeries environments. In particular, it demonstrates how to (1) configure communications to and from a representative sample of MQSeries products, (2) create required MQSeries objects, and (3) create and configure MQSeries channels. The use of channel exits is also described.

MQSeries Clients
The *MQSeries Clients* book, GC33-1632, describes how to install, configure, use, and manage MQSeries client systems.

MQSeries System Administration
The *MQSeries System Administration* book, SC33-1873, supports day-to-day management of local and remote MQSeries objects. It includes topics such as security, recovery and restart, transactional support, problem determination, the dead-letter queue handler, and the MQSeries links for Lotus Notes**. It also includes the syntax of the MQSeries control commands.

This book applies to the following MQSeries products only:

- MQSeries for AIX V5.0
- MQSeries for HP-UX V5.0
- MQSeries for OS/2 Warp V5.0
- MQSeries for Sun Solaris V5.0
- MQSeries for Windows NT V5.0

MQSeries Command Reference
The *MQSeries Command Reference*, SC33-1369, contains the syntax of the MQSC commands, which are used by MQSeries system operators and administrators to manage MQSeries objects.

MQSeries Programmable System Management
The *MQSeries Programmable System Management* book, SC33-1482, provides both reference and guidance information for users of MQSeries events, programmable command formats (PCFs), and installable services.

MQSeries Messages
The *MQSeries Messages* book, GC33-1876, which describes “AMQ” messages issued by MQSeries, applies to these MQSeries products only:

- MQSeries for AIX V5.0
- MQSeries for HP-UX V5.0
- MQSeries for OS/2 Warp V5.0
- MQSeries for Sun Solaris V5.0
- MQSeries for Windows NT V5.0
- MQSeries for Windows V2.0
- MQSeries for Windows V2.1

This book is available in softcopy only.

MQSeries Application Programming Guide
The *MQSeries Application Programming Guide*, SC33-0807, provides guidance information for users of the message queue interface (MQI). It describes how to design, write, and build an MQSeries application. It also includes full descriptions of the sample programs supplied with MQSeries.

MQSeries Application Programming Reference
The *MQSeries Application Programming Reference*, SC33-1673, provides comprehensive reference information for users of the MQI. It includes: data-type descriptions; MQI call syntax; attributes of MQSeries objects; return codes; constants; and code-page conversion tables.

MQSeries Application Programming Reference Summary
The *MQSeries Application Programming Reference Summary*, SX33-6095, summarizes the information in the *MQSeries Application Programming Reference* manual.

MQSeries Using C++
MQSeries Using C++, SC33-1877, provides both guidance and reference information for users of the MQSeries C++ programming-language binding to the MQI. MQSeries C++ is supported by V5.0 of MQSeries for AIX, HP-UX, OS/2 Warp, Sun Solaris, and Windows NT, and by MQSeries clients supplied with those products and installed in the following environments:

- AIX
- HP-UX
- OS/2
- Sun Solaris
- Windows NT
- Windows 3.1
- Windows 95

MQSeries C++ is also supported by MQSeries for AS/400 V4R2.

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

MQSeries for AIX V5.0 Quick Beginnings, GC33-1867

MQSeries for AS/400

MQSeries for AS/400 Version 4 Release 2 Licensed Program Specifications, GC33-1958

MQSeries for AS/400 Version 4 Release 2 Administration Guide, GC33-1956

MQSeries for AS/400 Version 4 Release 2 Application Programming Reference (RPG), SC33-1957

MQSeries for AT&T GIS UNIX

MQSeries for AT&T GIS UNIX Version 2.2 System Management Guide, SC33-1642

MQSeries for Digital OpenVMS

MQSeries for Digital OpenVMS Version 2.2 System Management Guide, GC33-1791

MQSeries for HP-UX

MQSeries for HP-UX V5.0 Quick Beginnings, GC33-1869

MQSeries for MVS/ESA

MQSeries for MVS/ESA Version 1 Release 2 Licensed Program Specifications, GC33-1350

MQSeries for MVS/ESA Version 1 Release 2 Program Directory

MQSeries for MVS/ESA Version 1 Release 2 System Management Guide, SC33-0806

MQSeries for MVS/ESA Version 1 Release 2 Messages and Codes, GC33-0819

MQSeries for MVS/ESA Version 1 Release 2 Problem Determination Guide, GC33-0808

MQSeries for OS/2 Warp

MQSeries for OS/2 Warp V5.0 Quick Beginnings, GC33-1868

MQSeries link for R/3

MQSeries link for R/3 Version 1.0 User's Guide, GC33-1934

MQSeries for SINIX and DC/OSx

MQSeries for SINIX and DC/OSx Version 2.2 System Management Guide, GC33-1768

MQSeries for SunOS

MQSeries for SunOS Version 2.2 System Management Guide, GC33-1772

MQSeries for Sun Solaris

MQSeries for Sun Solaris V5.0 Quick Beginnings, GC33-1870

MQSeries for Tandem NonStop Kernel

MQSeries for Tandem NonStop Kernel Version 2.2 System Management Guide, GC33-1893

MQSeries Three Tier

MQSeries Three Tier Administration Guide, SC33-1451
MQSeries Three Tier Reference Summary, SX33-6098
MQSeries Three Tier Application Design, SC33-1636
MQSeries Three Tier Application Programming, SC33-1452

MQSeries for Windows

MQSeries for Windows Version 2.0 User's Guide, GC33-1822

MQSeries for Windows Version 2.1 User's Guide, GC33-1965

MQSeries for Windows NT

MQSeries for Windows NT V5.0 Quick Beginnings, GC33-1871

MQSeries Level 1 product publications

For information about the MQSeries Level 1 products, see the following publications:

MQSeries: Concepts and Architecture, GC33-1141

MQSeries Version 1 Products for UNIX Operating Systems Messages and Codes, SC33-1754

MQSeries for SCO UNIX Version 1.4 User's Guide, SC33-1378

MQSeries for UnixWare Version 1.4.1 User's Guide, SC33-1379

MQSeries for VSE/ESA Version 1 Release 4 Licensed Program Specifications, GC33-1483

MQSeries for VSE/ESA Version 1 Release 4 User's Guide, SC33-1142

Softcopy books

Most of the MQSeries books are supplied in both hardcopy and softcopy formats.

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 products, including all MQSeries V5.0 products. Books in PostScript format can be printed on a PostScript printer or viewed with a suitable viewer.

HTML format

The MQSeries documentation is provided in HTML format with these MQSeries products:

- MQSeries for AIX V5.0
- MQSeries for HP-UX V5.0
- MQSeries for OS/2 Warp V5.0
- MQSeries for Sun Solaris V5.0
- MQSeries for Windows NT V5.0

The MQSeries books are also available from the MQSeries product family Web site:

| `http://www.software.ibm.com/ts/mqseries/`

Information Presentation Facility (IPF) format

In the OS/2 environment, the MQSeries documentation is supplied in IBM IPF format on the MQSeries product CD-ROM.

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

MQSeries web site

The MQSeries product family Web site is at:

```
http://www.software.ibm.com/ts/mqseries/
```

By following links from this Web site you can:

- Obtain latest information about the MQSeries product family.
- Access the MQSeries books in HTML format.
- Download MQSeries SupportPacs.

Summary of changes

This section lists the major revisions to this book for the current edition and the preceding two editions.

Changes to this edition, SC33-1482-06

Changes to the book for this edition are marked by vertical bars in the left margin; these changes include:

- New versions of the following products:
  - MQSeries for AS/400
  - MQSeries for Tandem NonStop Kernel
- Minor technical and editorial improvements throughout the book

Changes to the Sixth Edition include:

- New versions of the following products:
  - MQSeries for AIX
  - MQSeries for HP-UX
  - MQSeries for OS/2
  - MQSeries for Sun Solaris
  - MQSeries for Windows NT
- The changes to the products include additional support for:
  - Distribution lists
  - Direct SPX support
  - Channel heartbeats
  - Fast messages
  - Auto-definition of channels
  - MCA exit chaining

Changes for the Fifth Edition include:

- MQSeries for OS/400 V3R2 is included, adding support for Event monitoring and enhanced support for Programmable Command Formats (PCFs).
- MQSeries for MVS/ESA V1.1.4 is included (Event monitoring).
- Two new Events are added, applicable to MQSeries for MVS/ESA V1.1.4, for the IMS Bridge:
  - Bridge Stopped
  - Bridge Started

The following new products are now included:

- MQSeries for SINIX and DC/OSx Version 2.2
- MQSeries for SunOS Version 2.2
- MQSeries for Sun Solaris Version 2.2

References to “UNIX systems” include AIX, HP-UX, AT&T GIS UNIX, SINIX and DC/OSx, SunOS and Sun Solaris systems.

The following changes were included in the BookManager version of SC33-1482-03 available in March 1996 on the Transaction Processing and Data Collection Kit, SK2T-0730:

- MQSeries for Windows NT V2.0 included.
- MQSeries for HP-UX Version 2.2.1 included.

Part 1. Event monitoring

Chapter 1. Using instrumentation events to monitor queue managers

MQSeries instrumentation events provide information about errors, warnings, and other significant occurrences in a queue manager. You can, therefore, use these events to monitor the operation of queue managers (in conjunction with other methods such as NetView). This chapter tells you what these events are, and how you use them.

Instrumentation events are supported by:

MQSeries for AIX
MQSeries for AS/400
MQSeries for AT&T GIS UNIX
MQSeries for Digital OpenVMS
MQSeries for HP-UX
MQSeries for MVS/ESA
MQSeries for OS/2 Warp
MQSeries for SINIX and DC/OSx
MQSeries for SunOS
MQSeries for Sun Solaris
MQSeries for Tandem NonStop Kernel
MQSeries for Windows NT
MQSeries for Windows

Monitoring queue managers

Instrumentation events can be generated for queue managers running on Digital OpenVMS, MVS/ESA, OS/2, OS/400, Tandem NonStopKernel, Windows 95, Windows NT, and UNIX platforms. By incorporating these events into your own system management application, you can monitor the activities across many queue managers, across many different nodes, for multiple MQSeries applications. In particular, you can monitor all the nodes in your system from a single node (for those nodes that support MQSeries events) as shown in Figure 1 on page 4.

Instrumentation events can be reported through a user-written reporting mechanism to an administration application that supports the presentation of the events to an operator.

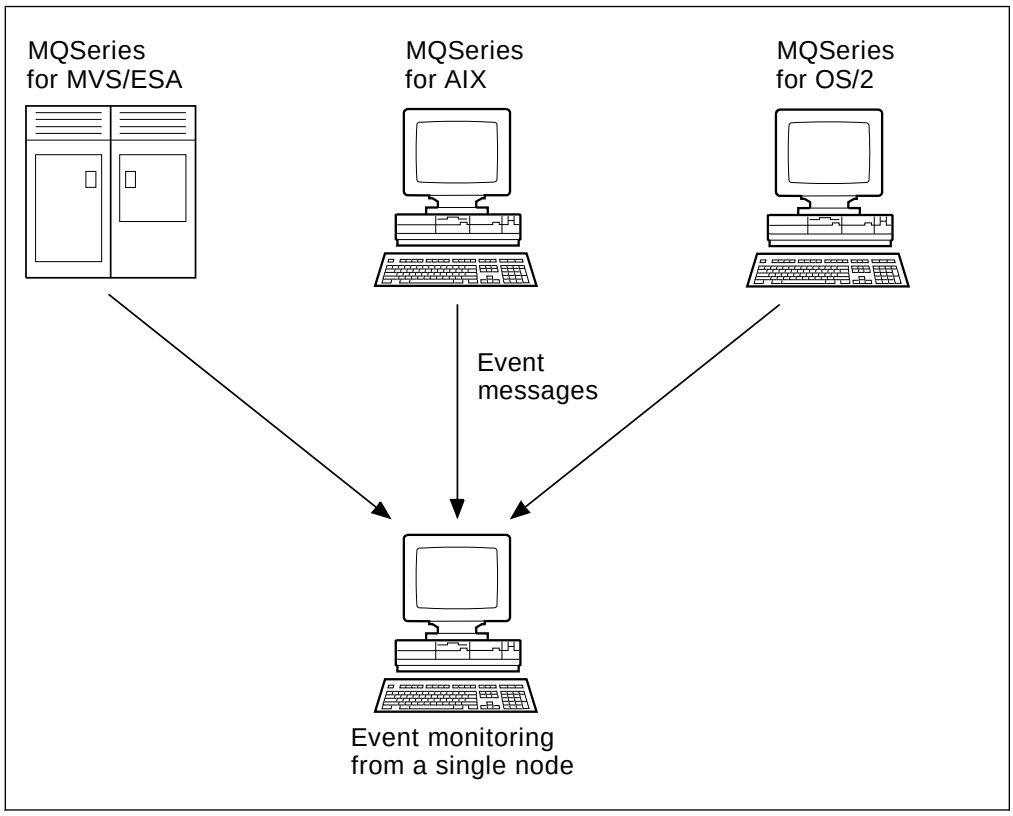


Figure 1. Monitoring queue managers across different platforms, on a single node

Instrumentation events also enable applications acting as agents for other administration networks, for example NetView, to monitor reports and create the appropriate alerts.

What instrumentation events are

In MQSeries, an instrumentation event is a logical combination of conditions that is detected by a queue manager or channel instance. The result of such an event is that the queue manager or channel instance puts a special message, called an *event message*, on an event queue. Event queues are described in “Event notification through event queues” on page 6.

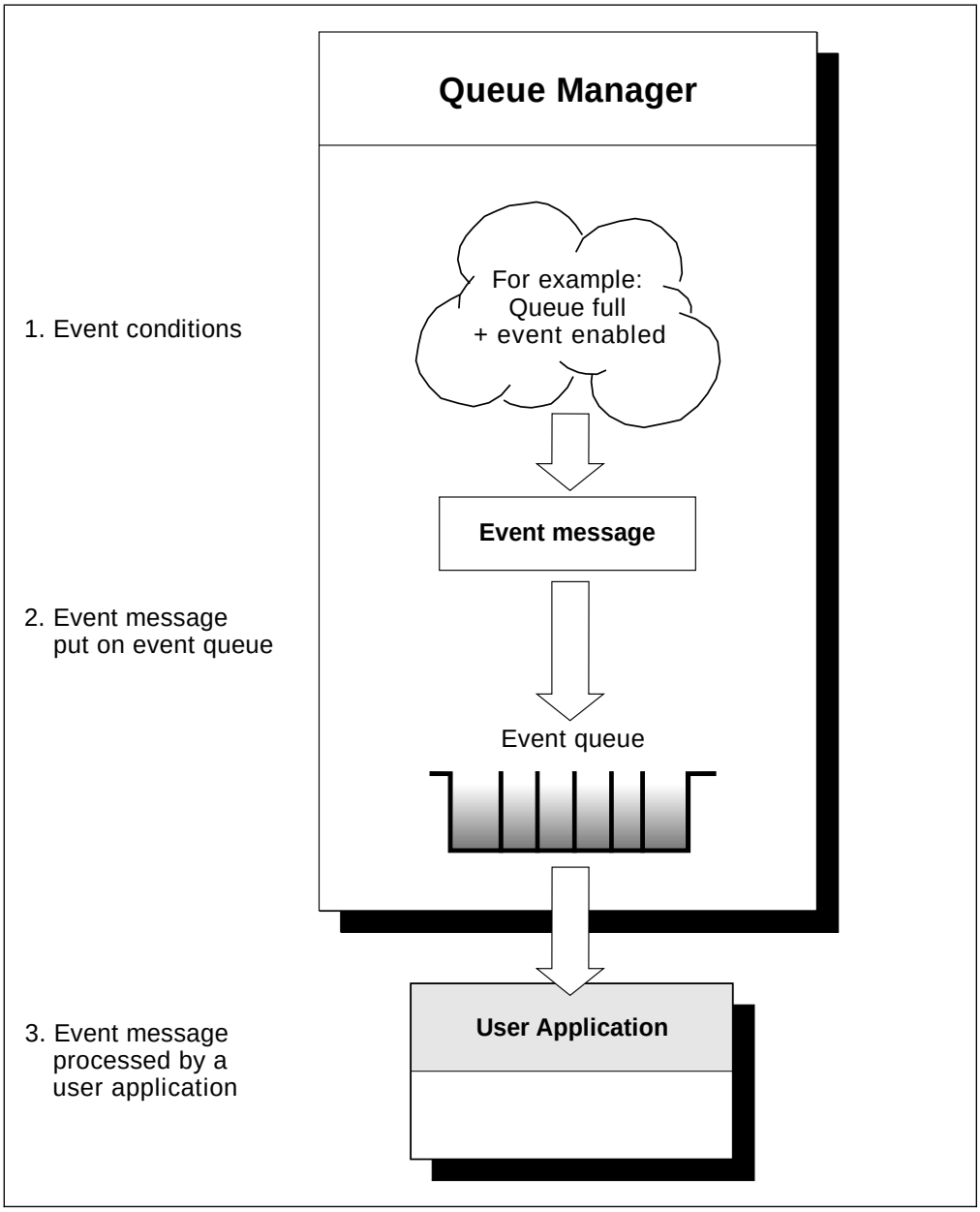


Figure 2. Understanding instrumentation events

For example, the conditions giving rise to a *Queue Full* event are:

- Queue Full events are enabled for a specified queue
  and
- An application issues an MQPUT request to put a message on that queue, but the request fails because the queue is full.

Other conditions that can give rise to instrumentation events include:

- A threshold limit for the number of messages on a queue is reached.
- A channel instance is started or stopped.
- On the MQSeries products for UNIX systems, MQSeries for Digital OpenVMS, MQSeries for Tandem NonStop Kernel, and on MQSeries for Windows NT, an application attempts to open a queue specifying a user ID that is not authorized.

For the full list of events see Table 15 on page 39.

With the exception of channel events, all instrumentation events must be enabled before they can be generated.

Types of events

MQSeries instrumentation events may be categorized as follows:

Queue manager events
These events are related to the definitions of resources within queue managers. For example, an application attempts to put a message to a queue that does not exist.

Performance events
These events are notifications that a threshold condition has been reached by a resource. For example, a queue depth limit has been reached.

Channel events
These events are reported by channels as a result of conditions detected during their operation. For example, when a channel instance is stopped.

The event type is returned in the command identifier field in the message data.

Trigger events

When we discuss triggering in other MQSeries books, we sometimes refer to a *trigger event*. This occurs when a queue manager detects that the conditions for a trigger event have been met. For example, for a queue for which triggers are active, a message of the required priority has been put on a queue so that the trigger depth is reached.

The result of a trigger event is that a trigger message is put onto an initiation queue and an application program is started. No other event messages as described in this book are involved (unless, for example, the initiation queue fills up and generates an instrumentation event).

Event notification through event queues

When an event occurs the queue manager puts an event message on the appropriate event queue, if defined. The event message contains information about the event that you can retrieve by writing a suitable MQI application program that:

- Gets the message from the queue.
- Processes the message to extract the event data. For an overview of event message formats, see “Format of event messages” on page 9. For detailed descriptions about the format of each event message, see “Event message formats” on page 35.

For each queue manager, each category of event has its own event queue. All events in that category result in an event message being put onto the same queue.

This event queue:	Contains messages from:
SYSTEM.ADMIN.QMGR.EVENT	Queue manager events
SYSTEM.ADMIN.PERFM.EVENT	Performance events
SYSTEM.ADMIN.CHANNEL.EVENT	Channel events

You can define event queues either as local queues, alias queues, or as local definitions of remote queues. If you define all your event queues as local definitions of the same remote queue on one queue manager, you can centralize your monitoring activities.

Using triggered event queues

You can set up the event queues with triggers so that when an event is generated, the event message being put onto the event queue starts a (user-written) monitoring application. This application can process the event messages and take appropriate action. For example, certain events may require that an operator be informed, other events may start off an application that performs some administration tasks automatically.

When an event queue is unavailable

If an event occurs when the event queue is not available, the event message is lost. For example, if you do not define an event queue for a category of event, all event messages for that category will be lost. The event messages are *not*, for example, saved on the dead-letter (undelivered-message) queue.

However, the event queue may be defined as a remote queue. Then, if there is a problem on the remote system putting messages to the resolved queue, the event message **will** appear on the remote system's dead-letter queue.

An event queue may be unavailable for many different reasons including:

- The queue has not been defined.
- The queue has been deleted.
- The queue is full.
- The queue has been put-inhibited.

The absence of an event queue does not prevent the event from occurring. For example, after a performance event, the queue manager changes the queue attributes and resets the queue statistics. This happens whether or not the event message is put on the performance event queue. For more information about performance events changing queue attributes, see Chapter 3, "Understanding performance events" on page 17.

Enabling and disabling events

You can enable and disable events by specifying the appropriate values for queue manager or queue attributes (or both) depending on the type of event. You do this using:

- MQSeries commands (MQSC). For more information, see the *MQSeries Command Reference*.
- PCF commands, for queue managers on OS/400, OS/2, Windows NT, OpenVMS, Tandem NSK, and UNIX systems. For more information, see Chapter 6, "Introduction to Programmable Command Formats" on page 123.
- Control Language (CL) commands for queue managers on OS/400. For more information, see the *MQSeries for AS/400 Administration Guide*
- The operations and controls panels for queue managers on MVS/ESA. For more information, see the *MQSeries for MVS/ESA System Management Guide*.

Note: Attributes related to events for both queues and queue managers can be set by command only. They are not supported by the MQI call MQSET.

Enabling and disabling an event depends on the category of the event:

- Queue Manager events are enabled by setting attributes on the queue manager. See “Enabling and disabling queue manager events” on page 11 for more information.
- Performance events as a whole must be enabled on the queue manager, otherwise no performance events can occur. Then, you enable the specific performance events by setting the appropriate queue attribute. You also have to specify the conditions that give rise to the event. For more information, see “Enabling queue service interval events” on page 20 and “Understanding queue depth events” on page 27.
- Channel events do not require enabling, they occur automatically. Similarly, channel events cannot be disabled. However, channel events can be suppressed by not defining the channel events queue, or by making it put-inhibited. Note that this could cause a queue to fill up if remote event queues point to a put-inhibited channel events queue.

Hints and tips for using events

Some things to consider about event queues:

- You must not define event queues as transmission queues because event messages have formats that are incompatible with the format of messages required for transmission queues.
- Performance events are not generated for the event queues themselves.
- If a queue manager attempts to put a queue manager or a performance event message on an event queue and an error is detected which would normally create an event, another event is not created and no action is taken.

  Notes:

  1. If a channel event is put onto an event queue, an error condition causes the queue manager to create an event as usual.
  2. Putting a message on the dead-letter queue can cause an event to be generated if the event conditions are met.

- Event queues may have trigger actions associated with them and may therefore create trigger messages. However, If these trigger messages, in turn, cause conditions that would normally generate an event, no event is in fact generated. This ensures that looping does not occur.
- MQGET calls and MQPUT calls within a unit of work can cause performance related events to occur regardless of whether the unit of work is committed or backed out.
- The putting of the event message and any subsequent actions arising do not affect the reason code to the MQI call that caused the event.

Format of event messages

Event messages contain information about the event and its origin. Typically, these messages are processed by a system management application program tailored to meet the requirements of the enterprise at which it runs. As with all MQSeries messages, an event message has two parts: a message descriptor and the message data. The message descriptor is based on the MQMD structure, which is defined in the *MQSeries Application Programming Reference*. The message data is also made up of two parts:

- An *event header* containing the reason code that identifies the event type
- The *event data*, which provides further information about the event

"Message descriptors in event messages" on page 35 describes the format of the message descriptor when used with event messages.

Monitoring events across different platforms

If you write an application using events to monitor queue managers, you need to:

1. Set up channels between the queue managers in your network.
2. Implement the required data conversions. The normal rules of data conversion apply. For example, if you are monitoring events on a UNIX system queue manager from an MVS/ESA queue manager, you must ensure that you perform the EBCDIC to ASCII conversions.

See the *MQSeries Application Programming Guide* for more information.

Chapter 2. Queue manager and channel events

This chapter provides a brief overview of both queue manager events and channel events.

Queue manager events

Queue manager events are related to the definitions of resources within queue managers. The event messages for queue manager events are put on the SYSTEM.ADMIN.QMGR.EVENT queue. The following queue manager event types are supported:

- Authority (on OS/400, Windows NT, OpenVMS, Tandem NSK, and UNIX systems only)
- Inhibit
- Local
- Remote
- Start and Stop (for MVS/ESA: Start only)

For each event type in this list, there is a queue manager attribute that enables or disables the event type. See the *MQSeries Command Reference* for more information.

The conditions that give rise to the event (when enabled) include:

- An application issues an MQI call, which fails. The reason code from the call is the same as the reason code in the event message.

  Note that a similar condition may occur during the internal operation of a queue manager, for example, when generating a report message. The reason code in an event message may be one that does match an MQI reason code even though it is not associated with any application. Therefore you should not assume that, because an event message reason code looks like an MQI reason code, the event was necessarily caused by an unsuccessful MQI call from an application.

- A command is issued to a queue manager and the processing of this command causes an event. For example:
  - A queue manager is stopped or started.
  - A command is issued where the associated user ID is not authorized for that command.

Enabling and disabling queue manager events

You enable queue manager events by specifying the appropriate attribute on the MQSeries command ALTER QMGR, or the appropriate parameters and values on the equivalent PCF command, Change Queue Manager. For example, to enable inhibit events on the default queue manager use this MQSeries command:

```
ALTER QMGR INHIBTEV (ENABLED)
```

To disable the event, set the INHIBTEV attribute to DISABLED using this MQSC:

```
ALTER QMGR INHIBTEV (DISABLED)
```

To enable the same event from a PCF command, use this combination of parameters and values:

Command	Command parameter	Parameter value
Change Queue Manager	InhibitEvent	MQEVR_ENABLED

To disable these events, you issue the same command but specify a parameter value of MQEVR_DISABLED.

Authority events

Note to users

1. All authority events are valid on Digital OpenVMS, OS/400, Windows NT, and UNIX systems only.
2. Tandem NSK supports only Not Authorized (type 1).

Authority events indicate that an authorization violation has been detected. For example, an application attempts to open a queue for which it does not have the required authority, or a command is issued from a user ID that does not have the required authority.

You enable authority events using:

- The AUTHOREV attribute on the MQSeries command ALTER QMGR
- The *AuthorityEvent* parameter on the Change Queue Manager PCF command

For more information about the event data returned in authority event messages see:

"Not Authorized (type 1)" on page 69
"Not Authorized (type 2)" on page 71
"Not Authorized (type 3)" on page 73
"Not Authorized (type 4)" on page 75

Inhibit events

Inhibit events indicate that an MQPUT or MQGET operation has been attempted against a queue, where the queue is inhibited for puts or gets respectively.

You enable inhibit events using:

- The INHIBTEV attribute on the MQSeries command ALTER QMGR
- The *InhibitEvent* parameter on the Change Queue Manager PCF command.

For more information about the event data returned in inhibit event messages, see:

"Get Inhibited" on page 67
"Put Inhibited" on page 77

Local events

Local events indicate that an application (or the queue manager) has not been able to access a local queue, or other local object. For example, when an application attempts to access an object that has not been defined.

You enable local events using:

- The LOCALEV attribute on the MQSeries command ALTER QMGR
- The *LocalEvent* parameter on the Change Queue Manager PCF command

For more information about the event data returned in local event messages, see:

"Alias Base Queue Type Error" on page 40
"Queue Type Error" on page 92
"Unknown Alias Base Queue" on page 100
"Unknown Object Name" on page 104

Remote events

Remote events indicate that an application (or the queue manager) cannot access a (remote) queue on another queue manager. For example, when the transmission queue to be used is not correctly defined.

You enable remote events using:

- The REMOTEEV attribute on the MQSeries command ALTER QMGR
- The *RemoteEvent* parameter on the Change Queue Manager PCF command

For more information about the event data returned in the remote event messages, see:

"Default Transmission Queue Type Error" on page 63
"Default Transmission Queue Usage Error" on page 65
"Queue Type Error" on page 92
"Remote Queue Name Error" on page 94
"Transmission Queue Type Error" on page 96
"Transmission Queue Usage Error" on page 98
"Unknown Default Transmission Queue" on page 102
"Unknown Remote Queue Manager" on page 106
"Unknown Transmission Queue" on page 109

Start and stop events

Start and stop events (start only for MVS/ESA) indicate that a queue manager has been started or has been requested to stop or quiesce.

You enable start and stop events using:

- The STRSTPEV attribute on the MQSeries command ALTER QMGR
- The *StartStopEvent* parameter on the Change Queue Manager PCF command

Stop events are not recorded unless the default message-persistence of the SYSTEM.ADMIN.QMGR.EVENT queue is defined as persistent.

For more information about the event data returned in the start and stop event messages, see:

"Queue Manager Active" on page 85

"Queue Manager Not Active" on page 86

Enabling queue manager events summary

The following figures summarize how to enable queue manager events:

Table 2. Enabling queue manager events using MQSeries commands

Queue manager events	
Event	***Queue manager attribute***
Authority	AUTHOREV (ENABLED)
Inhibit	INHIBTEV (ENABLED)
Local	LOCALEV (ENABLED)
Remote	REMOTEEV (ENABLED)
Start and Stop	STRSTPEV (ENABLED)

Table 3. Enabling queue manager events using PCF commands

Attribute name	**Parameter identifier**	**Value**
AuthorityEvent	MQIA_AUTHORITY_EVENT	MQEVR_ENABLED
InhibitEvent	MQIA_INHIBIT_EVENT	MQEVR_ENABLED
LocalEvent	MQIA_LOCAL_EVENT	MQEVR_ENABLED
RemoteEvent	MQIA_REMOTE_EVENT	MQEVR_ENABLED
StartStopEvent	MQIA_Q_START_STOP_EVENT	MQEVR_ENABLED

Channel events

Channel events are generated:

- By a command to start or stop a channel
- When a channel instance starts or stops
- When a channel receives a conversion error warning when getting a message
- When an attempt is made to create a channel automatically; the event is generated whether the attempt succeeds or fails.

Note: Using MQSeries for MVS/ESA with CICS, no channel events are generated.

When a command is used to start a channel an event is generated, and then another evant is generated when the channel instance starts. However, starting a channel by a listener or by a queue manager trigger message does not generate an event; in this case the only event generated is when the channel instance starts.

A successful start or stop channel command will generate at least two events. The events are generated for both queue managers that are connected by the channel, unless one of the queue managers does not support events, for example versions of MQSeries for AS/400 previous to V3R2. Channel event messages are put onto the SYSTEM.ADMIN.CHANNEL.EVENT queue, if it is available. Otherwise, they are ignored.

For more information about the event data returned in the channel event messages, see:

Enabling channel events

Most channel events are enabled automatically and cannot be enabled or disabled by command. The exceptions are the two automatic channel definition events. The generation of these events is controlled by the *ChannelAutoDefEvent* queue-manager attribute.

Refer to the *MQSeries Application Programming Reference* manual for further details of this attribute.

If a queue manager does not have a SYSTEM.ADMIN.CHANNEL.EVENT queue, or if this queue is put inhibited, all channel event messages are discarded, unless they are being put by an MCA across a link to a remote queue. In this case they are put on the dead-letter queue.

Chapter 3. Understanding performance events

This chapter describes what performance events are, how they are generated, how they can be enabled, and how they are used.

In this chapter, the examples assume that you set queue attributes by using the appropriate MQSeries commands (MQSC). See the *MQSeries Command Reference* for more information. You can also set them using:

- The operations and controls panels, for queue managers, on MVS/ESA.
- The corresponding PCF commands, for queue managers, on:
  - Digital OpenVMS
  - OS/2
  - OS/400
  - Tandem NSK
  - Windows NT
  - UNIX systems

See Chapter 6, “Introduction to Programmable Command Formats” on page 123 for more information.

What performance events are

Performance events are related to conditions that can affect the performance of applications that use a specified queue.

There are two types of performance event:

- *Queue depth events*, related to the number of messages on a queue, that is how full, or empty, the queue is.
- *Queue service interval events*, related to whether messages are processed within a user-specified time interval.

The scope of performance events is the queue, so that MQPUT calls and MQGET calls on one queue do not affect the generation of performance events on another queue.

Note: A message must be either put on, or removed from, a queue for any performance event to be generated.

Performance event data

When a performance event is generated, the queue manager puts the associated event message on the SYSTEM.ADMIN.PERFM.EVENT queue.

The event data contains a reason code that identifies the cause of the event, a set of performance event statistics, and other data. For more information about the event data returned in performance event messages, see:

“Queue Depth High” on page 79
“Queue Depth Low” on page 81
“Queue Full” on page 83
“Queue Service Interval High” on page 88
“Queue Service Interval OK” on page 90

Understanding performance event statistics

The event data in the event message contains information about the event for system management programs. For all performance events, the event data contains the names of the queue manager and the queue associated with the event. Also, the event data contains statistics related to the event. You can use these statistics to analyze the behavior of a specified queue. Table 4 summarizes the event statistics. All the statistics refer to what has happened since the last time the statistics were reset.

Table 4. Performance event statistics

Parameter	Description
TimeSinceReset	The elapsed time since the statistics were last reset.
HighQDepth	The maximum number of messages on the queue since the statistics were last reset.
MsgEnqCount	The number of messages enqueued (the number of MQPUT calls to the queue), since the statistics were last reset.
MsgDeqCount	The number of messages dequeued (the number of MQGET calls to the queue), since the statistics were last reset.

Performance event statistics are reset when:

- Any performance event occurs.
- The PCF command, Reset Queue Statistics, is issued from a user-written administration program. There is no MQSC equivalent for this command.

Understanding queue service interval events

Queue service interval events indicate whether a queue was 'serviced' within a user-defined time interval called the *service interval*. Depending on the circumstances at your installation, you can use queue service interval events to monitor whether messages are being taken off queues quickly enough.

What queue service interval events are

There are two types of queue service interval event:

- A **Queue Service Interval OK** event, which indicates that following an MQPUT call or an MQGET call that leaves a non-empty queue, an MQPUT call or an MQGET call was performed within a user-defined time period, known as the *service interval*.

  In this section, Queue Service Interval OK events are referred to as OK events.

- A **Queue Service Interval High** event, which indicates that following an MQGET call or put that leaves a non-empty queue, an MQGET call was not performed within the user-defined service interval.

  This event message can be caused by an MQPUT call or an MQGET call.

  In this section, Queue Service Interval High events are referred to as high events.

These events are mutually exclusive, that is, if one is enabled the other is disabled. However, both events can be simultaneously disabled.

Figure 3 shows a graph of queue depth against time. At P1, an application issues an MQPUT, to put a message on the queue. At G1, another application issues an MQGET to remove the message from the queue.

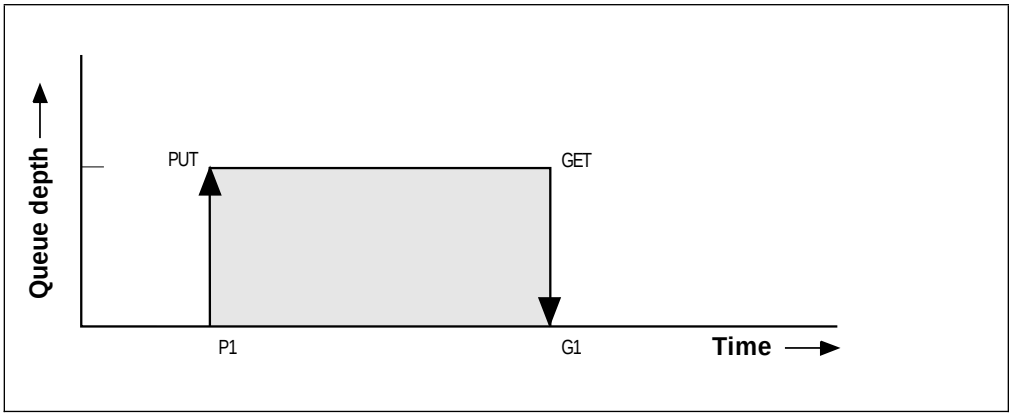


Figure 3. Understanding queue service interval events

In terms of queue service interval events, these are the possible outcomes:

- If the elapsed time between the put and get is less than or equal to the service interval:
  - If OK events are enabled, a *Queue Service Interval OK* event is generated at G1.
  - If high events are enabled, no event is generated at G1.
  - If neither event is enabled, no queue service interval event is generated.
- If the elapsed time between the put and get is greater than the service interval:
  - If high events are enabled, a *Queue Service Interval High* event is generated at G1.
  - If OK events are enabled, no event is generated at G1.
  - If neither event is enabled, no queue service interval event is generated.

The actual algorithm for starting the service timer and generating events is described in “Queue service-interval-events algorithm” on page 21.

Understanding the service timer

Queue service interval events use an internal timer, called the *service timer*, which is controlled by the queue manager. The service timer is only used if one or other of the queue service interval events are enabled.

What precisely does the service timer measure?

The service timer measures the elapsed time between an MQPUT call to an empty queue or an MQGET call and the next put or get, provided the queue depth is non-zero between these two operations.

When is the service timer active?

The service timer is always active, that is, running, if the queue has messages on it (depth is non-zero) and a queue service interval event is

enabled. If the queue becomes empty (queue depth zero), the timer is put into an OFF state, to be restarted on the next put.

When is the service timer reset?
The service timer is always reset after an MQGET call. It is also reset by an MQPUT call to an empty queue. However, it is not necessarily reset on a queue service interval event.

How is the service timer used?
Following an MQGET call or an MQPUT call, the queue manager compares the elapsed time as measured by the service timer, with the user-defined service interval. The result of this comparison is that:

- An OK event is generated if the operation is an MQGET call and the elapsed time is less than or equal to the service interval, AND this event is enabled.
- A high event is generated if the elapsed time is greater than the service interval, AND this event is enabled.

Can applications read the service timer?
No, the service timer is an internal timer that is not available to applications.

What about the *TimeSinceReset* parameter?
The *TimeSinceReset* parameter is returned as part of the event statistics in the event data. It specifies the time between successive queue service interval events, unless the event statistics are reset. The reset can be caused by a queue depth event or you can reset them yourself explicitly using the PCF command Reset Queue Statistics.

Enabling queue service interval events

To configure a queue for queue service interval events you must:

1. Enable performance events on the queue manager, using the queue manager attribute *PerformanceEvent* (PERFMEV in MQSC).
2. Set the control attribute, *QServiceIntervalEvent*, for a Queue Service Interval High or OK event on the queue, as required (QSVCIEV in MQSC).
3. Specify the service interval time by setting the *QServiceInterval* attribute for the queue to the appropriate length of time (QSVCINT in MQSC).

For example, to enable Queue Service Interval High events with a service interval time of 10 seconds (10 000 milliseconds) use the following MQSC:

```
ALTER QMGR +
PERFMEV(ENABLED)

ALTER QLOCAL('MYQUEUE') +
      QSVCINT(10000) +
      QSVCIEV(HIGH)
```

Note: When enabled, a queue service interval event can only be generated on an MQPUT call or an MQGET call. The event is **not** generated when the elapsed time becomes equal to the service interval time.

Automatic enabling of queue service interval events

The high and OK events are mutually exclusive, that is, when one is enabled, the other is automatically disabled.

When a high event is generated on a queue, the queue manager automatically disables high events and enables OK events for that queue.

Similarly, when an OK event is generated on a queue, the queue manager automatically disables OK events and enables high events for that queue.

Queue service-interval-events algorithm

This section gives the formal rules associated with the timer, and with the queue service interval events.

Service timer

The service timer is reset to zero and restarted:

- Following an MQPUT call to an empty queue.
- Following an MQGET call, if the queue is not empty after the MQGET call.

The resetting of the timer does not depend on whether an event has been generated.

At queue manager startup the service timer is set to startup time if the queue depth is greater than zero.

If the queue is empty following an MQGET call, the timer is put into an OFF state.

Queue Service Interval OK events

- The Queue Service Interval OK event must be enabled.
- If the service time (elapsed time) is less than or equal to the service interval, an event is generated on the next MQGET call.

Queue Service Interval High events

- The high event must be enabled.
- If the service time is greater than the service interval, an event is generated on the next MQPUT or MQGET call.

Queue service-interval-events examples

This section provides progressively more complex examples to illustrate the use of queue service interval events.

The figures accompanying the examples have the same structure:

- The top section is a graph of queue depth against time, showing individual MQGET calls and MQPUT calls.
- The middle section shows a comparison of the time constraints. There are three time periods that you must consider:
  - The user-defined service interval.
  - The time measured by the service timer.

  - The time since event statistics were last reset (TimeSinceReset in the event data).
- The bottom section of each figure shows which events are enabled at any instant and what events are generated.

The following examples illustrate:

- How the queue depth varies over time.
- How the elapsed time as measured by the service timer compares with the service interval.
- Which event is enabled.
- What events are generated.

Example 1 (queue service interval events)

This example shows a simple sequence of MQGET calls and MQPUT calls, where the queue depth is always one or zero.

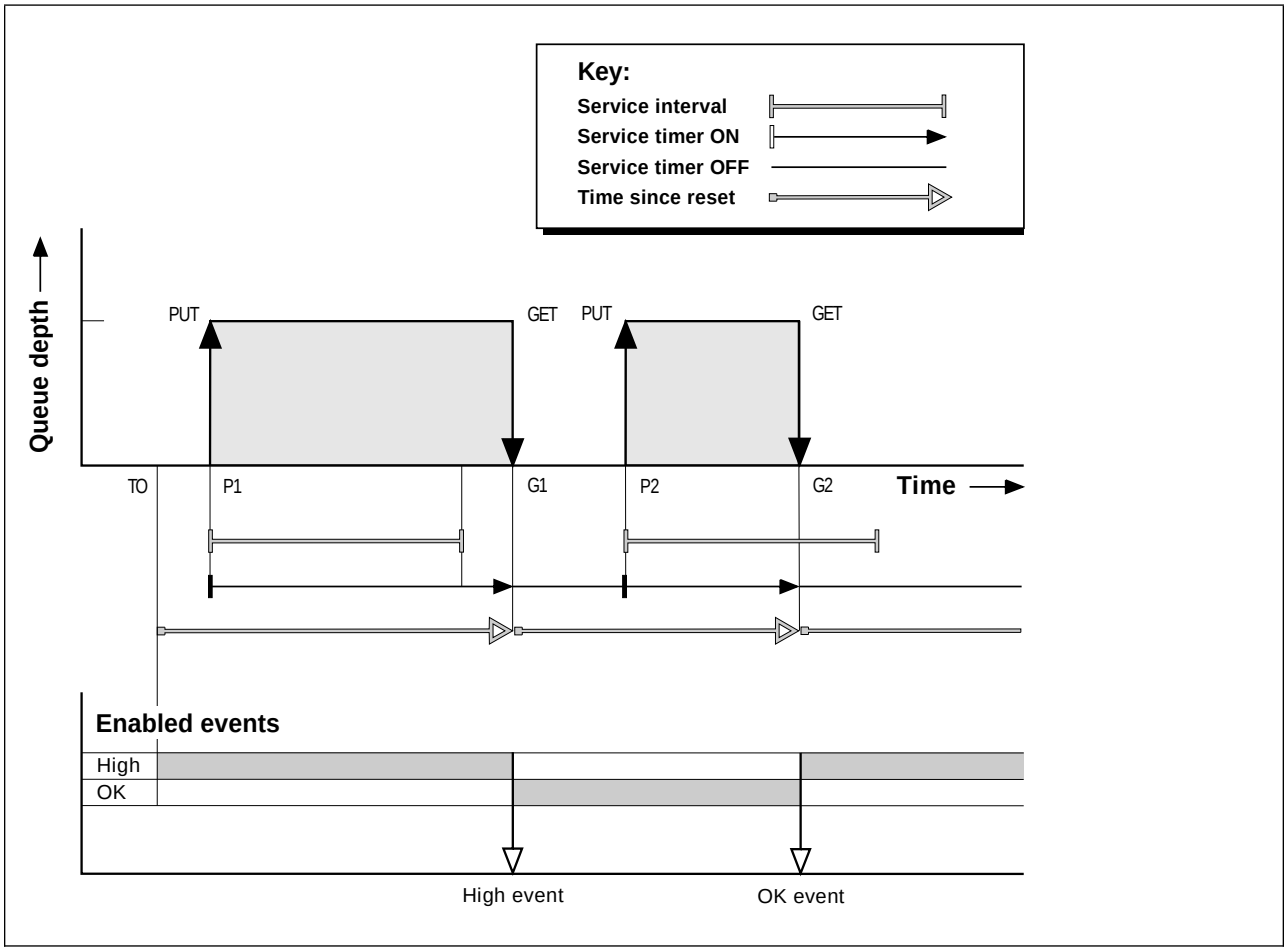


Figure 4. Queue service interval events - example 1

Commentary

1. At P1, an application puts a message onto an empty queue. This starts the service timer.

   Note that T0 may be queue manager startup time.

2. At G1, another application gets the message from the queue. Because the elapsed time between P1 and G1 is greater than the service interval, a Queue Service Interval High event is generated on the MQGET call at G1. When the high event is generated, the queue manager resets the event control attribute so that:

   a. The OK event is automatically enabled.
   b. The high event is disabled.

   Because the queue is now empty, the service timer is switched to an OFF state.

3. At P2, a second message is put onto the queue. This restarts the service timer.

4. At G2, the message is removed from the queue. However, because the elapsed time between P2 and G2 is less than the service interval, a Queue Service Interval OK event is generated on the MQGET call at G2. When the OK event is generated, the queue manager resets the control attribute so that:

   a. The high event is automatically enabled.
   b. The OK event is disabled.

   Because the queue is empty, the service timer is again switched to an OFF state.

Event statistics summary for example 1

Table 5 summarizes the event statistics for this example.

Table 5. Event statistics summary for example 1

	Event 1	**Event 2**
Time of event	T_{G1}	T_{G2}
Type of event	High	OK
TimeSinceReset	$T_{G1} - T_0$	$T_{G2} - T_{P2}$
HighQDepth	1	1
MsgEnqCount	1	1
MsgDeqCount	1	1

The middle part of Figure 4 on page 22 shows the elapsed time as measured by the service timer compared to the service interval for that queue. To see whether a queue service interval event will occur, compare the length of the horizontal line representing the service timer (with arrow) to that of the line representing the service interval. If the service timer line is longer, and the Queue Service Interval High event is enabled, a Queue Service Interval High event will occur on the next get. If the timer line is shorter, and the Queue Service Interval OK event is enabled, a Queue Service Interval OK event will occur on the next get.

Example 2 (queue service interval events)

This example illustrates a sequence of MQPUT calls and MQGET calls, where the queue depth is not always one or zero. It also shows instances of the timer being reset without events being generated, for example, at T_{P2}.

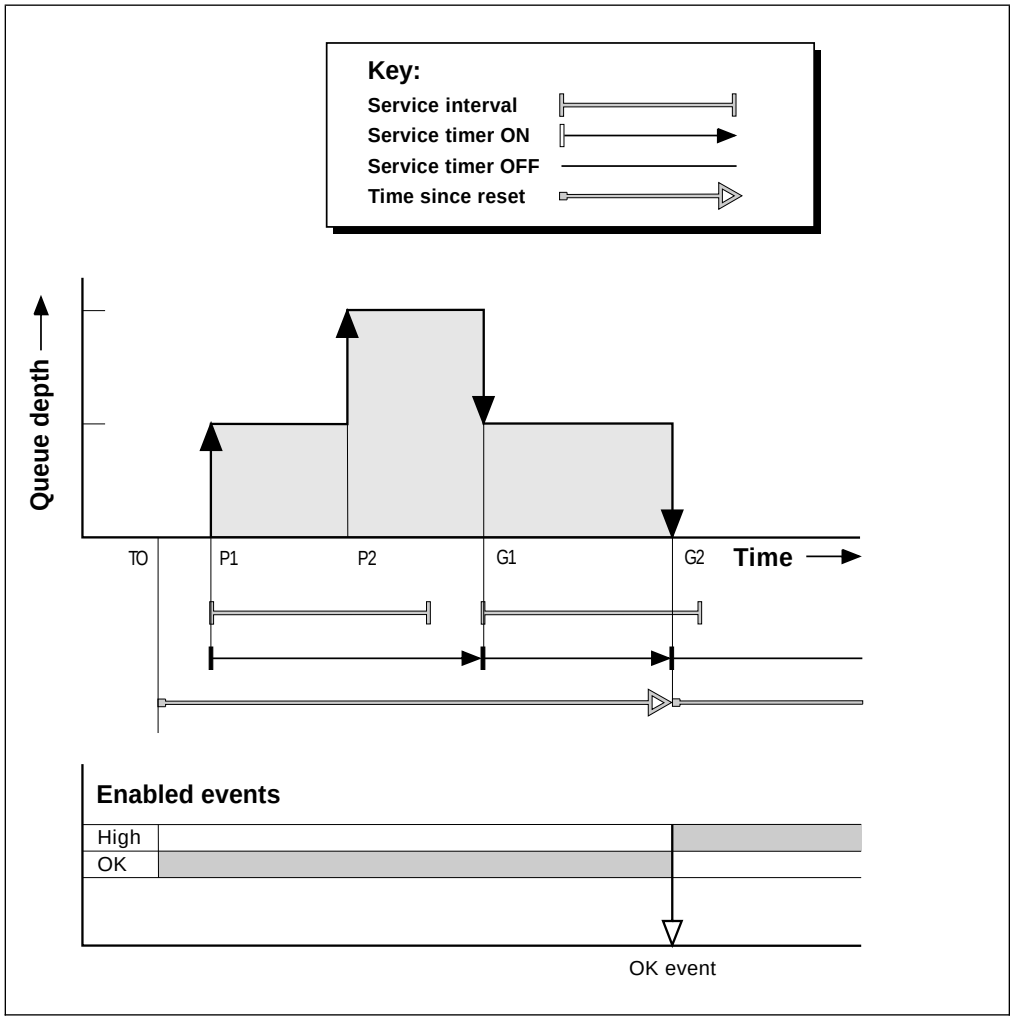


Figure 5. Queue service interval events - example 2

Commentary

In this example, **OK events are enabled** initially and queue statistics were reset at T_0.

1. At P1, the first put starts the service timer.
2. At P2, the second put does not generate an event because a put cannot cause an OK event.
3. At G1, the service interval has now been exceeded and therefore an OK event is not generated. However, the MQGET call causes the service timer to be reset.
4. At G2, the second get occurs within the service interval and this time an OK event is generated. The queue manager resets the event control attribute so that:

a. The high event is automatically enabled.
b. The OK event is disabled.

Because the queue is now empty, the service timer is switched to an OFF state.

Event statistics summary for example 2

Table 6 summarizes the event statistics for this example.

Table 6. Event statistics summary for example 2

Time of event	T_{G2}
Type of event	OK
TimeSinceReset	$T_{G2} - T_0$
HighQDepth	2
MsgEnqCount	2
MsgDeqCount	2

Example 3 (queue service interval events)

This example shows a sequence of MQGET calls and MQPUT calls that is more sporadic than the previous examples.

Commentary

1. At time T_0, the queue statistics are reset and Queue Service Interval High events are enabled.
2. At P1, the first put starts the service timer.
3. At P2, the second put increases the queue depth to two. A high event is not generated here because the service interval time has not been exceeded.
4. At P3, the third put causes a high event to be generated. (The timer has exceeded the service interval.) The timer is not reset because the queue depth was not zero before the put. However, OK events are enabled.
5. At G1, the MQGET call does not generate an event because the service interval has been exceeded and OK events are enabled. The MQGET call does, however, reset the service timer.
6. At G2, the MQGET call does not generate an event because the service interval has been exceeded and OK events are enabled. Again, the MQGET call resets the service timer.
7. At G3, the third get empties the queue and the service timer is **equal** to the service interval. Therefore an OK event is generated. The service timer is reset and high events are enabled. The MQGET call empties the queue, and this puts the timer in the OFF state.

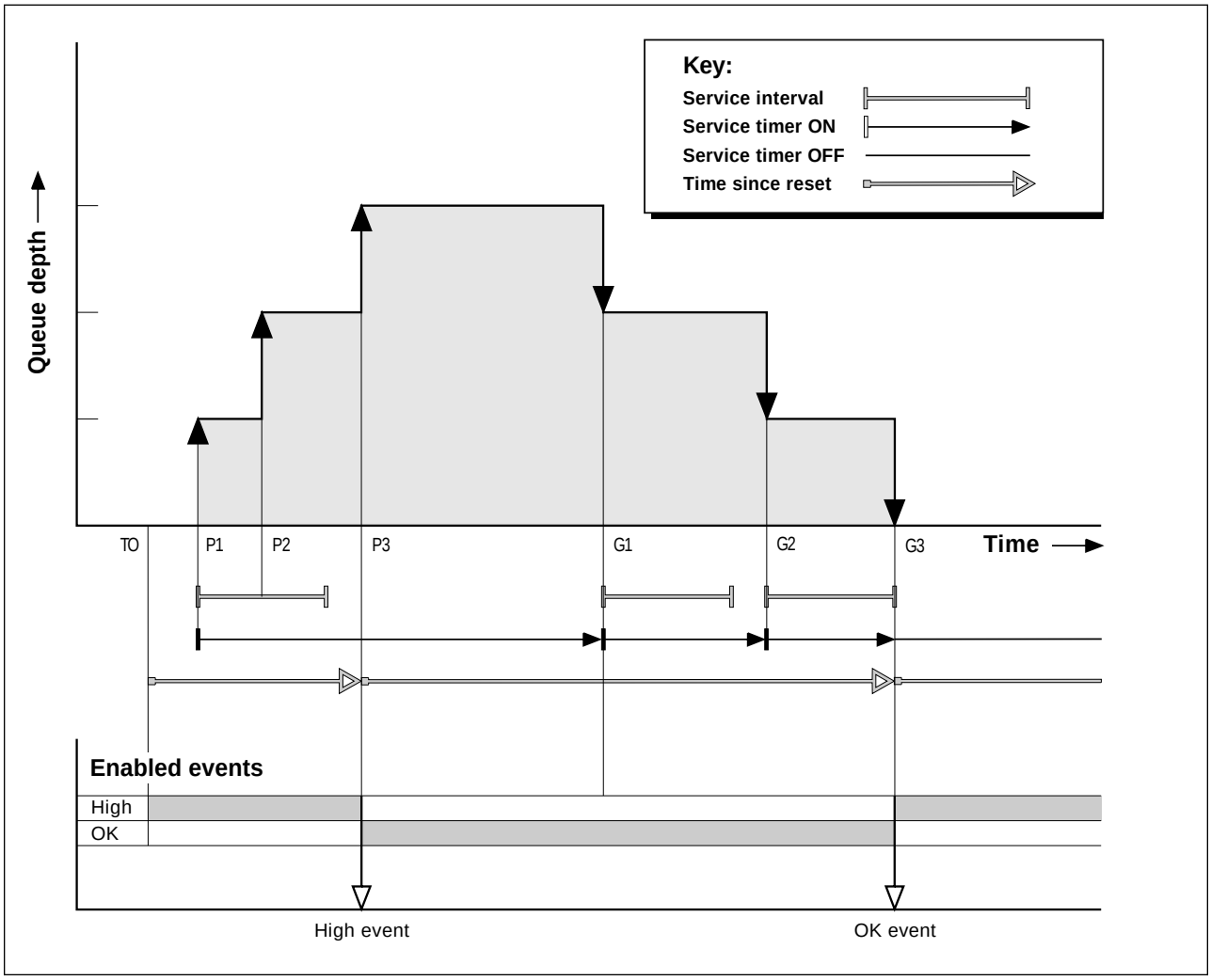


Figure 6. Queue service interval events - example 3

Event statistics summary for example 3

The following table summarizes the statistics returned in the event message data, for each event in this example.

Table 7. Event statistics summary for example 3

	Event 1	**Event 2**
Time of event	T_{P3}	T_{G3}
Type of event	High	OK
TimeSinceReset	$T_{P3} - T_0$	$T_{G3} - T_{P3}$
HighQDepth	3	3
MsgEnqCount	3	0
MsgDeqCount	0	3

What queue service interval events tell you

You must exercise some caution when you look at queue statistics. Figure 4 on page 22 shows a simple case where the messages are intermittent and each message is removed from the queue before the next one arrives. From the event data, you know that the maximum number of messages on the queue was one. You can, therefore, work out how long each message was on the queue.

However, in the general case, where there is more than one message on the queue and the sequence of MQGET calls and MQPUT calls is not predictable, you cannot use queue service interval events to calculate how long an individual message remains on a queue. The TimeSinceReset parameter, which is returned in the event data, can include a proportion of time when there are no messages on the queue. Therefore any results you derive from these statistics are implicitly averaged to include these times.

Understanding queue depth events

In MQSeries applications it is most important that queues do not become full. If they do, applications can no longer put messages on the queue that they specify. Although the message is not lost if this occurs, it can be a considerable inconvenience. The number of messages can build up on a queue if the messages are being put onto the queue faster than the applications that process them can take them off.

The solution to this problem depends on the particular circumstances, but may involve:

- Diverting some messages to another queue.
- Starting new applications to take more messages off the queue.
- Stopping non-essential message traffic.
- Increasing the queue depth to overcome a transient maximum.

Clearly, having advanced warning that problems may be on their way makes it easier to take preventive action. For this purpose, queue depth events are provided.

What queue depth events are

Queue depth events are related to the queue depth, that is, the number of messages on the queue. The types of queue depth events are:

- **Queue Depth High events**, which indicate that the queue depth has increased to a predefined threshold called the Queue Depth High limit.
- **Queue Depth Low events**, which indicate that the queue depth has decreased to a predefined threshold called the Queue Depth Low limit.
- **Queue Full events**, which indicate that the queue has reached its maximum depth, that is, the queue is full.

Queue Depth High events give advance warning that a queue is filling up. This means that having received this event, the system administrator should take some preventive action. If this action is successful and the queue depth drops to a ‘safe’ level, the queue manager can be configured to generate a Queue Depth Low event indicating an ‘all clear’ state.

Figure 8 on page 30 shows a graph of queue depth against time in such a case. The preventive action was (presumably) taken between T_2 and T_3 and continues to have effect until T_4 when the queue depth is well inside the 'safe' zone.

Enabling queue depth events

By default, all queue depth events are disabled. To configure a queue for any of the queue depth events you must:

1. Enable performance events on the queue manager, using the queue manager attribute *PerformanceEvent* (PERFMEV in MQSC).
2. Enable the event on the required queue by setting the following as required:
   - *QDepthHighEvent*(QDPHIEV in MQSC)
   - *QDepthLowEvent*(QDPLOEV in MQSC)
   - *QDepthMaxEvent*(QDPMAXEV in MQSC)
3. Set the limits, if required, to the appropriate levels, expressed as a percentage of the maximum queue depth, by setting either:
   - *QDepthHighLimit*(QDEPTHHI in MQSC), and
   - *QDepthLowLimit*(QDEPTHLO in MQSC).

Enabling Queue Depth High events

When enabled, a Queue Depth High event is generated when a message is put on the queue causing the queue depth to be greater than or equal to the value determined by the Queue Depth High limit.

To enable Queue Depth High events on the queue MYQUEUE with a limit set at 80%, use the following MQSC:

```
ALTER QMGR PERFMEV(ENABLED)
ALTER QLOCAL('MYQUEUE') QDEPTHHI(80) QDPHIEV(ENABLED)
```

Automatically enabling Queue Depth High events: A Queue Depth High event is automatically enabled by a Queue Depth Low event on the same queue.

A Queue Depth High event automatically enables both a Queue Depth Low and a Queue Full event on the same queue.

Enabling Queue Depth Low events

When enabled, a Queue Depth Low event is generated when a message is removed from a queue by an MQGET call operation causing the queue depth to be less than or equal to the value determined by the Queue Depth Low limit.

To enable Queue Depth Low events on the queue MYQUEUE with a limit set at 20%, use the following MQSC:

```
ALTER QMGR PERFMEV(ENABLED)
ALTER QLOCAL('MYQUEUE') QDEPTHLO(20) QDPLOEV(ENABLED)
```

Automatically enabling Queue Depth Low events: A Queue Depth Low event is automatically enabled by a Queue Depth High event or a Queue Full event on the same queue.

A Queue Depth Low event automatically enables both a Queue Depth High and a Queue Full event on the same queue.

Enabling Queue Full events

When enabled, a Queue Full event is generated when an application is unable to put a message onto a queue because the queue is full.

To enable Queue Full events on the queue MYQUEUE, use the following MQSC:

```
ALTER QMGR PERFMEV(ENABLED)
ALTER QLOCAL('MYQUEUE') QDPMAXEV(ENABLED)
```

Automatically enabling Queue Full events: A Queue Full event is automatically enabled by a Queue Depth High or a Queue Depth Low event on the same queue.

A Queue Full event automatically enables a Queue Depth Low event on the same queue.

Queue depth events examples

This section contains some examples of queue depth events. The following examples illustrate how queue depth varies over time.

Example 1 (queue depth events)

The queue, MYQUEUE1, has a maximum depth of 1000 messages, and the high and low queue depth limits are 80% and 20% respectively. Initially, Queue Depth High events are enabled, while the other queue depth events are disabled.

The MQSeries commands (MQSC) to configure this queue are:

```
ALTER QMGR PERFMEV(ENABLED)

DEFINE QLOCAL('MYQUEUE1') +
  MAXDEPTH(1000) +
  QDPMAXEV(DISABLED) +
  QDEPTHHI(80) +
  QDPHIEV(ENABLED) +
  QDEPTHLO(20) +
  QDPLOEV(DISABLED)
```

Figure 7. Definition of MYQUEUE1

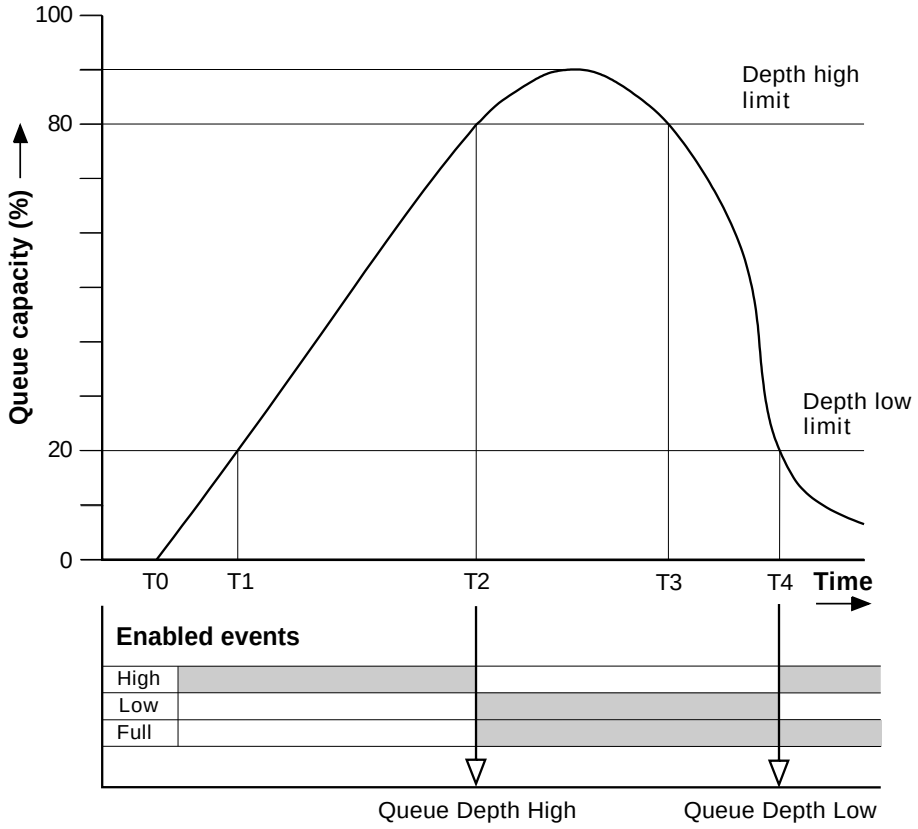


Figure 8. Queue depth events (1)

Commentary

Figure 8 shows how the queue depth changes over time:

1. At T_1, the queue depth is increasing (more MQPUT calls than MQGET calls) and crosses the Queue Depth Low limit. No event is generated at this time.
2. The queue depth continues to increase until T_2, when the depth high limit (80%) is reached and a Queue Depth High event is generated.

   This enables both Queue Full and Queue Depth Low events.
3. The (presumed) preventive actions instigated by the event prevent the queue from becoming full. By time T_3, the Queue Depth High limit has been reached again, this time from above. No event is generated at this time.
4. The queue depth continues to fall until T_4, when it reaches the depth low limit (20%) and a Queue Depth Low event is generated.

   This enables both Queue Full and Queue Depth High events.

Table 8 on page 31 summarizes the queue event statistics and Table 9 on page 31 summarizes which events are enabled at different times for this example.

Table 8. Event statistics summary for queue depth events (example 1)		
	Event 2	**Event 4**
Time of event	T_2	T_4
Type of event	Queue Depth High	Queue Depth Low
TimeSinceReset	$T_2 - T_0$	$T_4 - T_2$
HighQDepth (Maximum queue depth since reset)	800	900
MsgEnqCount	1157	1220
MsgDeqCount	357	1820

Table 9. Summary showing which events are enabled			
Time period	**Queue Depth High event**	**Queue Depth Low event**	**Queue Full event**
Before T_1	ENABLED	-	-
T_1 to T_2	ENABLED	-	-
T_2 to T_3	-	ENABLED	ENABLED
T_3 to T_4	-	ENABLED	ENABLED
After T_4	ENABLED	-	ENABLED

Example 2 (queue depth events)

This is a more extensive example, however, the principles remain the same. This example assumes the use of the same queue MYQUEUE1 as defined in Figure 7 on page 29.

Table 10 on page 33 summarizes the queue event statistics and Table 11 on page 33 summarizes which events are enabled at different times for this example.

Figure 9 on page 32 shows the variation of queue depth over time.

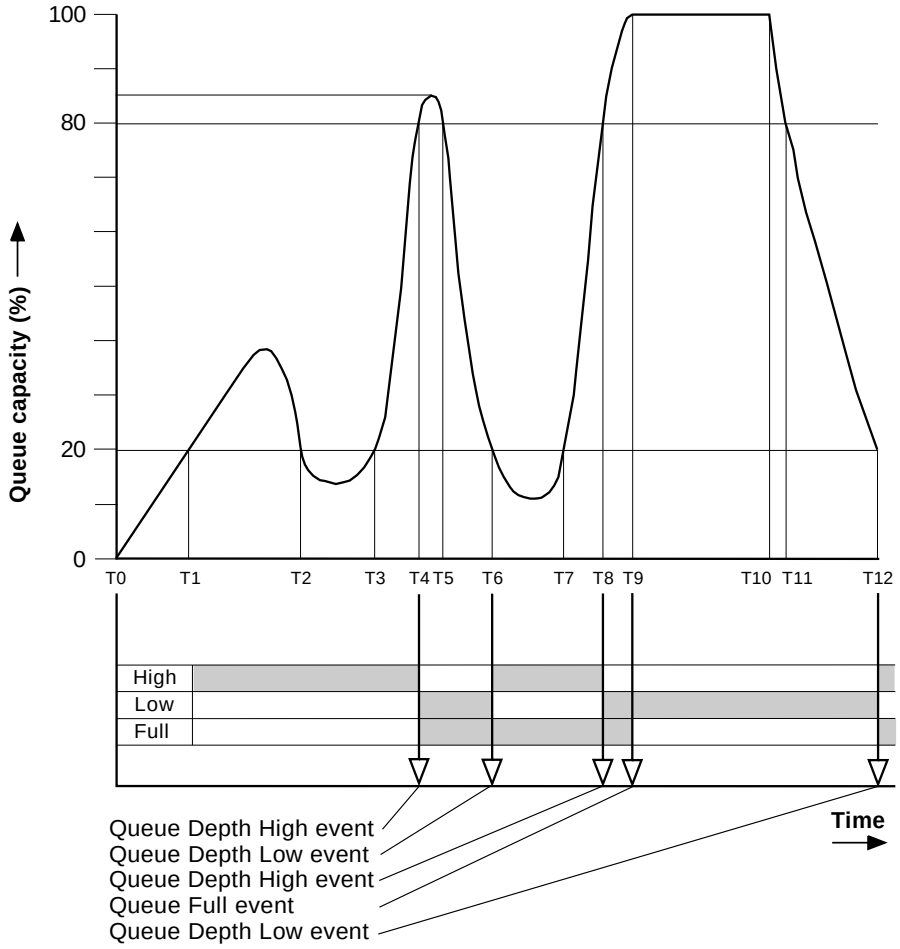


Figure 9. Queue depth events(2)

Commentary

Some points to note are:

1. No Queue Depth Low event is generated at:

   T_1 (Queue depth increasing, and not enabled)
   T_2 (Not enabled)
   T_3 (Queue depth increasing, and not enabled)

2. At T_4 a Queue Depth High event occurs. This enables both Queue Full and Queue Depth Low events.

3. At T_9 a Queue Full event occurs **after** the first message that cannot be put on the queue because the queue is full.

4. At T_{12} a Queue Depth Low event occurs.

Event statistics summary (example 2)

Table 10. Event statistics summary for queue depth events (example 2)

	Event 4	**Event 6**	**Event 8**	**Event 9**	**Event 12**
Time of event	T_4	T_6	T_8	T_9	T_{12}
Type of event	Queue Depth High	Queue Depth Low	Queue Depth High	Queue Full	Queue Depth Low
TimeSinceReset	$T_4 - T_0$	$T_6 - T_4$	$T_8 - T_6$	$T_9 - T_8$	$T_{12} - T_9$
HighQDepth	800	855	800	1000	1000
MsgEnqCount	1645	311	1377	324	221
MsgDeqCount	845	911	777	124	1021

Table 11. Summary showing which events are enabled

Time period	**Queue Depth High event**	**Queue Depth Low event**	**Queue Full event**
T_0 to T_4	ENABLED	-	-
T_4 to T_6	-	ENABLED	ENABLED
T_6 to T_8	ENABLED	-	ENABLED
T_8 to T_9	-	ENABLED	ENABLED
T_9 to T_{12}	-	ENABLED	-
After T_{12}	ENABLED	-	ENABLED

Note: Events are out of syncpoint, therefore you could have an empty queue, then fill it up causing an event, then roll back all of the messages under the control of a syncpoint manager. However, event enabling has been automatically set, so that the next time the queue fills up, no event is generated.

Enabling performance events summary

Table 12. Enabling performance events using MQSC

Queue depth event	***Queue attributes***
Queue depth high	QDPHIEV (ENABLED) QDEPTHHI (*hh*)
Queue depth low	QDPLOEV (ENABLED) QDEPTHLO (*ll*)
Queue full	QDPMAXEV (ENABLED)
Queue service interval event	***Queue attributes***
Queue Service Interval High Queue Service Interval OK No queue service interval events	QSVCIEV (HIGH) QSVCIEV (OK) QSVCIEV (NONE)
Service interval	QSVCINT (*tt*)

Notes:

All performance events must be enabled using the queue manager attribute PERFMEV.

Numeric values

hh Queue depth high limit.
ll Queue depth low limit.
(Both values are expressed as a percentage of the maximum queue depth, which is specified by the queue attribute MAXDEPTH.)

tt Service interval time in milliseconds.

Table 13. Enabling performance events using PCF commands

Attribute	**Parameter**	**Value**
QDepthHighEvent	MQIA_Q_DEPTH_HIGH_EVENT	MQEVR_ENABLED
QDepthHighLimit	MQIA_Q_DEPTH_HIGH_LIMIT	*hh*
QDepthLowEvent	MQIA_Q_DEPTH_LOW_EVENT	MQEVR_ENABLED
QDepthLowLimit	MQIA_Q_DEPTH_LOW_LIMIT	*ll*
QDepthMaxEvent	MQIA_Q_DEPTH_MAX_EVENT	MQEVR_ENABLED
QServiceIntervalEvent	MQIA_Q_SERVICE_INTERVAL_EVENT	MQQSIE_HIGH MQQSIE_OK MQQSIE_NONE
QServiceInterval	MQIA_Q_SERVICE_INTERVAL	*tt*

Notes:

All performance events must be enabled using the queue manager attribute *PerformanceEvent*.

Numeric values

hh Queue depth high limit.
ll Queue depth low limit.
(Both values are expressed as a percentage of the maximum queue depth, which is specified by the queue attribute *MaxQDepth*)

tt Service interval time in milliseconds.

Chapter 4. Event message reference

This chapter describes the information returned in the event message for each instrumentation event.

It provides an overview of the event message format and descriptions of the parameters returned in the event messages for each event.

Event message formats

Event messages are standard MQSeries messages containing a message descriptor and message data.

Table 14 on page 36 shows the basic structure of these messages, and the names of the fields in an event message for queue service interval events.

In general, you need only a subset of this information for any system management programs that you write. For example, your application might need the following data:

- The name of the application causing the event
- The name of the queue manager on which the event occurred
- The queue on which the event was generated
- The event statistics

Message descriptors in event messages

The format of the message descriptor is defined by the MQSeries MQMD data structure, which is found in all MQSeries messages and is described in the *MQSeries Application Programming Reference*. The message descriptor contains information that can be used by a user-written system monitoring application. For example:

- The message type
- The format type
- The date and time that the message was put on the event queue

In particular, the information in the descriptor informs a system management application that the message type is MQMT_DATAGRAM, and the message format is MQFMT_EVENT.

In an event message, many of these fields contain fixed data, which is supplied by the queue manager that generated the message. The fields that make up the MQMD structure are described in “MQMD (Message descriptor)” on page 37, and also “Message descriptor for a PCF command” on page 127. It also specifies the name of the queue manager (truncated to 28 characters) that put the message, and the date and time that the event message was put on the event queue.

Table 14. Event message structure for queue service interval events

Message descriptor	**Message data**	
MQMD structure [1]	**Event header** **MQCFH structure** [2]	**Event data** [3]
Structure identifier Structure version Report options Message type Expiration time Feedback code Encoding Coded character set ID Message format Message priority Persistence Message identifier Correlation identifier Backout count Reply-to queue Reply-to queue manager User identifier Accounting token Application identity data Application type Application name Put date Put time Application origin data Group identifier Message sequence number Offset Message flags Original length	Structure type Structure length Structure version number Command identifier (event type) Message sequence number Control options Completion code Reason code (MQRC_*) Parameter count	Queue manager name Queue name Time since last reset Maximum number of messages on the queue Number of messages put on the queue Number of messages taken off the queue

Notes:

1. MQMD is the standard structure for MQSeries message headers.
2. MQCFH is the standard structure for an event header. This is the same as the PCF header structure.
3. The parameters shown are those returned for a queue service interval event. The actual event data depends on the specific event.

Message data in event messages

The event message data is based on the programmable command format (PCF) that is used in PCF command inquiries and responses. If you do not know about PCF commands, see Chapter 6, “Introduction to Programmable Command Formats” on page 123 for information.

The event message consists of two parts: the event header and the event message data, seeTable 14 . In an event message is another data structure, MQCFH, which is described in “MQCFH (PCF header)” on page 38 and “MQCFH – PCF header” on page 334.

Event header

The information in MQCFH specifies that:

- The message is an event message.
- The category of event, that is, whether the event is a queue manager, performance, or channel event.
- A reason code specifying the cause of the event. For events caused by MQI calls, this reason code is the same as the reason code for the MQI call.

Reason codes have names that begin with the characters MQRC_. For example, the reason code MQRC_PUT_INHIBITED is generated when an application attempts to put a message on a queue that is not enabled for puts.

Event message data

The event message data contains information specific to the event. This includes the name of the queue manager and, where appropriate, the name of the queue.

The data structures returned depend on which particular event was generated. In addition, for some events, certain of the structures are optional, and are returned only if they contain information that is relevant to the circumstances giving rise to the event. The values in the data structures depend on the circumstances that caused the event to be generated.

Note: The event structures in the event data are not returned in a defined order. They must be identified from the parameter identifiers shown in the description.

MQMD (Message descriptor)

The MQMD structure describes the information that accompanies the message data of an event message. In this list, the strings in parentheses next to the parameter name are the data types of each parameter. These are described in the *MQSeries Application Programming Reference*.

For an event, the MQMD structure contains these values:

Parameter	Value
StrucId (MQCHAR4)	MQMD_STRUC_ID
Version (MQLONG)	MQMD_VERSION_1 or MQMD_VERSION_2
Report (MQLONG)	MQRO_NONE
MsgType (MQLONG)	MQMT_DATAGRAM
Expiry (MQLONG)	MQEI_UNLIMITED
Feedback (MQLONG)	MQFB_NONE
Encoding (MQLONG)	Encoding of the queue manager generating the event.
CodedCharSetId (MQLONG)	Coded character set ID (CCSID) of the queue manager generating the event.
Format (MQCHAR8)	MQFMT_EVENT
Priority (MQLONG)	Default priority of the event queue, if it is a local queue, or its local definition at the queue manager generating the event.
Persistence (MQLONG)	Default persistence of the event queue, if it is a local queue, or its local definition at the queue manager generating the event.
MsgId (MQBYTE24)	The value is uniquely generated by the queue manager.
CorrelId (MQBYTE24)	MQCI_NONE
BackoutCount (MQLONG)	Always 0.
ReplyToQ (MQCHAR48)	Always blank.
ReplyToQMgr (MQCHAR48)	The queue manager name at the originating system.

UserIdentifier (MQCHAR12)	Always blank.
AccountingToken (MQBYTE32)	MQACT_NONE
ApplIdentityData (MQCHAR32)	Always blank.
PutApplType (MQLONG)	Type of application that put the message: MQAT_QMGR for a local event queue.
PutApplName (MQCHAR28)	Name of application that put the message.
PutDate (MQCHAR8)	Date when message was put, generated by the queue manager.
PutTime (MQCHAR8)	Time when message was put, generated by the queue manager.
ApplOriginData (MQCHAR4)	Always blank.

If *Version* is MQMD_VERSION_2, the following additional fields are presen:

Parameter	Value
GroupId (MQBYTE24)	MQGI_NONE
MsgSeqNumber (MQLONG)	Always 1.
Offset (MQLONG)	Always 0.
MsgFlags (MQLONG)	MQMF_NONE
OriginalLength (MQLONG)	MQOL_UNDEFINED

MQCFH (PCF header)

The MQCFH structure is the event header, which has the same format as all PCF headers. In this list, the strings in parentheses next to the parameter name are the structure types of each parameter. These are described in the *MQSeries Application Programming Reference*.

For an event, the MQCFH structure contains these values:

Parameter	Value
Type (MQLONG)	MQCFT_EVENT
StrucLength (MQLONG)	MQCFH_STRUC_LENGTH Length of command format header structure.
Version (MQLONG)	MQCFH_VERSION_1
Command (MQLONG)	Command identifier, identifies the category of event as one of: MQCMD_Q_MGR_EVENT (Queue manager event) MQCMD_PERFM_EVENT (Performance event) MQCMD_CHANNEL_EVENT (Channel event)
MsgSeqNumber (MQLONG)	Always 1.
Control (MQLONG)	MQCFC_LAST Last message in the group.
CompCode (MQLONG)	Completion code, one of: MQCC_OK (Event reporting OK condition) MQCC_WARNING (Event reporting warning condition) all events have this completion code, unless otherwise specified.
Reason (MQLONG)	Reason code identifying event. Depends on the event being reported. **Note:** Events with the same reason code are further identified by the *ReasonQualifier* parameter in the event data.
ParameterCount (MQLONG)	The number of parameter structures that follow the MQCFH structure.

Event message data

Notes to users

1. The events described in the reference section are available on all platforms, unless specific limitations are shown at the start of an event.
2. In the event message reference that follows, the strings in parentheses next to the parameter name are the structure types of each parameter. These are described in Chapter 9, “Structures used for commands and responses” on page 333.
3. Version 2.0 of MQSeries for Windows does not generate MQSeries events.

Use the following table to locate information about a particular event message:

Table 15. Event message data summary

Event type	*Event name*	*page*
Authority events	Not Authorized (type 1)	69
	Not Authorized (type 2)	71
	Not Authorized (type 3)	73
	Not Authorized (type 4)	75
Channel events	Channel Activated	46
	Channel Auto-Definition Error	48
	Channel Auto-Definition OK	50
	Channel Conversion Error	52
	Channel Not Activated	55
	Channel Started	57
	Channel Stopped	59
IMS Bridge events	Bridge Started	42
	Bridge Stopped	44
Inhibit events	Get Inhibited	67
	Put Inhibited	77
Local events	Alias Base Queue Type Error	40
	Unknown Alias Base Queue	100
	Unknown Object Name	104
Performance events	Queue Depth High	79
	Queue Depth Low	81
	Queue Full	83
	Queue Service Interval High	88
	Queue Service Interval OK	90
Remote events	Default Transmission Queue Type Error	63
	Default Transmission Queue Usage Error	65
	Queue Type Error	92
	Remote Queue Name Error	94
	Transmission Queue Type Error	96
	Transmission Queue Usage Error	98
	Unknown Default Transmission Queue	102
	Unknown Remote Queue Manager	106
	Unknown Transmission Queue	109
Start and stop events	Queue Manager Active	85
	Queue Manager Not Active	86

Alias Base Queue Type Error

Details of the condition generating the event are given, in the following text, in the *Reason* parameter of the event header.

Event message

When an event is generated, an event message is put on the SYSTEM.ADMIN.QMGR.EVENT queue.

The event message consists of the:

- Event header, containing a reason code parameter with a value of MQRC_ALIAS_BASE_Q_TYPE_ERROR
- Event data

Event data summary

Always returned:

QMgrName, *QName*, *BaseQName*, *QType*, *ApplType*, *ApplName*

Returned optionally:

ObjectQMgrName

Event header

Reason (MQLONG)

Reason code identifying the event.

The value is:

MQRC_ALIAS_BASE_Q_TYPE_ERROR
(2001, X'7D1') Alias base queue not a valid type.

An MQOPEN or MQPUT1 call was issued specifying an alias queue as the destination, but the *BaseQName* in the alias queue definition resolves to a queue that is not a local queue, or local definition of a remote queue.

Event data

QMgrName (MQCFST)

Name of the queue manager generating the event (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

QName (MQCFST)

Queue name from object descriptor (MQOD) (parameter identifier: MQCA_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

BaseQName (MQCFST)

Queue name to which the alias resolves (parameter identifier: MQCA_BASE_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

QType (MQCFIN)
Type of queue to which the alias resolves (parameter identifier: MQIA_Q_TYPE).

The value can be:

MQQT_ALIAS
Alias queue definition.

MQQT_MODEL
Model queue definition.

ApplType (MQCFIN)
Type of the application making the call that caused the event (parameter identifier: MQIA_APPL_TYPE).

ApplName (MQCFST)
Name of the application making the call that caused the event (parameter identifier: MQCACF_APPL_NAME).

The maximum length of the string is MQ_APPL_NAME_LENGTH.

Note: If the application is a server for clients, the *ApplType* and *ApplName* parameters identify the server, rather than the client.

ObjectQMgrName (MQCFST)
Name of the object queue manager (parameter identifier: MQCACF_OBJECT_Q_MGR_NAME).

This parameter is returned if the *ObjectName* in the object descriptor (MQOD) (when the object was opened) is not the queue manager currently connected.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

Bridge Started

This event is produced only by MQSeries for MVS/ESA

Details of the condition generating the event are given, in the following text, in the *Reason* parameter of the event header.

Event message

When an event is generated, an event message is put on the SYSTEM.ADMIN.CHANNEL.EVENT queue.

The event message consists of the:

- Event header, containing a reason code parameter with a value of MQRC_BRIDGE_STARTED
- Event data

Event data summary

Always returned:

QMgrName, *BridgeType*, *BridgeName*

Returned optionally:

None

Event header

Reason (MQLONG)

Name of the reason code.

The value is:

MQRC_BRIDGE_STARTED
(2125, X'84D') Bridge started.

The IMS bridge has been started.

Event data

QMgrName (MQCFST)

Name of the queue manager generating the event (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

BridgeType (MQCFIN)

Bridge type (parameter identifier: MQIACF_BRIDGE_TYPE).

The value is:

MQBT_OTMA
OTMA bridge.

BridgeName (MQCFST)

Bridge name (parameter identifier: MQCACF_BRIDGE_NAME).

For bridges of type MQBT_OTMA, the name is of the form `XCFgroupXCFmember`, where `XCFgroup` is the XCF group name to which both IMS and MQSeries belong. `XCFmember` is the XCF member name of the IMS system. The maximum length of the string is MQ_BRIDGE_NAME_LENGTH.

Bridge Stopped

This event is produced only by MQSeries for MVS/ESA

Details of the condition generating the event are given, in the following text, in the *Reason* parameter of the event header.

Event message

When an event is generated, an event message is put on the SYSTEM.ADMIN.CHANNEL.EVENT queue.

The event message consists of the:

- Event header, containing a reason code parameter with a value of MQRC_BRIDGE_STOPPED
- Event data

Event data summary

Always returned:

QMgrName, *ReasonQualifier*, *BridgeType*, *BridgeName*

Returned optionally:

ErrorIdentifier,

Event header

Reason (MQLONG)

Name of the reason code.

The value is:

MQRC_BRIDGE_STOPPED

(2126, X'84E') Bridge stopped.

The IMS bridge has been stopped.

Event data

QMgrName (MQCFST)

The name of the queue manager generating the event (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

ReasonQualifier (MQCFIN)

Identifier that qualifies the reason code (parameter identifier: MQIACF_REASON_QUALIFIER).

The value is one of the following:

MQRQ_BRIDGE_STOPPED_OK

Bridge has been stopped with either a zero return code or a warning return code.

For MQBT_OTMA bridges, one side or the other issued a normal IXCLEAVE request.

MQRQ_BRIDGE_STOPPED_ERROR
Bridge has been stopped but there is an error reported.

BridgeType (MQCFIN)
Bridge type (parameter identifier: MQIACF_BRIDGE_TYPE).

The value is:

MQBT_OTMA
OTMA bridge.

BridgeName (MQCFST)
Bridge name (parameter identifier: MQCACF_BRIDGE_NAME).

For bridges of type MQBT_OTMA, the name is of the form `XCFgroupXCFmember`, where `XCFgroup` is the XCF group name to which both IMS and MQSeries belong. `XCFmember` is the XCF member name of the IMS system. The maximum length of the string is MQ_BRIDGE_NAME_LENGTH.

ErrorIdentifier (MQCFIN)
Identifier of the cause of the error (parameter identifier: MQIACF_ERROR_IDENTIFIER).

When a bridge is stopped due to an error, this is the code that identifies the error. If the event message is because of a bridge stop failure, the following fields are set:

- The IMS sense code.

Channel Activated

> *This event is not produced if you are using CICS for distributed queue management in MQSeries for MVS/ESA.*

Details of the condition generating the event are given, in the following text, in the *Reason* parameter of the event header.

Event message

When an event is generated, an event message is put on the SYSTEM.ADMIN.CHANNEL.EVENT queue.

The event message consists of the:

- Event header, containing a reason code parameter with a value of MQRC_CHANNEL_ACTIVATED
- Event data

Event data summary

Always returned:

QMgrName, *ChannelName*,

Returned optionally:

XmitQName, *ConnectionName*

Event header

Reason (MQLONG)

Name of the reason code.

The value is:

MQRC_CHANNEL_ACTIVATED
(2295, X'8F7') Channel activated.

This condition is detected when a channel, which has been waiting to become active, and for which a Channel Not Activated event has been generated, is now able to become active, because an active slot has been released by another channel.

This event is not generated for a channel which is able to become active without waiting for an active slot to be released.

Event data

QMgrName (MQCFST)

Name of the queue manager generating the event (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

XmitQName (MQCFST)
Transmission queue name (parameter identifier: MQCACH_XMIT_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

This is not returned for commands containing a generic name; it is not returned for a receiver or a server-connection channel type.

ConnectionName (MQCFST)
Connection name (parameter identifier: MQCACH_CONNECTION_NAME).

The maximum length of the string is MQ_CONN_NAME_LENGTH.

For TCP/IP this is the internet address only if the channel has successfully established a connection. Otherwise it is the contents of the *ConnectionName* field in the channel definition.

This is not returned for commands containing a generic name.

Channel Auto-definition Error

> *This event is supported only if you are using MQSeries for AS/400 V4R2, or any MQSeries Version 5 product.*

Details of the condition generating the event are given, in the following text, in the *Reason* parameter of the event header.

Event message

When an event is generated, an event message is put on the SYSTEM.ADMIN.CHANNEL.EVENT queue.

The event message consists of the:

- Event header, containing a reason code parameter with a value of MQRC_CHANNEL_AUTO_DEF_ERROR
- Event data

Event data summary

Always returned:

QMgrName, *ChannelName*, *ChannelType*, *ErrorIdentifier*, *ConnectionName*

Returned optionally:

AuxErrorDataInt1

Event header

Reason (MQLONG)

Name of the reason code.

The value is:

MQRC_CHANNEL_AUTO_DEF_ERROR
(2234, X'8BA') Automatic channel definition failed.

This condition is detected when the automatic definition of a channel fails; this may be because an error occurred during the definition process, or because the channel automatic-definition exit inhibited the definition. Additional information is returned in the event message indicating the reason for the failure.

Event data

QMgrName (MQCFST)

The name of the queue manager generating the event (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

ChannelName (MQCFST)

Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

Specifies the name of the channel for which the auto-definition has failed.

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

ChannelType (MQCFIN)
Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

Specifies the type of the channel for which the auto-definition has failed.

The value is one of the following:

MQCHT_RECEIVER
Receiver.

MQCHT_SVRCONN
Server-connection (for use by clients).

ErrorIdentifier (MQCFIN)
Identifier of the cause of the error (parameter identifier: MQIACF_ERROR_IDENTIFIER).

This contains the reason code (MQRC_* or MQRCCF_*) resulting from the channel definition attempt, or else the value MQRCCF_SUPPRESSED_BY_EXIT) if the attempt to create the definition was disallowed by the exit.

ConnectionName (MQCFST)
Connection name (parameter identifier: MQCACH_CONNECTION_NAME).

Name of partner attempting to establish connection.

The maximum length of the string is MQ_CONN_NAME_LENGTH.

AuxErrorDataInt1 (MQCFIN)
Auxiliary error data (parameter identifier: MQIACF_AUX_ERROR_DATA_INT_1).

This is present only if *ErrorIdentifier* contains MQRCCF_SUPPRESSED_BY_EXIT. It contains the value returned by the exit in the *Feedback* field of the MQCXP to indicate why the auto definition has been disallowed.

Channel Auto-definition OK

> *This event is supported only if you are using MQSeries for AS/400 V4R2, or any MQSeries Version 5 product.*

Details of the condition generating the event are given, in the following text, in the *Reason* parameter of the event header.

Event message

When an event is generated, an event message is put on the SYSTEM.ADMIN.CHANNEL.EVENT queue.

The event message consists of the:

- Event header, containing a reason code parameter with a value of MQRC_CHANNEL_AUTO_DEF_OK
- Event data

Event data summary

Always returned:

QMgrName, *ChannelName*, *ChannelType*, *ConnectionName*

Event header

Reason (MQLONG)

Name of the reason code.

The value is:

MQRC_CHANNEL_AUTO_DEF_OK
(2233, X'8B9') Automatic channel definition succeeded.

This condition is detected when the automatic definition of a channel is successful. The channel is defined by the MCA.

Event data

QMgrName (MQCFST)

The name of the queue manager generating the event (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

ChannelName (MQCFST)

Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

Specifies the name of the channel being defined.

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

ChannelType (MQCFIN)
Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

Specifies the type of channel being defined.

The value is one of the following:

MQCHT_RECEIVER
Receiver.

MQCHT_SVRCONN
Server-connection (for use by clients).

ConnectionName (MQCFST)
Connection name (parameter identifier: MQCACH_CONNECTION_NAME).

Name of partner attempting to establish connection.

The maximum length of the string is MQ_CONN_NAME_LENGTH.

Channel Conversion Error

This event is not produced if you are using CICS for distributed queue management in MQSeries for MVS/ESA.

Note to users

MQSeries for Windows V2.1 ***does not*** define the channel event queue for you, so the default action is ***not to generate channel events***. This is because, once you have defined a channel event queue, you cannot stop channel event messages being generated. If you want MQ to generate channel events, you must define the channel event queue yourself using the name SYSTEM.ADMIN.CHANNEL.EVENT.

Details of the condition generating the event are given, in the following text, in the *Reason* parameter of the event header.

Event message

When an event is generated, an event message is put on the SYSTEM.ADMIN.CHANNEL.EVENT queue.

The event message consists of the:

- Event header, containing a reason code parameter with a value of MQRC_CHANNEL_CONV_ERROR
- Event data

Event data summary

Always returned:

QMgrName, *ConversionReasonCode*, *ChannelName*, *Format*,

Returned optionally:

XmitQName, *ConnectionName*

Event header

Reason (MQLONG)

Name of the reason code generating the event.

The value is:

MQRC_CHANNEL_CONV_ERROR
(2284, X'8EC') Channel conversion error.

This condition is detected when a channel is unable to do data conversion and the MQGET call to get a message from the transmission queue resulted in a data conversion error. The conversion reason code identifies the reason for the failure.

Event data

QMgrName (MQCFST)

Name of the queue manager generating the event (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

ConversionReasonCode (MQCFIN)

Identifier of the cause of the conversion error (parameter identifier: MQIACF_CONV_REASON_CODE).

The value can be one of the following:

MQRC_CONVERTED_MSG_TOO_BIG
(2120, X'848') Converted message too big for application buffer.

MQRC_FORMAT_ERROR
(2110, X'83E') Message format not valid.

MQRC_NOT_CONVERTED
(2119, X'847') Application message data not converted.

MQRC_SOURCE_CCSID_ERROR
(2111, X'83F') Source coded character set identifier not valid.

MQRC_SOURCE_DECIMAL_ENC_ERROR
(2113, X'841') Packed-decimal encoding in message not recognized.

MQRC_SOURCE_FLOAT_ENC_ERROR
(2114, X'842') Floating-point encoding in message not recognized.

MQRC_SOURCE_INTEGER_ENC_ERROR
(2112, X'840') Integer encoding in message not recognized.

MQRC_TARGET_CCSID_ERROR
(2115, X'843') Target coded character set identifier not valid.

MQRC_TARGET_DECIMAL_ENC_ERROR
(2117, X'845') Packed-decimal encoding specified by receiver not recognized.

MQRC_TARGET_FLOAT_ENC_ERROR
(2118, X'846') Floating-point encoding specified by receiver not recognized.

MQRC_TARGET_INTEGER_ENC_ERROR
(2116, X'844') Integer encoding specified by receiver not recognized.

MQRC_TRUNCATED_MSG_ACCEPTED
(2079, X'81F') Truncated message returned (processing completed).

MQRC_TRUNCATED_MSG_FAILED
(2080, X'820') Truncated message returned (processing not completed).

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Format (MQCFST)
Name of format (parameter identifier: MQCACH_FORMAT_NAME).

The maximum length of the string is MQ_FORMAT_LENGTH.

XmitQName (MQCFST)
Transmission queue name (parameter identifier: MQCACH_XMIT_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

ConnectionName (MQCFST)
Connection name (parameter identifier: MQCACH_CONNECTION_NAME).

The maximum length of the string is MQ_CONN_NAME_LENGTH.

For TCP/IP this is the internet address only if the channel has successfully established a connection. Otherwise it is the contents of the *ConnectionName* field in the channel definition.

Channel Not Activated

This event is not produced if you are using CICS for distributed queue management in MQSeries for MVS/ESA.

Note to users

MQSeries for Windows V2.1 ***does not*** define the channel event queue for you, so the default action is ***not to generate channel events***. This is because, once you have defined a channel event queue, you cannot stop channel event messages being generated. If you want MQ to generate channel events, you must define the channel event queue yourself using the name SYSTEM.ADMIN.CHANNEL.EVENT.

Details of the condition generating the event are given, in the following text, in the *Reason* parameter of the event header.

Event message

When an event is generated, an event message is put on the SYSTEM.ADMIN.CHANNEL.EVENT queue.

The event message consists of the:

- Event header, containing a reason code parameter with a value of MQRC_CHANNEL_NOT_ACTIVATED.
- Event data

Event data summary

Always returned:

QMgrName, *ChannelName*,

Returned optionally:

XmitQName, *ConnectionName*

Event header

Reason (MQLONG)

Name of the reason code.

The value is:

MQRC_CHANNEL_NOT_ACTIVATED
(2296, X'8F8') Channel cannot be activated.

This condition is detected when a channel is required to become active, either because it is starting, or because it is about to make another attempt to establish connection with its partner. However, it is unable to do so because the limit on the number of active channels has been reached (see the MaxActiveChannels parameter in the qm.ini file, or, for MVS/ESA see the ACTCHL parameter in CSQXPARM). The channel

waits until it is able to take over an active slot released when another channel ceases to be active. At that time a Channel Activated event is generated.

Event data

QMgrName (MQCFST)

Name of the queue manager generating the event (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

ChannelName (MQCFST)

Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

XmitQName (MQCFST)

Transmission queue name (parameter identifier: MQCACH_XMIT_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

This is not returned for commands containing a generic name; it is not returned for a receiver or server-connection channel type.

ConnectionName (MQCFST)

Connection name (parameter identifier: MQCACH_CONNECTION_NAME).

The maximum length of the string is MQ_CONN_NAME_LENGTH.

For TCP/IP this is the Internet address only if the channel has successfully established a connection. Otherwise it is the contents of the *ConnectionName* field in the channel definition.

This is not returned for commands containing a generic name.

Channel Started

This event is not produced if you are using CICS for distributed queue management in MQSeries for MVS/ESA.

Note to users

MQSeries for Windows V2.1 ***does not*** define the channel event queue for you, so the default action is ***not to generate channel events***. This is because, once you have defined a channel event queue, you cannot stop channel event messages being generated. If you want MQ to generate channel events, you must define the channel event queue yourself using the name SYSTEM.ADMIN.CHANNEL.EVENT.

Details of the condition generating the event are given, in the following text, in the *Reason* parameter of the event header.

Event message

When an event is generated, an event message is put on the SYSTEM.ADMIN.CHANNEL.EVENT queue.

The event message consists of the:

- Event header, containing a reason code parameter with a value of MQRC_CHANNEL_STARTED
- Event data

Event data summary

Always returned:

QMgrName, *ChannelName*,

Returned optionally:

XmitQName, *ConnectionName*

Event header

Reason (MQLONG)

Name of the reason code.

The value is:

MQRC_CHANNEL_STARTED
(2282, X'8EA') Channel started.

Either

- An operator has issued a Start Channel command, or
- An instance of a channel has been successfully established.

  This condition is detected when Initial Data negotiation is complete and resynchronization has

been performed where necessary such that message transfer can proceed.

Event data

QMgrName (MQCFST)

Name of the queue manager generating the event (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

ChannelName (MQCFST)

Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

XmitQName (MQCFST)

Transmission queue name (parameter identifier: MQCACH_XMIT_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

This is not returned for commands containing a generic name; it is not returned for a receiver or server-connection channel type.

ConnectionName (MQCFST)

Connection name (parameter identifier: MQCACH_CONNECTION_NAME).

The maximum length of the string is MQ_CONN_NAME_LENGTH.

For TCP/IP this is the Internet address only if the channel has successfully established a connection. Otherwise it is the contents of the *ConnectionName* field in the channel definition.

This is not returned for commands containing a generic name.

Channel Stopped

This event is not produced if you are using CICS for distributed queue management in MQSeries for MVS/ESA.

Note to users

MQSeries for Windows V2.1 ***does not*** define the channel event queue for you, so the default action is ***not to generate channel events***. This is because, once you have defined a channel event queue, you cannot stop channel event messages being generated. If you want MQ to generate channel events, you must define the channel event queue yourself using the name SYSTEM.ADMIN.CHANNEL.EVENT.

Details of the condition generating the event are given, in the following text, in the *Reason* parameter of the event header.

Event message

When an event is generated, an event message is put on the SYSTEM.ADMIN.CHANNEL.EVENT queue.

The event message consists of the:

- Event header, containing a reason code parameter with a value of MQRC_CHANNEL_STOPPED
- Event data

Event data summary

Always returned:

QMgrName, *ReasonQualifier*, *ChannelName*, *ErrorIdentifier*, *AuxErrorDataInt1*, *AuxErrorDataInt2*, *AuxErrorDataStr1*, *AuxErrorDataStr2*, *AuxErrorDataStr3*

Returned optionally:

XmitQName, *ConnectionName*

Event header

Reason (MQLONG)

Name of the reason code.

The value is:

MQRC_CHANNEL_STOPPED
(2283, X'8EB') Channel stopped.

This condition is detected when the channel has been stopped. The reason qualifier identifies the reasons for stopping.

Event data

QMgrName (MQCFST)

The name of the queue manager generating the event (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

ReasonQualifier (MQCFIN)

Identifier that qualifies the reason code (parameter identifier: MQIACF_REASON_QUALIFIER).

The value is one of the following:

MQRQ_CHANNEL_STOPPED_OK
: Channel has been closed with either a zero return code or a warning return code.

MQRQ_CHANNEL_STOPPED_ERROR
: Channel has been closed but there is an error reported and the channel is not in stopped or retry state.

MQRQ_CHANNEL_STOPPED_RETRY
: Channel has been closed and it is in retry state.

MQRQ_CHANNEL_STOPPED_DISABLED
: Channel has been closed and it is in a stopped state.

ChannelName (MQCFST)

Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

ErrorIdentifier (MQCFIN)

Identifier of the cause of the error (parameter identifier: MQIACF_ERROR_IDENTIFIER).

When a channel is stopped due to an error, this is the code that identifies the error. If the event message is because of a channel stop failure, the following fields are set:

1. *ReasonQualifier*, containing the value MQRQ_CHANNEL_STOPPED_ERROR
2. *ErrorIdentifier*, containing the code number of an error message that describes the error
3. *AuxErrorDataInt1*, containing error message integer insert 1
4. *AuxErrorDataInt2*, containing error message integer insert 2
5. *AuxErrorDataStr1*, containing error message string insert 1
6. *AuxErrorDataStr2*, containing error message string insert 2
7. *AuxErrorDataStr3*, containing error message string insert 3

The meanings of the error message inserts depend on the code number of the error message. Details of error-message code numbers and the inserts for specific platforms can be found as follows:

- For MVS/ESA, see the section “Distributed queuing message codes” in the *MQSeries for MVS/ESA Messages and Codes* book.
- For other platforms, the last four digits of *ErrorIdentifier* when displayed in hexadecimal notation indicate the decimal code number of the error message.

  For example, if *ErrorIdentifier* has the value X'xxxxyyyy', the message code of the error message explaining the error is AMQyyyy.

AuxErrorDataInt1 (MQCFIN)
: First integer of auxiliary error data for channel errors (parameter identifier: MQIACF_AUX_ERROR_DATA_INT_1).

  When a channel is in a stopped condition due to an error, this is the first integer parameter that qualifies the error. This information is for use by IBM service personnel; include it in any problem report that you submit to IBM regarding this event message.

AuxErrorDataInt2 (MQCFIN)
: Second integer of auxiliary error data for channel errors (parameter identifier: MQIACF_AUX_ERROR_DATA_INT_2).

  If the channel is stopped due to an error, this is the second integer parameter that qualifies the error. This information is for use by IBM service personnel; include it in any problem report that you submit to IBM regarding this event message.

AuxErrorDataStr1 (MQCFST)
: First string of auxiliary error data for channel errors (parameter identifier: MQCACF_AUX_ERROR_DATA_STR_1).

  If the channel is stopped due to an error, this is the first string parameter that qualifies the error. This information is for use by IBM service personnel; include it in any problem report that you submit to IBM regarding this event message.

AuxErrorDataStr2 (MQCFST)
: Second string of auxiliary error data for channel errors (parameter identifier: MQCACF_AUX_ERROR_DATA_STR_2).

  If the channel is stopped due to an error, this is the second string parameter that qualifies the error. This information is for use by IBM service personnel; include it in any problem report that you submit to IBM regarding this event message.

AuxErrorDataStr3 (MQCFST)
: Third string of auxiliary error data for channel errors (parameter identifier: MQCACF_AUX_ERROR_DATA_STR_3).

  If the channel is stopped due to an error, this is the third string parameter that qualifies the error. This information is for use by IBM service personnel; include it in any problem report that you submit to IBM regarding this event message.

XmitQName (MQCFST)
Transmission queue name (parameter identifier: MQCACH_XMIT_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

This is not returned for commands containing a generic name. It is not returned for a receiver or server-connection channel type.

ConnectionName (MQCFST)
Connection name (parameter identifier: MQCACH_CONNECTION_NAME).

The maximum length of the string is MQ_CONN_NAME_LENGTH.

For TCP/IP this is the internet address only if the channel has successfully established a connection. Otherwise it is the contents of the *ConnectionName* field in the channel definition.

This is not returned for commands containing a generic name.

Default Transmission Queue Type Error

Details of the condition generating the event are given, in the following text, in the *Reason* parameter of the event header.

Event message

When an event is generated, an event message is put on the SYSTEM.ADMIN.QMGR.EVENT queue.

The event message consists of the:

- Event header, containing a reason code parameter with a value of MQRC_DEF_XMIT_Q_TYPE_ERROR
- Event data

Event data summary

Always returned:

QMgrName, *QName*, *XmitQName*, *QType*, *ApplType*, *ApplName*

Returned optionally:

ObjectQMgrName

Event header

Reason (MQLONG)

Name of the reason code.

The value is:

MQRC_DEF_XMIT_Q_TYPE_ERROR
(2198, X'896') Default transmission queue not local.

An MQOPEN or MQPUT1 call was issued specifying a remote queue as the destination. Either a local definition of the remote queue was specified, or a queue-manager alias was being resolved, but in either case the *XmitQName* attribute in the local definition is blank.

No transmission queue is defined with the same name as the destination queue manager, so the local queue manager has attempted to use the default transmission queue. However, although there is a queue defined by the *DefXmitQName* queue-manager attribute, it is not a local queue. See the *MQSeries Application Programming Guide* for more information.

Event data

QMgrName (MQCFST)

Name of the queue manager generating the event (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

QName (MQCFST)
Queue name from object descriptor (MQOD) (parameter identifier: MQCA_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

XmitQName (MQCFST)
Default transmission queue name (parameter identifier: MQCA_XMIT_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

QType (MQCFIN)
Type of default transmission queue (parameter identifier: MQIA_Q_TYPE).

The value can be:

MQQT_ALIAS
Alias queue definition.

MQQT_REMOTE
Local definition of a remote queue.

ApplType (MQCFIN)
Type of application making the MQI call that caused the event (parameter identifier: MQIA_APPL_TYPE).

ApplName (MQCFST)
Name of the application making the MQI call that caused the event (parameter identifier: MQCACF_APPL_NAME).

The maximum length of the string is MQ_APPL_NAME_LENGTH.

Note: If the application is a server for clients, the *ApplType* and *ApplName* parameters identify the server, rather than a client.

ObjectQMgrName (MQCFST)
Name of the object queue manager (parameter identifier: MQCACF_OBJECT_Q_MGR_NAME).

This parameter is returned if the *ObjectName* in the object descriptor (MQOD) (when the object was opened) is not the queue manager currently connected.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

Default Transmission Queue Usage Error

Details of the condition generating the event are given, in the following text, in the *Reason* parameter of the event header.

Event message

When an event is generated, an event message is put on the SYSTEM.ADMIN.QMGR.EVENT queue.

The event message consists of the:

- Event header, containing a reason code parameter with a value of MQRC_DEF_XMIT_Q_USAGE_ERROR
- Event data

Event data summary

Always returned:

QMgrName, *QName*, *XmitQName*, *ApplType*, *ApplName*

Returned optionally:

ObjectQMgrName

Event header

Reason (MQLONG)

Name of the reason code.

The value is:

MQRC_DEF_XMIT_Q_USAGE_ERROR
(2199, X'897') Default transmission queue usage error.

An MQOPEN or MQPUT1 call was issued specifying a remote queue as the destination. Either a local definition of the remote queue was specified, or a queue-manager alias was being resolved, but in either case the *XmitQName* attribute in the local definition is blank.

No transmission queue is defined with the same name as the destination queue manager, so the local queue manager has attempted to use the default transmission queue. However, the queue defined by the *DefXmitQName* queue-manager attribute does not have a *Usage* attribute of MQUS_TRANSMISSION. See the *MQSeries Application Programming Guide* for more information.

Event data

QMgrName (MQCFST)

Name of the queue manager generating the event (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

QName (MQCFST)
Queue name from object descriptor (MQOD) (parameter identifier: MQCA_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

XmitQName (MQCFST)
Default transmission queue name (parameter identifier: MQCA_XMIT_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

ApplType (MQCFIN)
Type of application making the MQI call that caused the event (parameter identifier: MQIA_APPL_TYPE).

ApplName (MQCFST)
Name of the application making the MQI call that caused the event (parameter identifier: MQCACF_APPL_NAME).

The maximum length of the string is MQ_APPL_NAME_LENGTH.

Note: If the application is a server for clients, the *ApplType* and *ApplName* parameters identify the server, rather than a client.

ObjectQMgrName (MQCFST)
Name of the object queue manager (parameter identifier: MQCACF_OBJECT_Q_MGR_NAME).

This parameter is returned if the *ObjectName* in the object descriptor (MQOD) (when the object was opened) is not the queue manager currently connected.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

Get Inhibited

Details of the condition generating the event are given, in the following text, in the *Reason* parameter of the event header.

Event message

When an event is generated, an event message is put on the SYSTEM.ADMIN.QMGR.EVENT queue.

The event message consists of the:

- Event header, containing a reason code parameter with a value of MQRC_GET_INHIBITED
- Event data

Event data summary

Always returned:

QMgrName, *QName*, *ApplType*, *ApplName*,

Returned optionally:

None

Event header

Reason (MQLONG)

Name of the reason code.

The value is:

MQRC_GET_INHIBITED

(2016, X'7E0') Gets inhibited for the queue.

MQGET calls are currently inhibited for the queue (see the *InhibitGet* queue attribute in the description of attributes common to all queues in the *MQSeries Application Programming Reference*) or for the queue to which this queue resolves.

Event data

QMgrName (MQCFST)

Name of the queue manager generating the event (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

QName (MQCFST)

Queue name from object descriptor (MQOD) (parameter identifier: MQCA_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

ApplType (MQCFIN)

Type of the application that issued the get (parameter identifier: MQIA_APPL_TYPE).

ApplName (MQCFST)

Name of the application that issued the get (parameter identifier: MQCACF_APPL_NAME).

The maximum length of the string is MQ_APPL_NAME_LENGTH.

Note: If the application is a server for clients, the *ApplType* and *ApplName* parameters identify the server, rather than the client.

Not Authorized (type 1)

This event is not supported if you are using MQSeries for MVS/ESA, MQSeries for OS/2, or MQSeries for Windows Version 2.1.

Details of the condition generating the event are given, in the following text, in the *Reason* parameter of the event header.

Event message

When an event is generated, an event message is put on the SYSTEM.ADMIN.QMGR.EVENT queue.

The event message consists of the:

- Event header, containing a reason code parameter with a value of MQRC_NOT_AUTHORIZED
- Event data

Event data summary

Always returned:

QMgrName, *ReasonQualifier*, *UserIdentifier*, *ApplType*, *ApplName*

Returned optionally:

None

Event header

Reason (MQLONG)

Name of the reason code.

The value is:

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

On an MQCONN call, the user is not authorized to connect to the queue manager.

Event data

QMgrName (MQCFST)

Name of the queue manager generating the event (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

ReasonQualifier (MQCFIN)

Identifier for type 1 authority events (parameter identifier: MQIACF_REASON_QUALIFIER).

The value must be:

MQRQ_CONN_NOT_AUTHORIZED
Connection not authorized.

UserIdentifier (MQCFST)
User identifier that caused the authorization check (parameter identifier: MQCACF_USER_IDENTIFIER).

The maximum length of the string is MQ_USER_ID_LENGTH.

ApplType (MQCFIN)
Type of application cauisng the event (parameter identifier: MQIA_APPL_TYPE).

ApplName (MQCFST)
Name of the application causing the event (parameter identifier: MQCACF_APPL_NAME).

The maximum length of the string is MQ_APPL_NAME_LENGTH.

Note: If the application is a server for clients, the *ApplType* and *ApplName* parameters identify the server, rather than the client.

Not Authorized (type 2)

This event is not supported if you are using MQSeries for MVS/ESA, MQSeries for OS/2, MQSeries for Tandem NSK Version 2.2, or MQSeries for Windows Version 2.1.

Details of the condition generating the event are given, in the following text, in the *Reason* parameter of the event header.

Event message

When an event is generated, an event message is put on the SYSTEM.ADMIN.QMGR.EVENT queue.

The event message consists of the:

- Event header, containing a reason code parameter with a value of MQRC_NOT_AUTHORIZED
- Event data

Event data summary

Always returned:

QMgrName, *ReasonQualifier*, *Options*, *UserIdentifier*, *ApplType*, *ApplName*

Returned optionally:

ObjectQMgrName, *QName*, *ProcessName*

Event header

Reason (MQLONG)

Name of the reason code.

The value is:

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

On an MQOPEN or MQPUT1 call, the user is not authorized to open the object for the option(s) specified.

Event data

QMgrName (MQCFST)

Name of the queue manager generating the event (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

ReasonQualifier (MQCFIN)

Identifier for type 2 authority events (parameter identifier: MQIACF_REASON_QUALIFIER).

The value must be:

MQRQ_OPEN_NOT_AUTHORIZED
Open not authorized.

Options (MQCFIN)
Options specified on the MQOPEN call (parameter identifier: MQIACF_OPEN_OPTIONS).

UserIdentifier (MQCFST)
User identifier that caused the authorization check (parameter identifier: MQCACF_USER_IDENTIFIER).

The maximum length of the string is MQ_USER_ID_LENGTH.

ApplType (MQCFIN)
Type of application causing the authorization check (parameter identifier: MQIA_APPL_TYPE).

ApplName (MQCFST)
Name of the application causing the authorization check (parameter identifier: MQCACF_APPL_NAME).

The maximum length of the string is MQ_APPL_NAME_LENGTH.

Note: If the application is a server for clients, the *ApplType* and *ApplName* parameters identify the server, rather than the client.

ObjectQMgrName (MQCFST)
The name of the object queue manager (parameter identifier: MQCACF_OBJECT_Q_MGR_NAME).

This parameter is returned if the *ObjectName* in the object descriptor (MQOD) (when the object was opened) is not the queue manager currently connected.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

QName (MQCFST)
Queue name from object descriptor (MQOD) (parameter identifier: MQCA_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

ProcessName (MQCFST)
Name of the process whose attributes have changed (parameter identifier: MQCA_PROCESS_NAME).

The maximum length of the string is MQ_PROCESS_NAME_LENGTH.

Not Authorized (type 3)

This event is not supported if you are using MQSeries for MVS/ESA, MQSeries for OS/2, MQSeries for Tandem NSK Version 2.2, or MQSeries for Windows Version 2.1.

Details of the condition generating the event are given, in the following text, in the *Reason* parameter of the event header.

Event message

When an event is generated, an event message is put on the SYSTEM.ADMIN.QMGR.EVENT queue.

The event message consists of the:

- Event header, containing a reason code parameter with a value of MQRC_NOT_AUTHORIZED
- Event data

Event data summary

Always returned:

QMgrName, *ReasonQualifier*, *QName*, *UserIdentifier*, *ApplType*, *ApplName*

Returned optionally:

None

Event header

Reason (MQLONG)

Name of the reason code.

The value is:

MQRC_NOT_AUTHORIZED

(2035, X'7F3') Not authorized for access.

On an MQCLOSE call, the user is not authorized to delete the object, which is a permanent dynamic queue, and the *Hobj* parameter specified on the MQCLOSE call is not the handle returned by the MQOPEN call which created the queue.

Event data

QMgrName (MQCFST)

Name of the queue manager generating the event (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

ReasonQualifier (MQCFIN)

Identifier for type 3 authority events (parameter identifier: MQIACF_REASON_QUALIFIER).

The value is:

MQRQ_CLOSE_NOT_AUTHORIZED
Close not authorized.

QName (MQCFST)
Queue name from object descriptor (MQOD) (parameter identifier: MQCA_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

UserIdentifier (MQCFST)
User identifier that caused the authorization check (parameter identifier: MQCACF_USER_IDENTIFIER).

The maximum length of the string is MQ_USER_ID_LENGTH.

ApplType (MQCFIN)
Type of application that caused the authorization check (parameter identifier: MQIA_APPL_TYPE).

ApplName (MQCFST)
Name of the application causing the authorization check (parameter identifier: MQCACF_APPL_NAME).

The maximum length of the string is MQ_APPL_NAME_LENGTH.

Note: If the application is a server for clients, the *ApplType* and *ApplName* parameters identify the server, rather than the client.

Not Authorized (type 4)

This event is not supported if you are using MQSeries for MVS/ESA, MQSeries for OS/2, MQSeries for Tandem NSK Version 2.2, or MQSeries for Windows Version 2.1.

Details of the condition generating the event are given, in the following text, in the *Reason* parameter of the event header.

Event message

When an event is generated, an event message is put on the SYSTEM.ADMIN.QMGR.EVENT queue.

The event message consists of the:

- Event header, containing a reason code parameter with a value of MQRC_NOT_AUTHORIZED
- Event data

Event data summary

Always returned:

QMgrName, *ReasonQualifier*, *Command*, *UserIdentifier*

Returned optionally:

None

Event header

Reason (MQLONG)

Name of the reason code.

The value is:

MQRC_NOT_AUTHORIZED

(2035, X'7F3') Not authorized for access.

Indicates that a command has been issued from a user ID that is not authorized to access the object specified in the command.

Event data

QMgrName (MQCFST)

Name of the queue manager generating the event (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

ReasonQualifier (MQCFIN)

Identifier for type 4 authority events (parameter identifier: MQIACF_REASON_QUALIFIER).

The value must be:

MQRQ_CMD_NOT_AUTHORIZED
Command not authorized.

Command (MQCFIN)
Identifier for the command (parameter identifier: MQIACF_COMMAND).

See the PCF header (MQCFH) structure, described on page 334

UserIdentifier (MQCFST)
User identifier that caused the authorization check (parameter identifier: MQCACF_USER_IDENTIFIER).

The maximum length of the string is MQ_USER_ID_LENGTH.

Put Inhibited

Details of the condition generating the event are given, in the following text, in the *Reason* parameter of the event header.

Event message

When an event is generated, an event message is put on the SYSTEM.ADMIN.QMGR.EVENT queue.

The event message consists of the:

- Event header, containing a reason code parameter with a value of MQRC_PUT_INHIBITED
- Event data

Event data summary

Always returned:

QMgrName, *QName*, *ApplType*, *ApplName*

Returned optionally:

ObjectQMgrName

Event header

Reason (MQLONG)

Name of the reason code generating the event.

The value is:

MQRC_PUT_INHIBITED
(2051, X'803') Put calls inhibited for the queue.

MQPUT and MQPUT1 calls are currently inhibited for the queue (see the *InhibitPut* queue attribute in the description of attributes common to all queues in the *MQSeries Application Programming Reference*) or for the queue to which this queue resolves.

Event data

QMgrName (MQCFST)

Name of the queue manager generating the event (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

QName (MQCFST)

Queue name from object descriptor (MQOD) (parameter identifier: MQCA_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

ApplType (MQCFIN)

Type of the application that issued the put (parameter identifier: MQIA_APPL_TYPE).

ApplName (MQCFST)

Name of the application that issued the put (parameter identifier: MQCACF_APPL_NAME).

The maximum length of the string is MQ_APPL_NAME_LENGTH.

Note: If the application is a server for clients, the *ApplType* and *ApplName* parameters identify the server, rather than the client.

ObjectQMgrName (MQCFST)
Queue-manager name from object descriptor (MQOD) (parameter identifier: MQCACF_OBJECT_Q_MGR_NAME).

This parameter is returned only if it has a value that is different from *QMgrName*. This occurs when the *ObjectQMgrName* field in the object descriptor provided by the application on the MQOPEN or MQPUT1 call is neither blank nor the name of the application's local queue manager. However, it can also occur when *ObjectQMgrName* in the object descriptor is blank, but a name service provides a queue-manager name which is not the name of the application's local queue manager.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

Queue Depth High

Details of the condition generating the event are given, in the following text, in the *Reason* parameter of the event header.

Event message

When an event is generated, an event message is put on the SYSTEM.ADMIN.PERFM.EVENT queue.

The event message consists of the:

- Event header, containing a reason code parameter with a value of MQRC_Q_DEPTH_HIGH
- Event data

Event data summary

Always returned:

QMgrName, *QName*, *TimeSinceReset*, *HighQDepth*, *MsgEnqCount*, *MsgDeqCount*

Returned optionally:

None

Event header

Reason (MQLONG)

Name of the reason code.

The value is:

MQRC_Q_DEPTH_HIGH

(2224, X'8B0') Queue depth high limit reached or exceeded.

An MQPUT or MQPUT1 call has caused the queue depth to be incremented to or above the limit specified in the *QDepthHighLimit* attribute.

Corrective action: None. This reason code is only used to identify the corresponding event message.

Event data

QMgrName (MQCFST)

Name of the queue manager generating the event (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

QName (MQCFST)

Name of the queue on which the limit has been reached (parameter identifier: MQCA_BASE_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

TimeSinceReset (MQCFIN)

Time, in seconds, since the statistics were last reset (parameter identifier: MQIA_TIME_SINCE_RESET).

The value recorded by this timer is also used as the *interval time* in queue service interval events.

HighQDepth (MQCFIN)
Maximum number of messages on the queue since the queue statistics were last reset (parameter identifier: MQIA_HIGH_Q_DEPTH).

MsgEnqCount (MQCFIN)
Number of messages enqueued (parameter identifier: MQIA_MSG_ENQ_COUNT).

That is, the number of messages put on the queue since the queue statistics were last reset.

MsgDeqCount (MQCFIN)
Number of messages removed from the queue (parameter identifier: MQIA_MSG_DEQ_COUNT).

That is, the number of messages removed from the queue since the queue statistics were last reset.

Queue Depth Low

Details of the condition generating the event are given, in the following text, in the *Reason* parameter of the event header.

Event message

When an event is generated, an event message is put on the SYSTEM.ADMIN.PERFM.EVENT queue.

The event message consists of the:

- Event header, containing a reason code parameter with a value of MQRC_Q_DEPTH_LOW
- Event data

Event data summary

Always returned:

QMgrName, *QName*, *TimeSinceReset*, *HighQDepth*, *MsgEnqCount*, *MsgDeqCount*

Returned optionally:

None

Event header

Reason (MQLONG)

Name of the reason code.

The value is:

MQRC_Q_DEPTH_LOW

(2225, X'8B1') Queue depth low limit reached or exceeded.

An MQGET call has caused the queue depth to be decremented to or below the limit specified in the *QDepthLowLimit* attribute.

Corrective action: None. This reason code is only used to identify the corresponding event message.

Event data

QMgrName (MQCFST)

Name of the queue manager generating the event (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

QName (MQCFST)

Name of the queue on which the limit has been reached (parameter identifier: MQCA_BASE_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

TimeSinceReset (MQCFIN)

Time, in seconds, since the statistics were last reset (parameter identifier: MQIA_TIME_SINCE_RESET).

The value recorded by this timer is also used as the *interval time* in queue service interval events.

HighQDepth (MQCFIN)
Maximum number of messages on the queue since the queue statistics were last reset (parameter identifier: MQIA_HIGH_Q_DEPTH).

MsgEnqCount (MQCFIN)
Number of messages enqueued (parameter identifier: MQIA_MSG_ENQ_COUNT).

That is, the number of messages put on the queue since the queue statistics were last reset.

MsgDeqCount (MQCFIN)
Number of messages removed from the queue (parameter identifier: MQIA_MSG_DEQ_COUNT).

That is, the number of messages removed from the queue since the queue statistics were last reset.

Queue Full

Details of the condition generating the event are given, in the following text, in the *Reason* parameter of the event header.

Event message

When an event is generated, an event message is put on the SYSTEM.ADMIN.PERFM.EVENT queue.

The event message consists of the:

- Event header, containing a reason code parameter with a value of MQRC_Q_FULL
- Event data

Event data summary

Always returned:

QMgrName, *QName*, *TimeSinceReset*, *HighQDepth*, *MsgEnqCount*, *MsgDeqCount*

Returned optionally:

None

Event header

Reason (MQLONG)

Name of the reason code.

The value is:

MQRC_Q_FULL

(2053, X'805') Queue already contains maximum number of messages.

On an MQPUT or MQPUT1 call, the call failed because the queue is full, that is it already contains the maximum number of messages possible (see the *MaxQDepth* local-queue attribute of local-queue attributes in the *MQSeries Application Programming Reference*)

This reason code can also occur in the *Feedback* field in the message descriptor of a report message; in this case it indicates that the error was encountered by a message channel agent when it attempted to put the message on a remote queue.

Corrective action: Retry the operation later. Consider increasing the maximum depth for this queue, or arranging for more instances of the application to service the queue.

Event data

QMgrName (MQCFST)

Name of the queue manager generating the event (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

QName (MQCFST)

The name of the queue on which the put was rejected (parameter identifier: MQCA_BASE_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

TimeSinceReset (MQCFIN)
Time, in seconds, since the statistics were last reset (parameter identifier: MQIA_TIME_SINCE_RESET).

HighQDepth (MQCFIN)
The maximum number of messages on a queue (parameter identifier: MQIA_HIGH_Q_DEPTH).

MsgEnqCount (MQCFIN)
Number of messages enqueued (parameter identifier: MQIA_MSG_ENQ_COUNT).

That is, the number of messages placed on the queue since queue statistics were reset.

MsgDeqCount (MQCFIN)
The number of messages removed from the queue (parameter identifier: MQIA_MSG_DEQ_COUNT).

That is, the number of messages removed from the queue since queue statistics were reset.

Queue Manager Active

> *This event is not produced for the first start of an MQSeries for MVS/ESA queue manager, only on subsequent starts.*

Details of the condition generating the event are given, in the following text, in the *Reason* parameter of the event header.

Event message

When an event is generated, an event message is put on the SYSTEM.ADMIN.QMGR.EVENT queue.

The event message consists of the:

- Event header, containing a reason code parameter with a value of MQRC_Q_MGR_ACTIVE
- Event data

Event data summary

Always returned:

QMgrName

Returned optionally:

None

Event header

Reason (MQLONG)

Name of the reason code.

The value is:

MQRC_Q_MGR_ACTIVE

(2222, X'8AE') Queue manager created.

This condition is detected when a queue manager becomes active.

On MVS/ESA, this event is not generated for the first start of a queue manager, only on subsequent restarts.

Event data

QMgrName (MQCFST)

Name of the queue manager generating the event (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

Queue Manager Not Active

This event is not produced by MQSeries for MVS/ESA.

Details of the condition generating the event are given, in the following text, in the *Reason* parameter of the event header.

Note: This event is not generated on MVS.

Event message

When an event is generated, an event message is put on the SYSTEM.ADMIN.QMGR.EVENT queue.

The event message consists of the:

- Event header, containing a reason code parameter with a value of MQRC_Q_MGR_NOT_ACTIVE
- Event data

Event data summary

Always returned: *QMgrName*, *ReasonQualifier*,

Returned optionally: None

Event header

Reason (MQLONG)

Name of the reason code.

The value is:

MQRC_Q_MGR_NOT_ACTIVE
(2223, X'8AE') Queue manager unavailable.

This condition is detected when a queue manager is requested to stop or quiesce.

Event data

QMgrName (MQCFST)

Name of the queue manager generating the event (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

ReasonQualifier (MQCFIN)

Identifier of cases of this reason code (parameter identifier: MQIACF_REASON_QUALIFIER).

This specifies the type of stop that was requested. The value is one of the following:

MQRQ_Q_MGR_STOPPING
Queue manager stopping.

MQRQ_Q_MGR_QUIESCING
Queue manager quiescing.

Queue Service Interval High

Details of the condition generating the event are given, in the following text, in the *Reason* parameter of the event header.

Event message

When an event is generated, an event message is put on the SYSTEM.ADMIN.PERFM.EVENT queue.

The event message consists of the:

- Event header, containing a reason code parameter with a value of MQRC_Q_SERVICE_INTERVAL_HIGH
- Event data

Event data summary

Always returned:

QMgrName, *QName*, *TimeSinceReset*, *HighQDepth*, *MsgEnqCount*, *MsgDeqCount*

Returned optionally:

None

Event header

Reason (MQLONG)

Name of the reason code.

The value is:

MQRC_Q_SERVICE_INTERVAL_HIGH

(2226, X'8B2') Queue service interval high.

No successful gets or puts have been detected within an interval which is greater than the limit specified in the *QServiceInterval* attribute.

Corrective action: None. This reason code is only used to identify the corresponding event message.

Event data

QMgrName (MQCFST)

Name of the queue manager generating the event (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

QName (MQCFST)

Name of the queue specified on the command which caused this queue service interval event to be generated (parameter identifier: MQCA_BASE_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

TimeSinceReset (MQCFIN)

Time, in seconds, since the statistics were reset (parameter identifier: MQIA_TIME_SINCE_RESET).

For a service interval high event, this value is greater than the service interval.

HighQDepth (MQCFIN)
Maximum number of messages on a queue, since queue statistics were reset (parameter identifier: MQIA_HIGH_Q_DEPTH).

MsgEnqCount (MQCFIN)
Number of messages enqueued (parameter identifier: MQIA_MSG_ENQ_COUNT).

That is, the number of messages put on the queue since the queue statistics were last reset.

MsgDeqCount (MQCFIN)
Number of messages removed from the queue (parameter identifier: MQIA_MSG_DEQ_COUNT).

That is, the number of messages removed from the queue since the queue statistics were last reset.

Queue Service Interval OK

Details of the condition generating the event are given, in the following text, in the *Reason* parameter of the event header.

Event message

When an event is generated, an event message is put on the SYSTEM.ADMIN.PERFM.EVENT queue.

The event message consists of the:

- Event header, containing a reason code parameter with a value of MQRC_Q_SERVICE_INTERVAL_OK
- Event data

Event data summary

Always returned:

QMgrName, *QName*, *TimeSinceReset*, *HighQDepth*, *MsgEnqCount*, *MsgDeqCount*

Returned optionally:

None

Event header

Reason (MQLONG)

Name of the reason code.

The value is:

MQRC_Q_SERVICE_INTERVAL_OK

(2227, X'8B3') Queue service interval ok.

A successful get has been detected within an interval which is less than or equal to the limit specified in the *QServiceInterval* attribute.

Corrective action: None. This reason code is only used to identify the corresponding event message.

Event data

QMgrName (MQCFST)

Name of the queue manager generating the event (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

QName (MQCFST)

Queue name specified on the command that caused this queue service interval event to be generated (parameter identifier: MQCA_BASE_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

TimeSinceReset (MQCFIN)

Time, in seconds, since the statistics were reset (parameter identifier: MQIA_TIME_SINCE_RESET).

HighQDepth (MQCFIN)
The maximum number of messages on a queue since ststistics were reset (parameter identifier: MQIA_HIGH_Q_DEPTH).

MsgEnqCount (MQCFIN)
Number of messages enqueued (parameter identifier: MQIA_MSG_ENQ_COUNT).

That is the number of messages put on the queue since the queue ststistics were last reset.

MsgDeqCount (MQCFIN)
Number of messages removed from the queue (parameter identifier: MQIA_MSG_DEQ_COUNT).

That is the number of messages removed from the queue since the queue statistics were last reset.

Queue Type Error

Details of the condition generating the event are given, in the following text, in the *Reason* parameter of the event header.

Event message

When an event is generated, an event message is put on the SYSTEM.ADMIN.QMGR.EVENT queue.

The event message consists of the:

- Event header, containing a reason code parameter with a value of MQRC_Q_TYPE_ERROR
- Event data

Event data summary

Always returned:

QMgrName, *QName*, *ApplType*, *ApplName*

Returned optionally:

ObjectQMgrName

Event header

Reason (MQLONG)

Name of the reason code.

The value is:

MQRC_Q_TYPE_ERROR
(2057, X'809') Queue type not valid.

One of the following occurred:

- On an MQOPEN call, the *ObjectQMgrName* field in the object descriptor specifies the name of a local definition of a remote queue (in order to specify a queue-manager alias), and in that local definition the *RemoteQMgrName* attribute is the name of the local queue manager. However, the *ObjectName* field specifies the name of a model queue on the local queue manager; this is not allowed. See the *MQSeries Application Programming Guide* for more information.
- On an MQPUT1 call, the object descriptor MQOD specifies the name of a model queue.
- On a previous MQPUT or MQPUT1 call, the *ReplyToQ* field in the message descriptor specified the name of a model queue, but a model queue cannot be specified as the destination for reply or report messages. Only the name of a predefined queue, or the name of the *dynamic* queue created from the model queue, can be specified as the destination. In this situation the reason code MQRC_Q_TYPE_ERROR is returned in the *Reason*

field of the MQDLH structure when the reply message or report message is placed on the dead-letter queue.

Event data

QMgrName (MQCFST)

Name of the queue manager generating the event (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

QName (MQCFST)

Queue name from object descriptor (MQOD) (parameter identifier: MQCA_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

ApplType (MQCFIN)

Type of application making the MQI call that caused the event (parameter identifier: MQIA_APPL_TYPE).

ApplName (MQCFST)

Name of the application making the MQI call that caused the event (parameter identifier: MQCACF_APPL_NAME).

The maximum length of the string is MQ_APPL_NAME_LENGTH.

Note: If the application is a server for clients, the *ApplType* and *ApplName* parameters identify the server, rather than a client.

ObjectQMgrName (MQCFST)

Name of the object queue manager (parameter identifier: MQCACF_OBJECT_Q_MGR_NAME).

This parameter is returned if the *ObjectName* in the object descriptor (MQOD) (when the object was opened) is not the queue manager currently connected.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

Remote Queue Name Error

Details of the condition generating the event are given, in the following text, in the *Reason* parameter of the event header.

Event message

When an event is generated, an event message is put on the SYSTEM.ADMIN.QMGR.EVENT queue.

The event message consists of the:

- Event header, containing a reason code parameter with a value of MQRC_REMOTE_Q_NAME_ERROR
- Event data

Event data summary

Always returned:

QMgrName, *QName*, *ApplType*, *ApplName*

Returned optionally:

ObjectQMgrName

Event header

Reason (MQLONG)

Name of the reason code.

The value is:

MQRC_REMOTE_Q_NAME_ERROR

(2184, X'888') Remote queue name not valid.

On an MQOPEN or MQPUT1 call, one of the following occurred:

- A local definition of a remote queue (or an alias to one) was specified, but the *RemoteQName* attribute in the remote queue definition is entirely blank. Note that this error occurs even if the *XmitQName* in the definition is not blank.
- The *ObjectQMgrName* field in the object descriptor was not blank and not the name of the local queue manager, but the *ObjectName* field is blank.

Event data

QMgrName (MQCFST)

Name of the queue manager generating the event (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

QName (MQCFST)

Queue name from object descriptor (MQOD) (parameter identifier: MQCA_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

ApplType (MQCFIN)
Type of application making the MQI call that caused the event (parameter identifier: MQIA_APPL_TYPE).

ApplName (MQCFST)
Name of the application making the MQI call that caused the event (parameter identifier: MQCACF_APPL_NAME).

The maximum length of the string is MQ_APPL_NAME_LENGTH.

Note: If the application is a server for clients, the *ApplType* and *ApplName* parameters identify the server, rather than a client.

ObjectQMgrName (MQCFST)
Name of the object queue manager (parameter identifier: MQCACF_OBJECT_Q_MGR_NAME).

This parameter is returned if the *ObjectName* in the object descriptor (MQOD) (when the object was opened) is not the queue manager currently connected.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

Transmission Queue Type Error

Details of the condition generating the event are given, in the following text, in the *Reason* parameter of the event header.

Event message

When an event is generated, an event message is put on the SYSTEM.ADMIN.QMGR.EVENT queue.

The event message consists of the:

- Event header, containing a reason code parameter with a value of MQRC_XMIT_Q_TYPE_ERROR
- Event data

Event data summary

Always returned:

QMgrName, *QName*, *XmitQName*, *QType*, *ApplType*, *ApplName*

Returned optionally:

ObjectQMgrName

Event header

Reason (MQLONG)

Name of the reason code.

The value is:

MQRC_XMIT_Q_TYPE_ERROR
(2091, X'82B') Transmission queue not local.

On an MQOPEN or MQPUT1 call, a message is to be sent to a remote queue manager. The *ObjectName* or *ObjectQMgrName* field in the object descriptor specifies the name of a local definition of a remote queue but one of the following applies to the *XmitQName* attribute of the definition:

- *XmitQName* is not blank, but specifies a queue that is not a local queue
- *XmitQName* is blank, but *RemoteQMgrName* specifies a queue that is not a local queue

This reason also occurs if the queue name is resolved through a cell directory, and the remote queue manager name obtained from the cell directory is the name of a queue, but this is not a local queue.

Event data

QMgrName (MQCFST)

Name of the queue manager generating the event (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

QName (MQCFST)
Queue name from object descriptor (MQOD) (parameter identifier: MQCA_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

XmitQName (MQCFST)
Transmission queue name (parameter identifier: MQCA_XMIT_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

QType (MQCFIN)
Type of transmission queue (parameter identifier: MQIA_Q_TYPE).

The value can be:

MQQT_ALIAS
Alias queue definition.

MQQT_REMOTE
Local definition of a remote queue.

ApplType (MQCFIN)
Type of application making the MQI call that caused the event (parameter identifier: MQIA_APPL_TYPE).

ApplName (MQCFST)
Name of the current application Name of the application making the MQI call that caused the event (parameter identifier: MQCACF_APPL_NAME).

The maximum length of the string is MQ_APPL_NAME_LENGTH.

Note: If the application is a server for clients, the *ApplType* and *ApplName* parameters identify the server, rather than a client.

ObjectQMgrName (MQCFST)
Name of the object queue manager (parameter identifier: MQCACF_OBJECT_Q_MGR_NAME).

This parameter is returned if the *ObjectName* in the object descriptor (MQOD) (when the object was opened) is not the queue manager currently connected.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

Transmission Queue Usage Error

Details of the condition generating the event are given, in the following text, in the *Reason* parameter of the event header.

Event message

When an event is generated, an event message is put on the SYSTEM.ADMIN.QMGR.EVENT queue.

The event message consists of the:

- Event header, containing a reason code parameter with a value of MQRC_XMIT_Q_USAGE_ERROR
- Event data

Event data summary

Always returned:

QMgrName, *QName*, *XmitQName*, *ApplType*, *ApplName*

Returned optionally:

ObjectQMgrName

Event header

Reason (MQLONG)

Name of the reason code generating the event.

The value is:

MQRC_XMIT_Q_USAGE_ERROR

(2092, X'82C') Transmission queue with wrong usage.

On an MQOPEN or MQPUT1 call, a message is to be sent to a remote queue manager, but one of the following occurred:

- *ObjectQMgrName* specifies the name of a local queue, but it does not have a *Usage* attribute of MQUS_TRANSMISSION.
- The *ObjectName* or *ObjectQMgrName* field in the object descriptor specifies the name of a local definition of a remote queue but one of the following applies to the *XmitQName* attribute of the definition:
  - *XmitQName* is not blank, but specifies a queue that does not have a *Usage* attribute of MQUS_TRANSMISSION
  - *XmitQName* is blank, but *RemoteQMgrName* specifies a queue that does not have a *Usage* attribute of MQUS_TRANSMISSION
- The queue name is resolved through a cell directory, and the remote queue manager name obtained from the cell directory is the name of a

local queue, but it does not have a *Usage* attribute of MQUS_TRANSMISSION.

Event data

QMgrName (MQCFST)

Name of the queue manager generating the event (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

QName (MQCFST)

Queue name from object descriptor (MQOD) (parameter identifier: MQCA_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

XmitQName (MQCFST)

Transmission queue name (parameter identifier: MQCA_XMIT_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

ApplType (MQCFIN)

Type of application making the MQI call that caused the event Type of current application (parameter identifier: MQIA_APPL_TYPE).

ApplName (MQCFST)

Name of the application making the MQI call that caused the event (parameter identifier: MQCACF_APPL_NAME).

The maximum length of the string is MQ_APPL_NAME_LENGTH.

Note: If the application is a server for clients, the *ApplType* and *ApplName* parameters identify the server, rather than the client.

ObjectQMgrName (MQCFST)

Name of the object queue manager (parameter identifier: MQCACF_OBJECT_Q_MGR_NAME).

This parameter is returned if the *ObjectName* in the object descriptor (MQOD) (when the object was opened) is not the queue manager currently connected.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

Unknown Alias Base Queue

Details of the condition generating the event are given, in the following text, in the *Reason* parameter of the event header.

Event message

When an event is generated, an event message is put on the SYSTEM.ADMIN.QMGR.EVENT queue.

The event message consists of the:

- Event header, containing a reason code parameter with a value of MQRC_UNKNOWN_ALIAS_BASE_Q
- Event data

Event data summary

Always returned:

QMgrName, *QName*, *BaseQName*, *ApplType*, *ApplName*

Returned optionally:

ObjectQMgrName

Event header

Reason (MQLONG)

Name of the reason code.

The value is:

MQRC_UNKNOWN_ALIAS_BASE_Q
(2082, X'822') Unknown alias base queue.

An MQOPEN or MQPUT1 call was issued specifying an alias queue as the target, but the *BaseQName* in the alias queue attributes is not recognized as a queue name.

Event data

QMgrName (MQCFST)

The name of the queue manager generating the event (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

QName (MQCFST)

Queue name from object descriptor (MQOD) (parameter identifier: MQCA_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

BaseQName (MQCFST)

Queue name to which the alias resolves (parameter identifier: MQCA_BASE_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

ApplType (MQCFIN)
Type of the application making the MQI call that causes the event. (parameter identifier: MQIA_APPL_TYPE).

ApplName (MQCFST)
Name of the application making the MQI call that causes the event. (parameter identifier: MQCACF_APPL_NAME).

The maximum length of the string is MQ_APPL_NAME_LENGTH.

Note: If the application is a server for clients, the *ApplType* and *ApplName* parameters identify the server, rather than the client.

ObjectQMgrName (MQCFST)
Name of the object queue manager (parameter identifier: MQCACF_OBJECT_Q_MGR_NAME).

This parameter is returned if the *ObjectName* in the object descriptor (MQOD) (when the object was opened) is not the queue manager currently connected.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

Unknown Default Transmission Queue

Details of the condition generating the event are given, in the following text, in the *Reason* parameter of the Event header.

Event message

When an event is generated, an event message is put on the SYSTEM.ADMIN.QMGR.EVENT queue.

The event message consists of the:

- Event header, containing a reason code parameter with a value of MQRC_UNKNOWN_DEF_XMIT_Q
- Event data

Event data summary

Always returned:

QMgrName, *QName*, *XmitQName*, *ApplType*, *ApplName*

Returned optionally:

ObjectQMgrName

Event header

Reason (MQLONG)

Name of the reason code.

The value is:

MQRC_UNKNOWN_DEF_XMIT_Q

(2197, X'895') Unknown default transmission queue.

An MQOPEN or MQPUT1 call was issued specifying a remote queue as the destination. If a local definition of the remote queue was specified, or if a queue-manager alias is being resolved, the *XmitQName* attribute in the local definition is blank.

No queue is defined with the same name as the destination queue manager. The queue manager has therefore attempted to use the default transmission queue. However, the name defined by the *DefXmitQName* queue-manager attribute is not the name of a locally-defined queue.

Event data

QMgrName (MQCFST)

Name of the queue manager generating the event (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

QName (MQCFST)

Queue name from object descriptor (MQOD) (parameter identifier: MQCA_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

XmitQName (MQCFST)
Default transmission queue name (parameter identifier: MQCA_XMIT_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

ApplType (MQCFIN)
Type of application attempting to open the remote queue (parameter identifier: MQIA_APPL_TYPE).

ApplName (MQCFST)
Name of the application attempting to open the remote queue (parameter identifier: MQCACF_APPL_NAME).

The maximum length of the string is MQ_APPL_NAME_LENGTH.

Note: If the application is a server for clients, the *ApplType* and *ApplName* parameters identify the server, rather than a client.

ObjectQMgrName (MQCFST)
Name of the object queue manager (parameter identifier: MQCACF_OBJECT_Q_MGR_NAME).

This parameter is returned if the *ObjectName* in the object descriptor (MQOD) (when the object was opened) is not the queue manager currently connected.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

Unknown Object Name

Details of the condition generating the event are given, in the following text, in the *Reason* parameter of the event header.

Event message

When an event is generated, an event message is put on the SYSTEM.ADMIN.QMGR.EVENT queue.

The event message consists of the:

- Event header, containing a reason code parameter with a value of MQRC_UNKNOWN_OBJECT_NAME
- Event data

Event data summary

Always returned:

QMgrName, *ApplType*, *ApplName*

In addition, one of:

QName, *ProcessName*

Returned optionally:

ObjectQMgrName

Event header

Reason (MQLONG)

Name of the reason code.

The value is:

MQRC_UNKNOWN_OBJECT_NAME

(2085, X'825') Unknown object name.

On an MQOPEN or MQPUT1 call, the *ObjectQMgrName* field in the object descriptor MQOD is set to one of the following:

- Blank
- The name of the local queue manager
- The name of a local definition of a remote queue (a queue-manager alias) in which the *RemoteQMgrName* attribute is the name of the local queue manager

However, the *ObjectName* in the object descriptor is not recognized for the specified object type.

See also MQRC_Q_DELETED.

Event data

ApplType (MQCFIN)

Type of the application issuing the MQI call that caused the event (parameter identifier: MQIA_APPL_TYPE).

ApplName (MQCFST)
Name of the application issuing the MQI call that caused the event (parameter identifier: MQCACF_APPL_NAME).

The maximum length of the string is MQ_APPL_NAME_LENGTH.

Note: If the application is a server for clients, the *ApplType* and *ApplName* parameters identify the server, rather than the client.

QMgrName (MQCFST)
Name of the queue manager generating the event (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

QName (MQCFST)
Queue name from object descriptor (MQOD) (parameter identifier: MQCA_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

ProcessName (MQCFST)
Name of the process (application) issuing the MQI call that caused the event (parameter identifier: MQCA_PROCESS_NAME).

The maximum length of the string is MQ_PROCESS_NAME_LENGTH.

ObjectQMgrName (MQCFST)
Name of the object queue manager (parameter identifier: MQCACF_OBJECT_Q_MGR_NAME).

This parameter is returned if the *ObjectName* in the object descriptor (MQOD) (when the object was opened) is not the queue manager currently connected.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

Unknown Remote Queue Manager

Details of the condition generating the event are given, in the following text, in the *Reason* parameter of the event header.

Event message

When an event is generated, an event message is put on the SYSTEM.ADMIN.QMGR.EVENT queue.

The event message consists of the:

- Event header, containing a reason code parameter with a value of MQRC_UNKNOWN_REMOTE_Q_MGR
- Event data

Event data summary

Always returned:

QMgrName, *QName*, *ApplType*, *ApplName*

Returned optionally:

ObjectQMgrName

Event header

Reason (MQLONG)

Name of the reason code.

The value is:

MQRC_UNKNOWN_REMOTE_Q_MGR
(2087, X'827') Unknown remote queue manager.

On an MQOPEN or MQPUT1 call, an error occurred with the queue-name resolution, for one of the following reasons:

- *ObjectQMgrName* is either blank or the name of the local queue manager, and *ObjectName* is the name of a local definition of a remote queue, which has a blank *XmitQName*. However, there is no (transmission) queue defined with the name of *RemoteQMgrName*, and the *DefXmitQName* queue-manager attribute is blank.
- *ObjectQMgrName* is the name of a queue-manager alias definition (held as the local definition of a remote queue), which has a blank *XmitQName*. However, there is no (transmission) queue defined with the name of *RemoteQMgrName*, and the *DefXmitQName* queue-manager attribute is blank.
- *ObjectQMgrName* specified is not:
  - Blank
  - The name of the local queue manager
  - The name of a local queue

  - The name of a queue-manager alias definition (that is, a local definition of a remote queue with a blank *RemoteQName*)

  and the *DefXmitQName* queue-manager attribute is blank.

- *ObjectQMgrName* is blank or is the name of the local queue manager, and *ObjectName* is the name of a local definition of a remote queue (or an alias to one), for which *RemoteQMgrName* is either blank or is the name of the local queue manager. Note that this error occurs even if the *XmitQName* is not blank.
- *ObjectQMgrName* is the name of a local definition of a remote queue. In this context, this should be a queue-manager alias definition, but the *RemoteQName* in the definition is not blank.
- *ObjectQMgrName* is the name of a model queue.
- The queue name is resolved through a cell directory. However, there is no queue defined with the same name as the remote queue manager name obtained from the cell directory. Also, the *DefXmitQName* queue-manager attribute is blank.

Event data

QMgrName (MQCFST)

Name of the queue manager generating the event (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

QName (MQCFST)

Queue name from *object descriptor (MQOD)* (parameter identifier: MQCA_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

ApplType (MQCFIN)

Type of application attempting to open the remote queue (parameter identifier: MQIA_APPL_TYPE).

ApplName (MQCFST)

Name of the application attempting to open the remote queue (parameter identifier: MQCACF_APPL_NAME).

The maximum length of the string is MQ_APPL_NAME_LENGTH.

Note: If the application is a server for clients, the *ApplType* and *ApplName* parameters identify the server, rather than the client.

ObjectQMgrName (MQCFST)

Name of the object queue manager (parameter identifier: MQCACF_OBJECT_Q_MGR_NAME).

This parameter is returned if the *ObjectName* in the object descriptor (MQOD) (when the object was opened) is not the queue manager currently connected.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

Unknown Transmission Queue

Details of the condition generating the event are given, in the following text, in the *Reason* parameter of the event header.

Event message

When an event is generated, an event message is put on the SYSTEM.ADMIN.QMGR.EVENT queue.

The event message consists of the:

- Event header, containing a reason code parameter with a value of MQRC_UNKNOWN_XMIT_Q
- Event data

Event data summary

Always returned:

QMgrName, *QName*, *XmitQName*, *ApplType*, *ApplName*

Returned optionally:

ObjectQMgrName

Event header

Reason (MQLONG)

Name of the reason code.

The value is:

MQRC_UNKNOWN_XMIT_Q
(2196, X'894') Unknown transmission queue.

On an MQOPEN or MQPUT1 call, a message is to be sent to a remote queue manager. The *ObjectName* or the *ObjectQMgrName* in the object descriptor specifies the name of a local definition of a remote queue (in the latter case queue-manager aliasing is being used), but the *XmitQName* attribute of the definition is not blank and not the name of a locally-defined queue.

Event data

QMgrName (MQCFST)

Name of the queue manager generating the event (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

QName (MQCFST)

Queue name from object descriptor (MQOD) (parameter identifier: MQCA_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

XmitQName (MQCFST)

Transmission queue name (parameter identifier: MQCA_XMIT_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

ApplType (MQCFIN)
Type of application that made the MQI call (parameter identifier: MQIA_APPL_TYPE).

ApplName (MQCFST)
Name of the current application (parameter identifier: MQCACF_APPL_NAME).

The maximum length of the string is MQ_APPL_NAME_LENGTH.

Note: If the application is a server for clients, the *ApplType* and *ApplName* parameters identify the server, rather than the client.

ObjectQMgrName (MQCFST)
Name of the object queue manager (parameter identifier: MQCACF_OBJECT_Q_MGR_NAME).

This parameter is returned if the *ObjectName* in the object descriptor (MQOD) (when the object was opened) is not the queue manager currently connected.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

Chapter 5. Example of using instrumentation events

This example shows how you can write a program for instrumentation events. It is written in C for queue managers on OS/2, Windows NT, or UNIX systems. It is not part of any MQSeries product and is therefore supplied as source only. The example is incomplete in that it does not enumerate all the possible outcomes of specified actions. Bearing this in mind, you can use this sample as a basis for your own programs that use events, in particular, the PCF formats used in event messages. However, you will need to modify this program to get it to run on your systems.

```
/********************************************************************/
/*                                                                  */
/* Program name: EVMON                                              */
/*                                                                  */
/* Description: C program that acts as an event monitor             */
/*                                                                  */
/*                                                                  */
/********************************************************************/
/*                                                                  */
/* Function:                                                        */
/*                                                                  */
/*                                                                  */
/*   EVMON is a C program that acts as an event monitor - reads an  */
/*   event queue and tells you if anything appears on it            */
/*                                                                  */
/*   Its first parameter is the queue manager name, the second is   */
/*   the event queue name. If these are not supplied it uses the    */
/*   defaults.                                                      */
/*                                                                  */
/********************************************************************/
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#ifndef min
  #define min(a,b)        (((a) < (b)) ? (a) : (b))
#endif
#ifdef OS2
  /******************************************************************/
  /* for beep                                                       */
  /******************************************************************/
  #define INCL_DOSPROCESS
  #include <os2.h>
#endif
/********************************************************************/
/* includes for MQI                                                 */
/********************************************************************/
#include <cmqc.h>
#include <cmqcfc.h>
void printfmqcfst(MQCFST* pmqcfst);
void printfmqcfin(MQCFIN* pmqcfst);
void printreas(MQLONG reason);

 #define PRINTREAS(param)                                          \
    case param:                                                    \
```

```
      printf("Reason = %s\n",#param);                              \
      break;

/*****************************************************************/
/* global variable                                               */
/*****************************************************************/
MQCFH    *evtmsg;                    /* evtmsg message buffer      */

int main(int argc, char **argv)
{
  /***************************************************************/
  /* declare variables                                           */
  /***************************************************************/
  int  i;                             /* auxiliary counter       */
  /***************************************************************/
  /* Declare MQI structures needed                               */
  /***************************************************************/
  MQOD     od = {MQOD_DEFAULT};      /* Object Descriptor          */
  MQMD     md = {MQMD_DEFAULT};      /* Message Descriptor         */
  MQGMO   gmo = {MQGMO_DEFAULT};     /* get message options        */
  /***************************************************************/
  /* note, uses defaults where it can                            */
  /***************************************************************/
  MQHCONN  Hcon;                     /* connection handle          */
  MQHOBJ   Hobj;                     /* object handle              */
  MQLONG   O_options;                /* MQOPEN options             */
  MQLONG   C_options;                /* MQCLOSE options            */
  MQLONG   CompCode;                 /* completion code            */
  MQLONG   OpenCode;                 /* MQOPEN completion code     */
  MQLONG   Reason;                   /* reason code                */
  MQLONG   CReason;                  /* reason code for MQCONN     */
  MQLONG   buflen;                   /* buffer length              */
  MQLONG   evtmsglen;                /* message length received    */
  MQCHAR   command[1100];            /* call command string ...    */
  MQCHAR   p1[600];                  /* ApplId insert              */
  MQCHAR   p2[900];                  /* evtmsg insert              */
  MQCHAR   p3[600];                  /* Environment insert         */
  MQLONG   mytype;                   /* saved application type     */
  char     QMName[50];               /* queue manager name         */
  MQCFST  *paras;                    /* the parameters             */
  int      counter;                  /* loop counter               */
  time_t   ltime;

  /***************************************************************/
  /* Connect to queue manager                                    */
  /***************************************************************/
  QMName[0] = 0;                      /* default queue manager     */
  if (argc > 1)
    strcpy(QMName, argv[1]);
  MQCONN(QMName,                      /* queue manager             */
         &Hcon,                   /* connection handle         */
         &CompCode,               /* completion code           */
         &CReason);               /* reason code               */

  /***************************************************************/
  /* Initialize object descriptor for subject queue              */
  /***************************************************************/
  strcpy(od.ObjectName, "SYSTEM.ADMIN.QMGR.EVENT");
```

```
if (argc > 2)
  strcpy(od.ObjectName, argv[2]);

/*****************************************************************/
/* Open the event queue for input; exclusive or shared.  Use of  */
/* the queue is controlled by the queue definition here          */
/*****************************************************************/
O_options = MQOO_INPUT_AS_Q_DEF     /* open queue for input      */
      + MQOO_FAIL_IF_QUIESCING      /* but not if qmgr stopping  */
      + MQOO_BROWSE;
MQOPEN(Hcon,                        /* connection handle         */
       &od,                     /* object descriptor for queue*/
       O_options,                   /* open options              */
       &Hobj,                   /* object handle             */
       &CompCode,               /* completion code           */
       &Reason);                /* reason code               */

/*****************************************************************/
/*   Get messages from the message queue                         */
/*****************************************************************/
while (CompCode != MQCC_FAILED)
{
  /***************************************************************/
  /* I don't know how big this message is so just get the        */
  /* descriptor first                                            */
  /***************************************************************/
  gmo.Options = MQGMO_WAIT + MQGMO_LOCK
     + MQGMO_BROWSE_FIRST + MQGMO_ACCEPT_TRUNCATED_MSG;
                                    /* wait for new messages     */
  gmo.WaitInterval = MQWI_UNLIMITED;/* no time limit             */
  buflen = 0;                       /* amount of message to get  */

  /***************************************************************/
  /* clear selectors to get messages in sequence                 */
  /***************************************************************/
  memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
  memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));

  /***************************************************************/
  /* wait for event message                                      */
  /***************************************************************/
  printf("...>\n");
  MQGET(Hcon,                       /* connection handle         */
        Hobj,                       /* object handle             */
        &md,                    /* message descriptor        */
        &gmo,                   /* get message options       */
        buflen,                     /* buffer length             */
        evtmsg,                     /* evtmsg message buffer     */
        &evtmsglen,             /* message length            */
        &CompCode,              /* completion code           */
        &Reason);               /* reason code               */

  /***************************************************************/
  /* report reason, if any                                       */
  /***************************************************************/
  if (Reason != MQRC_NONE && Reason != MQRC_TRUNCATED_MSG_ACCEPTED)
  {
    printf("MQGET ==> %ld\n", Reason);
```

```
}
else
{
  gmo.Options = MQGMO_NO_WAIT + MQGMO_MSG_UNDER_CURSOR;
  buflen = evtmsglen;             /* amount of message to get   */
  evtmsg = malloc(buflen);
  if (evtmsg != NULL)
  {
    /*************************************************************/
    /* clear selectors to get messages in sequence               */
    /*************************************************************/
    memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
    memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));

    /*************************************************************/
    /* get the event message                                     */
    /*************************************************************/
    printf("...>\n");
    MQGET(Hcon,                     /* connection handle        */
          Hobj,                     /* object handle            */
          &md,                 /* message descriptor       */
          &gmo,                /* get message options      */
          buflen,                   /* buffer length            */
          evtmsg,                   /* evtmsg message buffer    */
          &evtmsglen,          /* message length           */
          &CompCode,           /* completion code          */
          &Reason);            /* reason code              */

    /*************************************************************/
    /* report reason, if any                                     */
    /*************************************************************/
    if (Reason != MQRC_NONE)
    {
      printf("MQGET ==> %ld\n", Reason);
    }
  }
  else
  {
    CompCode = MQCC_FAILED;
  }
}
/*****************************************************************/
/* . . . process each message received                           */
/*****************************************************************/
if (CompCode != MQCC_FAILED)
{
  /*************************************************************/
  /* announce a message                                        */
  /*************************************************************/
  #ifdef OS2
    {
      unsigned short tone;
      for (tone = 1; tone < 8000; tone = tone * 2)
      {
        DosBeep(tone,50);
      }
    }
  #else
```

```
  printf("\a\a\a\a\a\a\a");
#endif
time(&ltime);
printf(ctime(&ltime));

if (evtmsglen != buflen)
  printf("DataLength = %ld?\n", evtmsglen);
else
{
  /*************************************************************/
  /* right let's look at the data                              */
  /*************************************************************/
  if (evtmsg->Type != MQCFT_EVENT)
  {
    printf("Something's wrong this isn't an event message,"
           " its type is %ld\n",evtmsg->Type);
  }
  else
  {
    if (evtmsg->Command == MQCMD_Q_MGR_EVENT)
    {
      printf("Queue Manager event: ");
    }
    else
      if (evtmsg->Command == MQCMD_CHANNEL_EVENT)
      {
        printf("Channel event: ");
      }
      else

⋮

      {
        printf("Unknown Event message, %ld.",
                evtmsg->Command);
      }

    if      (evtmsg->CompCode == MQCC_OK)
      printf("CompCode(OK)\n");
    else if (evtmsg->CompCode == MQCC_WARNING)
      printf("CompCode(WARNING)\n");
    else if (evtmsg->CompCode == MQCC_FAILED)
      printf("CompCode(FAILED)\n");
    else
      printf("* CompCode wrong * (%ld)\n",
                evtmsg->CompCode);

    if (evtmsg->StrucLength != MQCFH_STRUC_LENGTH)
    {
      printf("it's the wrong length, %ld\n",evtmsg->StrucLength);
    }

    if (evtmsg->Version != MQCFH_VERSION_1)
    {
      printf("it's the wrong version, %ld\n",evtmsg->Version);
    }

    if (evtmsg->MsgSeqNumber != 1)
    {
```

```
            printf("it's the wrong sequence number, %ld\n",
                    evtmsg->MsgSeqNumber);
          }

          if (evtmsg->Control != MQCFC_LAST)
          {
            printf("it's the wrong control option, %ld\n",
                    evtmsg->Control);
          }

          printreas(evtmsg->Reason);
          printf("parameter count is %ld\n", evtmsg->ParameterCount);
          /*********************************************************/
          /* get a pointer to the start of the parameters          */
          /*********************************************************/
          paras = (MQCFST *)(evtmsg + 1);
          counter = 1;
          while (counter <= evtmsg->ParameterCount)
          {
            switch (paras->Type)
            {
              case MQCFT_STRING:
                printfmqcfst(paras);
                paras = (MQCFST *)((char *)paras
                                   + paras->StrucLength);
                break;
              case MQCFT_INTEGER:
                printfmqcfin((MQCFIN*)paras);
                paras = (MQCFST *)((char *)paras
                                   + paras->StrucLength);
                break;
              default:
                printf("unknown parameter type, %ld\n",
                       paras->Type);
                counter = evtmsg->ParameterCount;
                break;
            }
            counter++;
          }
        }
      }   /* end evtmsg action             */
      free(evtmsg);
    }     /* end process for successful GET */
  }       /* end message processing loop    */

  /*****************************************************************/
  /* close the event queue - if it was opened                      */
  /*****************************************************************/
  if (OpenCode != MQCC_FAILED)
  {
    C_options = 0;                  /* no close options            */
    MQCLOSE(Hcon,                   /* connection handle           */
            &Hobj,              /* object handle               */
            C_options,
            &CompCode,          /* completion code             */
            &Reason);           /* reason code                 */
  /*****************************************************************/
  /* Disconnect from queue manager (unless previously connected)   */
```

```
   /*****************************************************************/
   if (CReason != MQRC_ALREADY_CONNECTED)
   {
     MQDISC(&Hcon,               /* connection handle           */
            &CompCode,           /* completion code             */
            &Reason);            /* reason code                 */
 /*****************************************************************/
 /*                                                               */
 /* END OF EVMON                                                  */
 /*                                                               */
 /*****************************************************************/
 }

#define PRINTPARAM(param)                                        \
   case param:                                                   \
     {                                                           \
       char *p = #param;                                         \
     strncpy(thestring,pmqcfst->String,min(sizeof(thestring),    \
             pmqcfst->StringLength));                            \
     printf("%s %s\n",p,thestring);                              \
     }                                                           \
     break;

#define PRINTAT(param)                                           \
   case param:                                                   \
     printf("MQIA_APPL_TYPE = %s\n",#param);                     \
     break;


void printfmqcfst(MQCFST* pmqcfst)
{
  char thestring[100];

  switch (pmqcfst->Parameter)
  {
    PRINTPARAM(MQCA_BASE_Q_NAME)
    PRINTPARAM(MQCA_PROCESS_NAME)
    PRINTPARAM(MQCA_Q_MGR_NAME)
    PRINTPARAM(MQCA_Q_NAME)
    PRINTPARAM(MQCA_XMIT_Q_NAME)
    PRINTPARAM(MQCACF_APPL_NAME)

  ⋮

    default:
      printf("Invalid parameter, %ld\n",pmqcfst->Parameter);
      break;
  }
}

void printfmqcfin(MQCFIN* pmqcfst)
{
  switch (pmqcfst->Parameter)
  {
    case MQIA_APPL_TYPE:
      switch (pmqcfst->Value)
      {
        PRINTAT(MQAT_UNKNOWN)
```

```
        PRINTAT(MQAT_OS2)
        PRINTAT(MQAT_DOS)
        PRINTAT(MQAT_UNIX)
        PRINTAT(MQAT_QMGR)
        PRINTAT(MQAT_OS400)
        PRINTAT(MQAT_WINDOWS)
        PRINTAT(MQAT_CICS_VSE)
        PRINTAT(MQAT_VMS)
        PRINTAT(MQAT_GUARDIAN)
        PRINTAT(MQAT_VOS)
      }
      break;
    case MQIA_Q_TYPE:
      if (pmqcfst->Value == MQQT_ALIAS)
      {
        printf("MQIA_Q_TYPE is MQQT_ALIAS\n");
      }
      else

⋮
      {
        if (pmqcfst->Value == MQQT_REMOTE)
        {
          printf("MQIA_Q_TYPE is MQQT_REMOTE\n");
          if (evtmsg->Reason == MQRC_ALIAS_BASE_Q_TYPE_ERROR)
          {
            printf("but remote is not valid here\n");
          }
        }
        else
        {
          printf("MQIA_Q_TYPE is wrong, %ld\n",pmqcfst->Value);
        }
      }
      break;
    case MQIACF_REASON_QUALIFIER:
      printf("MQIACF_REASON_QUALIFIER %ld\n",pmqcfst->Value);
      break;

    case MQIACF_ERROR_IDENTIFIER:
      printf("MQIACF_ERROR_INDENTIFIER %ld (X'%lX')\n",
              pmqcfst->Value,pmqcfst->Value);
      break;

    case MQIACF_AUX_ERROR_DATA_INT_1:
      printf("MQIACF_AUX_ERROR_DATA_INT_1 %ld (X'%lX')\n",
              pmqcfst->Value,pmqcfst->Value);
      break;

    case MQIACF_AUX_ERROR_DATA_INT_2:
      printf("MQIACF_AUX_ERROR_DATA_INT_2 %ld (X'%lX')\n",
              pmqcfst->Value,pmqcfst->Value);
      break;
⋮
    default :
      printf("Invalid parameter, %ld\n",pmqcfst->Parameter);
      break;
```

```
  }
}

void printreas(MQLONG reason)
{
  switch (reason)
  {
    PRINTREAS(MQRCCF_CFH_TYPE_ERROR)
    PRINTREAS(MQRCCF_CFH_LENGTH_ERROR)
    PRINTREAS(MQRCCF_CFH_VERSION_ERROR)
    PRINTREAS(MQRCCF_CFH_MSG_SEQ_NUMBER_ERR)

  .
  .
  .
    PRINTREAS(MQRC_NO_MSG_LOCKED)
    PRINTREAS(MQRC_CONNECTION_NOT_AUTHORIZED)
    PRINTREAS(MQRC_MSG_TOO_BIG_FOR_CHANNEL)
    PRINTREAS(MQRC_CALL_IN_PROGRESS)
    default:
      printf("It's an unknown reason, %ld\n",
              reason);
      break;
  }
}
```

Example using events

Part 2. Programmable Command Formats

Chapter 6. Introduction to Programmable Command Formats

This chapter introduces MQSeries Programmable Command Formats (PCFs) and their relationship to other parts of the MQSeries products.

The Programmable Command Formats described in this book are supported by:

| MQSeries for AIX Version 5
| MQSeries for AS/400 Version 4 Release 2
| MQSeries for AT&T GIS UNIX Version 2 Release 2
| MQSeries for Digital OpenVMS Version 2 Release 2
| MQSeries for HP-UX Version 5
| MQSeries for OS/2 Warp Version 5
| MQSeries for SINIX and DC/OSx Version 2 Release 2
| MQSeries for SunOS Version 2 Release 2
| MQSeries for Sun Solaris Version 5
| MQSeries for Tandem NonStop Kernel Version 2 Release 2
| MQSeries for Windows NT Version 5
| MQSeries for Windows Version 2 Release 1

Event messages also use the Programmable Command Formats. See Chapter 1, “Using instrumentation events to monitor queue managers” on page 3.

The problem PCF commands solve

The administration of distributed networks can become very complex. The problems of administration will continue to grow as networks increase in size and complexity.

Examples of administration specific to messaging and queuing include:

- Resource management.

  For example, queue creation and deletion.

- Performance monitoring.

  For example, maximum queue depth or message rate.

- Control.

  For example, tuning queue parameters such as maximum queue depth, maximum message length, and enabling and disabling queues.

- Message routing.

  Definition of alternative routes through a network.

MQSeries PCF commands can be used to simplify queue manager administration and other network administration. PCF commands allow you to use a single application to perform network administration from a single queue manager within the network.

What PCFs are

PCFs define command and reply messages that can be exchanged between a program and any queue manager (that supports PCFs) in a network. You can use PCF commands in a systems management application program for administration of MQSeries objects: queue managers, process definitions, queues, and channels. The application can operate from a single point in the network to communicate command and reply information with any queue manager, local or remote, via the local queue manager.

Each queue manager has an administration queue with a standard queue name and your application can send PCF command messages to that queue. Each queue manager also has a command server to service the command messages from the administration queue. PCF command messages can therefore be processed by any queue manager in the network and the reply data can be returned to your application, using your specified reply queue. PCF commands and reply messages are sent and received using the normal Message Queue interface (MQI).

Other programmable administration

Administration of MQSeries objects may be carried out in other ways.

MQSeries for AS/400

As well as PCFs, there are two other methods available:

OS/400 Control Language (CL)

This can be used to issue administration commands to MQSeries for AS/400 They can be issued either at the command line or by writing a CL program. These commands perform similar functions to PCF commands, but the format is completely different. CL commands are designed exclusively for OS/400 and CL responses are designed to be human-readable, whereas PCF commands are platform independent and both command and response formats are intended for program use.

MQSeries Commands (MQSC)

These provide a uniform method of issuing commands across MQSeries platforms. The general format of the commands is shown in the *MQSeries Command Reference*.

To issue the commands on OS/400 you create a list of commands in a Script file, and then run the file using the STRMQMMQSC command.

MQSC responses are designed to be human readable, whereas PCF command and response formats are intended for program use.

MQSeries for MVS/ESA

MQSeries for MVS/ESA supports the MQSeries commands (MQSC). With MVS/ESA these commands can be entered from the MVS console, or sent to the system command input queue. More information about issuing the commands is given in the *MQSeries Command Reference*, and the *MQSeries for MVS/ESA System Management Guide*.

PCF commands are not supported by MQSeries for MVS/ESA.

MQSeries for Windows

MQSeries for Windows supports the MQSeries commands (MQSC). You can enter these commands in a window provided by the MQSC utility, and also run MQSC command files.

MQSeries for Windows NT, OS/2, Digital OpenVMS and UNIX systems

As well as PCFs, there are two other methods available:

MQSeries commands (MQSC)

You can use the MQSC as single commands issued at the OS/2, Windows NT, or UNIX system command line. To issue more complicated, or multiple commands, the MQSC can be built into a file that you then run from the OS/2, Windows NT, or UNIX system command line. MQSC can be sent to a remote queue manager. For full details see the *MQSeries Command Reference*.

Control commands

MQSeries for OS/2, Windows NT, and UNIX systems provides another type of command for some of the functions. These are the *control commands* that you issue at the OS/2, Windows NT, or UNIX system command line. Reference material for these commands is contained in the *MQSeries System Administration* book.

MQSeries for Tandem NSK

As well as PCFs, there are three other methods available:

- MQSeries commands (MQSC)
- Control commands
- Message Queue Management (MQM) facility.

  MQSeries for Tandem NSK provides a panel interface for some of the functions. For full details see the *MQSeries for Tandem NonStop Kernel System Management Guide*.

Chapter 7. Using Programmable Command Formats

This chapter describes how to use the PCFs in a systems management application program for MQSeries remote administration.

PCF command messages

Each command and its parameters are sent as a separate command message containing a PCF header followed by a number of parameter structures (see "MQCFH – PCF header" on page 334). The PCF header identifies the command and the number of parameter structures that follow in the same message. Each parameter structure provides a parameter to the command.

Replies to the commands, generated by the command server, have a similar structure. There is a PCF header, followed by a number of parameter structures. Replies can consist of more than one message but commands always consist of one message only.

The queue to which the PCF commands are sent is always called the SYSTEM.ADMIN.COMMAND.QUEUE. The command server servicing this queue sends the replies to the queue defined by the *ReplyToQ* and *ReplyToQMgr* fields in the message descriptor of the command message.

How to issue PCF command messages

Use the normal Message Queue Interface (MQI) calls, MQPUT, MQGET and so on, to put and retrieve PCF command and response messages to and from their respective queues.

Note to users

You must start the command server on the target queue manager for the PCF command to process on that queue manager.

For a list of supplied header files, see Appendix C, "Header, COPY, and INCLUDE files" on page 485.

Message descriptor for a PCF command

The MQSeries message descriptor is fully documented in the *MQSeries Application Programming Reference*.

A PCF command message contains the following fields in the message descriptor:

Report
Any valid value, as required.

MsgType
This must be MQMT_REQUEST to indicate a message requiring a response.

Expiry
Any valid value, as required.

Feedback
Set to MQFB_NONE

Encoding
If you are sending to MQSeries for OS/400 V3R2 (or later), OS/2, Windows NT, or UNIX systems set this field to the encoding used for the message data; conversion will be performed if necessary.

CodedCharSetId
If you are sending to MQSeries for OS/400 V3R2 (or later), OS/2, Windows NT, or UNIX systems set this field to the coded character-set identifer used for the message data; conversion will be performed if necessary.

Format
Set to MQFMT_ADMIN.

Priority
Any valid value, as required.

Persistence
Any valid value, as required.

MsgId
The sending application may specify any value, or MQMI_NONE can be specified to request the queue manager to generate a unique message identifier.

CorrelId
The sending application may specify any value, or MQMI_NONE can be specified to indicate no correlation identifier.

ReplyToQ
The name of the queue to receive the response.

ReplyToQMgr
The name of the queue manager for the response (or blank).

Message context fields
These can be set to any valid values, as required. Normally the Put message option MQPMO_DEFAULT_CONTEXT is used to set the message context fields to the default values.

If you are using a version-2 MQMD structure, you must set the following additional fields:

GroupId
Set to MQGI_NONE

MsgSeqNumber
Set to 1

Offset
Set to 0

MsgFlags
Set to MQMF_NONE

OriginalLength
Set to MQOL_UNDEFINED

Sending user data

The PCF structures can also be used to send user-defined message data. In this case the message descriptor *Format* field should be set to MQFMT_PCF.

Responses

In response to each command, the command server generates one or more response messages. A response message has a similar format to a command message; the PCF header has the same command identifier value as the command to which it is a response (see "MQCFH – PCF header" on page 334 for details). The message identifier and correlation identifier are set according to the report options of the request.

If a single command specifies a generic object name, a separate response is returned in its own message for each matching object. For the purpose of response generation, a single command with a generic name is treated as multiple individual commands (except for the control field MQCFC_LAST or MQCFC_NOT_LAST). Otherwise, one command message generates one response message.

Certain PCF responses may return a structure even when it is not requested. This is shown in the definition of the response (Chapter 8) as *always returned*. The reason for this is that, for these responses, it is necessary to name the objects in the response so that one can know to which object the data applies.

There are three types of response, described below:

- OK response
- Error response
- Data response

OK response

This consists of a message starting with a command format header, with a *CompCode* field of MQCC_OK or MQCC_WARNING.

For MQCC_OK, the *Reason* is MQRC_NONE.

For MQCC_WARNING, the *Reason* identifies the nature of the warning. In this case the command format header may be followed by one or more warning parameter structures appropriate to this reason code.

In either case, for an inquire command further parameter structures may follow as described below.

Error response

If the command has an error, one or more error response messages are sent (more than one may be sent even for a command which would normally only have a single response message). These error response messages have MQCFC_LAST or MQCFC_NOT_LAST set as appropriate.

Each such message starts with a response format header, with a *CompCode* value of MQCC_FAILED and a *Reason* field which identifies the particular error. In general each message describes a different error. In addition, each message has either

zero or one (never more than one) error parameter structures following the header. This parameter structure, if there is one, is an MQCFIN structure, with a *Parameter* field containing one of the following:

- MQIACF_PARAMETER_ID

  The *Value* field in the structure is the parameter identifier of the parameter that was in error (for example, MQCA_Q_NAME).

- MQIACF_ERROR_ID

  This is used with a *Reason* value (in the command format header) of MQRC_UNEXPECTED_ERROR. The *Value* field in the MQCFIN structure is the unexpected reason code received by the command server.

- MQIACF_SELECTOR

  This occurs if a list structure (MQCFIL) sent with the command contains an invalid or duplicate selector. The *Reason* field in the command format header identifies the error, and the *Value* field in the MQCFIN structure is the parameter value in the MQCFIL structure of the command that was in error.

- MQIACF_ERROR_OFFSET

  This occurs when there is a data compare error on the Ping Channel command. The *Value* field in the structure is the offset of the Ping Channel compare error.

- MQIA_CODED_CHAR_SET_ID

  This occurs when the coded character-set identifier in the message descriptor of the incoming PCF command message does not match that of the target queue manager. The *Value* field in the structure is the coded character-set identifier of the queue manager.

The last (or only) error response message is a summary response, with a *CompCode* field of MQCC_FAILED, and a *Reason* field of MQRCCF_COMMAND_FAILED. This message has no parameter structure following the header.

Data Response

This consists of an OK response (as described above) to an inquire command. The OK response is followed by additional structures containing the requested data as described in Chapter 8, “Definitions of the Programmable Command Formats” on page 135.

Applications should not depend upon these additional parameter structures being returned in any particular order.

Message descriptor for a response

A response message (obtained using the Get-message option MQGMO_CONVERT) has the following fields in the message descriptor, defined by the putter of the message. The actual values in the fields are generated by the queue manager:

MsgType
This is MQMT_REPLY.

MsgId
This is generated by the queue manager.

CorrelId
This is generated according to the report options of the command message.

Format
This is MQFMT_ADMIN.

Encoding
Set to MQENC_NATIVE.

CodedCharSetId
Set to MQCCSI_Q_MGR.

Persistence
The same as in the command message.

Priority
The same as in the command message.

The response is generated with MQPMO_PASS_IDENTITY_CONTEXT.

Authority checking for PCF commands

When a PCF command is processed, the *UserIdentifier* from the message descriptor in the command message is used for the required MQSeries object authority checks. The checks are performed on the system on which the command is being processed, therefore this user ID must exist on the target system and have the required authorities to process the command. If the message has come from a remote system, one way of achieving this is to have a matching user ID on both the local and remote systems.

Authority checking is implemented differently on each platform.

MQseries for AS/400

In order to process any PCF command, the user ID must have *READ authority for the MQSeries object on the target system.

In addition, MQSeries object authority checks are performed for certain PCF commands, as shown in Table 16 on page 132. In most cases these are the same checks as those performed by the equivalent MQSeries CL commands issued on a local system. See the *MQSeries for AS/400 Administration Guide* for more information on the mapping from MQSeries authorities to OS/400 system authorities, and the authority requirements for the MQSeries CL commands. Details of security concerning exits are given in the *MQSeries Intercommunication* book.

To process any of the following commands the user ID must have *ALLOBJ authority, or the user ID must be QPGMR or QSYSOPR:

- Ping Channel
- Change Channel
- Copy Channel
- Create Channel
- Delete Channel
- Reset Channel
- Resolve Channel

- Start Channel
- Stop Channel
- Start Channel Initiator
- Start Channel Listener

Table 16. MQSeries for AS/400 - object authorities

Command	MQSeries object authority	*CTLG authority
Change Queue	*READ and *UPD	n/a
Change Queue Manager	*READ and *UPD	n/a
Change Process	*READ and *UPD	n/a
Clear Queue	*READ and *DLT	n/a
Copy Process	*from:* *READ	*ADD
Copy Process (Replace)	*from:* *READ *to:* *OBJOPR and *UPD	n/a
Copy Queue	*from:* *READ	*ADD
Copy Queue (Replace)	*from:* *READ *to:* *OBJOPR and *UPD	n/a
Create Process	*(system default process)* *READ	*ADD
Create Process (Replace)	*(system default process)* *READ *to:* *OBJOPR and *UPD	n/a
Create Queue	*(system default queue)* *READ	*ADD
Create Queue (Replace)	*(system default queue)* *READ *to:* *OBJOPR and *UPD	n/a
Delete Process	*OBJEXIST	*DLT
Delete Queue	*OBJEXIST	*DLT
Inquire Queue	*READ	n/a
Inquire Queue Manager	*READ	n/a
Inquire Process	*READ	n/a
Reset Queue Statistics	*UPD	n/a
Escape	*see Note*	*see Note*

Note: The required authority is determined by the MQSC command defined by the escape text, and it will be equivalent to one of the above.

MQSeries for OS/2 Warp

If there is no authorization service installed, or if the PCF command is a channel command, OS/2 performs no additional security checking other than making sure that the *UserIdentifier* of the message descriptor is not set to blanks. If there is an installed authorization service, this controls access to the queue manager, queue, and process objects, with access to channels unaffected.

MQSeries also has some channel security exit points so that you can supply your own user exit programs for security checking. Details are given in the *MQSeries Intercommunication* book.

MQSeries for Windows NT, Digital OpenVMS, Tandem NSK, and UNIX systems

In order to process any PCF command, the user ID must have *dsp* authority for the queue manager object on the target system. In addition, MQSeries object authority checks are performed for certain PCF commands, as shown in Table 17.

To process any of the following commands the user ID must belong to group *mqm*, or for Windows NT **only** the user ID must belong to group *Administrators*:

- Change Channel
- Copy Channel
- Create Channel
- Delete Channel
- Ping Channel
- Reset Channel
- Start Channel
- Stop Channel
- Start Channel Initiator
- Start Channel Listener
- Resolve Channel

Table 17 (Page 1 of 2). MQSeries for Windows NT, Digital OpenVMS, Tandem NSK, and UNIX systems - object authorities

Command	MQSeries object authority	Class authority (for object type)
Change Queue	chg	n/a
Change Queue Manager	chg	n/a
Change Process	chg	n/a
Clear Queue	clr	n/a
Copy Process	*from:* dsp	crt
Copy Process (Replace) *see Note 1*	*from:* dsp *to:* chg	n/a
Copy Queue	*from:* dsp	crt
Copy Queue (Replace) *see Note 1*	*from:* dsp *to:* chg	n/a
Create Process	*(system default process)* dsp	crt
Create Process (Replace) *see Note 1*	*(system default process)* dsp *to:* chg	n/a
Create Queue	*(system default queue)* dsp	crt
Create Queue (Replace) *see Note 1*	*(system default queue)* dsp *to:* n/a	crt
Delete Process	dlt	n/a
Delete Queue	dlt	n/a
Inquire Queue	dsp	n/a
Inquire Queue Manager	dsp	n/a
Inquire Process	dsp	n/a
Reset Queue Statistics	dsp and chg	n/a

Table 17 (Page 2 of 2). MQSeries for Windows NT, Digital OpenVMS, Tandem NSK, and UNIX systems - object authorities

Command	MQSeries object authority	Class authority (for object type)
Escape	*see Note 2*	*see Note 2*

Notes:

1. This applies if the object to be replaced does already exist, otherwise the authority check is as for Create without Replace.
2. The required authority is determined by the MQSC command defined by the escape text, and it will be equivalent to one of the above.

MQSeries also supplies some channel security exit points so that you can supply your own user exit programs for security checking. Details are given in the *MQSeries Intercommunication* book.

Chapter 8. Definitions of the Programmable Command Formats

This chapter contains reference material for the Programmable Command Formats (PCFs) of commands and responses sent between an MQSeries systems management application program and an MQSeries queue manager.

How the definitions are shown

For each PCF command or response there is a description of what the command or response does, giving the command identifier in parentheses. See "MQCFH – PCF header" on page 334 for details of the command identifier.

Notes to users

1. The PCFs described in the reference section are available on all platforms **except MVS/ESA**, unless specific limitations are shown at the start of a structure.
2. MQSeries for Windows V2.0 does not support PCFs.
3. You cannot use PCF commands to work with MQ connections or channel groups on MQSeries for Windows V2.1.

Commands

The *required parameters* and the *optional parameters* are listed. The parameters ***must*** occur in the order:

1. All required parameters, in the order stated, followed by
2. Optional parameters as required, in any order, unless specifically noted in the PCF definition.

Responses

The response data attribute is *always returned* whether it is requested or not. This parameter is required to identify, uniquely, the object when there is a possibility of multiple reply messages being returned.

The other attributes shown are *returned if requested* as optional parameters on the command. The response data attributes are not returned in a defined order.

Parameters and response data

Each parameter name is followed by its structure name in parentheses (details are given in Chapter 9, "Structures used for commands and responses" on page 333). The parameter identifier is given at the beginning of the description.

Constants

The values of constants used by PCF commands and responses are included in Appendix B, “Constants” on page 473.

Error codes

At the end of each command format definition there is a list of error codes that may be returned by that command. Full descriptions are given in the alphabetic list in Appendix A, “Error codes” on page 455.

Error codes applicable to all commands

In addition to those listed under each command format, any command may return the following in the response format header (descriptions of the MQRC_* error codes are given in the *MQSeries Application Programming Reference*):

Reason (MQLONG)
The value may be:

MQRC_CONNECTION_BROKEN
(2009, X'7D9') Connection to queue manager lost.

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_STORAGE_NOT_AVAILABLE
(2071, X'817') Insufficient storage available.

MQRC_MSG_TOO_BIG_FOR_Q
(2030, X'7EE') Message length greater than maximum for queue.

MQRC_NONE
(0, X'000') No reason to report.

MQRCCF_COMMAND_FAILED
Command failed.

MQRCCF_CFH_COMMAND_ERROR
Command identifier not valid.

MQRCCF_CFH_CONTROL_ERROR
Control option not valid.

MQRCCF_CFH_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFH_MSG_SEQ_NUMBER_ERR
Message sequence number not valid.

MQRCCF_CFH_PARM_COUNT_ERROR
Parameter count not valid.

MQRCCF_CFH_TYPE_ERROR
Type not valid.

MQRCCF_CFH_VERSION_ERROR
Structure version number is not valid.

MQRCCF_ENCODING_ERROR
Encoding error.

MQRCCF_MD_FORMAT_ERROR
Format not valid.

MQRCCF_MSG_TRUNCATED
Message truncated.

MQRCCF_MSG_LENGTH_ERROR
Message length not valid.

MQRCCF_MSG_SEQ_NUMBER_ERROR
Message sequence number not valid.

PCF commands and responses in groups

The commands and data responses are given in alphabetic order in this chapter. They can be usefully grouped as follows:

Queue Manager commands

Change Queue Manager (page 173)
Inquire Queue Manager (page 302)
Ping Queue Manager (page 317)

Process commands

Change Process (page 156)
Copy Process (page 198)
Create Process (page 232)
Delete Process (page 251)
Inquire Process (page 283)
Inquire Process Names (page 287)

Queue commands

Change Queue (page 160)
Clear Queue (page 179)
Copy Queue (page 202)
Create Queue (page 236)
Delete Queue (page 252)
Inquire Queue (page 289)
Inquire Queue Names (page 311)

Channel commands

Change Channel (page 139)
Copy Channel (page 181)
Create Channel (page 215)
Delete Channel (page 249)
Inquire Channel (page 256)
Inquire Channel Names (page 268)
Inquire Channel Status (page 271)
Ping Channel (page 314)
Reset Channel (page 318)
Resolve Channel (page 323)
Start Channel (page 325)
Start Channel Initiator (page 327)
Start Channel Listener (page 329)
Stop Channel (page 330)

Statistics command

Reset Queue Statistics (page 320)

Escape command

Escape (page 254)

Data responses to commands

Escape (Response) (page 255)
Inquire Channel (Response) (page 263)
Inquire Channel Names (Response) (page 270)
Inquire Channel Status (Response) (page 278)
Inquire Process (Response) (page 285)
Inquire Process Names (Response) (page 288)
Inquire Queue (Response) (page 295)
Inquire Queue Manager (Response) (page 305)
Inquire Queue Names (Response) (page 313)
Reset Queue Statistics (Response) (page 322)

Change Channel

The Change Channel (MQCMD_CHANGE_CHANNEL) command changes the specified attributes in a channel definition.

For any optional parameters that are omitted, the value does not change.

Required parameters:
ChannelName, *ChannelType*

Optional parameters (any ChannelType):
TransportType, *ChannelDesc*, *SecurityExit*, *MsgExit*, *SendExit*, *ReceiveExit*, *MaxMsgLength*, *SecurityUserData*, *MsgUserData*, *SendUserData*, *ReceiveUserData*

Optional parameters (sender or server ChannelType):
ModeName, *TpName*, *ConnectionName*, *XmitQName*, *MCAName*, *BatchSize*, *DiscInterval*, *ShortRetryCount*, *ShortRetryInterval*, *LongRetryCount*, *LongRetryInterval*, *SeqNumberWrap*, *DataConversion*, *MCAType*, *MCAUserIdentifier*, *UserIdentifier*, *Password*, *HeartbeatInterval*, *NonPersistentMsgSpeed BatchInterval*

Optional parameters (receiver ChannelType):
BatchSize, *PutAuthority*, *SeqNumberWrap*, *MCAUserIdentifier*, *MsgRetryExit*, *MsgRetryUserData*, *MsgRetryCount*, *MsgRetryInterval*, *HeartbeatInterval*, *NonPersistentMsgSpeed*

Optional parameters (requester ChannelType):
ModeName, *TpName*, *ConnectionName*, *MCAName*, *BatchSize*, *PutAuthority*, *SeqNumberWrap*, *MCAType*, *MCAUserIdentifier*, *UserIdentifier*, *Password*, *MsgRetryExit*, *MsgRetryUserData*, *MsgRetryCount*, *MsgRetryInterval* *HeartbeatInterval*, *NonPersistentMsgSpeed*

Optional parameters (server-connection ChannelType):
MCAUserIdentifier,

Optional parameters (client-connection ChannelType):
ModeName, *TpName QMgrName*, *ConnectionName UserIdentifier*, *Password*

Required parameters

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

Specifies the name of the channel definition to be changed.

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

ChannelType (MQCFIN)
Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

Specifies the type of the channel being changed. The value may be:

MQCHT_SENDER
Sender.

MQCHT_SERVER
Server.

MQCHT_RECEIVER
Receiver.

MQCHT_REQUESTER
Requester.

MQCHT_SVRCONN
Server-connection (for use by clients).

This value is not supported in the following environment: 32-bit Windows.

MQCHT_CLNTCONN
Client connection.

This value is not supported in the following environments: OS/400, 32-bit Windows.

Optional parameters

TransportType (MQCFIN)
Transmission protocol type (parameter identifier: MQIACH_XMIT_PROTOCOL_TYPE).

No check is made that the correct transport type has been specified if the channel is initiated from the other end. The value may be:

MQXPT_LU62
LU 6.2.

This value is not supported on 32-bit Windows.

MQXPT_TCP
TCP/IP.

This is the *only* value supported on 32-bit Windows.

MQXPT_NETBIOS
NetBIOS.

This value is supported in the following environments: OS/2, 32-bit Windows, Windows NT.

MQXPT_SPX
SPX.

This value is supported in the following environments: OS/2, Windows NT, Windows client, DOS client.

MQXPT_DECNET
DECnet.

This value is supported in the following environment: OpenVMS.

ChannelDesc (MQCFST)
Channel description (parameter identifier: MQCACH_DESC).

The maximum length of the string is MQ_CHANNEL_DESC_LENGTH.

Use characters from the character set, identified by the coded character set identifier (CCSID) for the message queue manager on which the command is executing, to ensure that the text is translated correctly.

SecurityExit (MQCFST)
Security exit name (parameter identifier: MQCACH_SEC_EXIT_NAME).

If a nonblank name is defined, the security exit is invoked at the following times:

- Immediately after establishing a channel.

  Before any messages are transferred, the exit is given the opportunity to instigate security flows to validate connection authorization.

- Upon receipt of a response to a security message flow.

  Any security message flows received from the remote processor on the remote machine are passed to the exit.

The exit is given the entire application message and message descriptor for modification.

The format of the string depends on the platform, as follows:

- On UNIX systems, it is of the form

  `libraryname(functionname)`

- On OS/2, Windows NT, and Windows 3.1, it is of the form

  `dllname(functionname)`

  where *dllname* is specified without the suffix ".DLL".

- On OS/400, it is of the form

  `progname libname`

  where *progname* occupies the first 10 characters, and *libname* the second 10 characters (both blank-padded to the right if necessary).

- On OpenVMS, it is of the form

  `imagename(functionname)`

The maximum length of the string is MQ_EXIT_NAME_LENGTH. Set unused character positions to blanks.

MsgExit (MQCFST)
Message exit name (parameter identifier: MQCACH_MSG_EXIT_NAME).

If a nonblank name is defined, the exit is invoked immediately after a message has been retrieved from the transmission queue. The exit is given the entire application message and message descriptor for modification.

For channels with a channel type (*ChannelType*) of MQCHT_SVRCONN or MQCHT_CLNTCONN, this parameter is not relevant, since message exits are not invoked for such channels.

The format of the string is the same as for *SecurityExit*.

The maximum length of the string is MQ_EXIT_NAME_LENGTH. Set unused character positions to blanks.

In the following environments, a list of exit names can be specified by using an MQCFSL structure instead of an MQCFST structure: AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

- The exits are invoked in the order specified in the list.

- A list with only one name is equivalent to specifying a single name in an MQCFST structure.
- You cannot specify both a list (MQCFSL) and a single entry (MQCFST) structure for the same channel attribute.
- The total length of all of the exit names in the list (excluding trailing blanks in each name) must not exceed MQ_TOTAL_EXIT_NAME_LENGTH. An individual string must not exceed MQ_EXIT_NAME_LENGTH.

SendExit (MQCFST)
Send exit name (parameter identifier: MQCACH_SEND_EXIT_NAME).

If a nonblank name is defined, the exit is invoked immediately before data is sent out on the network. The exit is given the complete transmission buffer before it is transmitted; the contents of the buffer can be modified as required.

The format of the string is the same as for *SecurityExit*.

The maximum length of the string is MQ_EXIT_NAME_LENGTH. Set unused character positions to blanks.

In the following environments, a list of exit names can be specified by using an MQCFSL structure instead of an MQCFST structure: AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

- The exits are invoked in the order specified in the list.
- A list with only one name is equivalent to specifying a single name in an MQCFST structure.
- You cannot specify both a list (MQCFSL) and a single entry (MQCFST) structure for the same channel attribute.
- The total length of all of the exit names in the list (excluding trailing blanks in each name) must not exceed MQ_TOTAL_EXIT_NAME_LENGTH. An individual string must not exceed MQ_EXIT_NAME_LENGTH.

ReceiveExit (MQCFST)
Receive exit name (parameter identifier: MQCACH_RCV_EXIT_NAME).

If a nonblank name is defined, the exit is invoked before data received from the network is processed. The complete transmission buffer is passed to the exit and the contents of the buffer can be modified as required.

The format of the string is the same as for *SecurityExit*.

The maximum length of the string is MQ_EXIT_NAME_LENGTH. Set unused character positions to blanks.

In the following environments, a list of exit names can be specified by using an MQCFSL structure instead of an MQCFST structure: AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

- The exits are invoked in the order specified in the list.
- A list with only one name is equivalent to specifying a single name in an MQCFST structure.

- You cannot specify both a list (MQCFSL) and a single entry (MQCFST) structure for the same channel attribute.
- The total length of all of the exit names in the list (excluding trailing blanks in each name) must not exceed MQ_TOTAL_EXIT_NAME_LENGTH. An individual string must not exceed MQ_EXIT_NAME_LENGTH.

MaxMsgLength (MQCFIN)
Maximum message length (parameter identifier: MQIACH_MAX_MSG_LENGTH).

Specifies the maximum message length that can be transmitted on the channel. This is compared with the value for the remote channel and the actual maximum is the lowest of the two values.

The value zero means the maximum message length for the queue manager.

The lower limit for this parameter is 0. The upper limit depends on the environment:

- On AIX, HP-UX, OS/2, Sun Solaris, and Windows NT, the maximum message length is 100 MB (104 857 600 bytes).
- On OpenVMS, OS/400, Tandem NSK, UNIX systems not listed above, and 32-bit Windows, the maximum message length is 4 MB (4 194 304 bytes).

SecurityUserData (MQCFST)
Security exit user data (parameter identifier: MQCACH_SEC_EXIT_USER_DATA).

Specifies user data that is passed to the security exit.

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

MsgUserData (MQCFST)
Message exit user data (parameter identifier: MQCACH_MSG_EXIT_USER_DATA).

Specifies user data that is passed to the message exit.

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

In the following environments, a list of exit user data strings can be specified by using an MQCFSL structure instead of an MQCFST structure: AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

- Each exit user data string is passed to the exit at the same ordinal position in the *MsgExit* list.
- A list with only one name is equivalent to specifying a single name in an MQCFST structure.
- You cannot specify both a list (MQCFSL) and a single entry (MQCFST) structure for the same channel attribute.
- The total length of all of the exit user data in the list (excluding trailing blanks in each string) must not exceed MQ_TOTAL_EXIT_DATA_LENGTH. An individual string must not exceed MQ_EXIT_DATA_LENGTH.

SendUserData (MQCFST)
Send exit user data (parameter identifier: MQCACH_SEND_EXIT_USER_DATA).

Specifies user data that is passed to the send exit.

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

In the following environments, a list of exit user data strings can be specified by using an MQCFSL structure instead of an MQCFST structure: AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

- Each exit user data string is passed to the exit at the same ordinal position in the *SendExit* list.
- A list with only one name is equivalent to specifying a single name in an MQCFST structure.
- You cannot specify both a list (MQCFSL) and a single entry (MQCFST) structure for the same channel attribute.
- The total length of all of the exit user data in the list (excluding trailing blanks in each string) must not exceed MQ_TOTAL_EXIT_DATA_LENGTH. An individual string must not exceed MQ_EXIT_DATA_LENGTH.

ReceiveUserData (MQCFST)
Receive exit user data (parameter identifier: MQCACH_RCV_EXIT_USER_DATA).

Specifies user data that is passed to the receive exit.

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

In the following environments, a list of exit user data strings can be specified by using an MQCFSL structure instead of an MQCFST structure: AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

- Each exit user data string is passed to the exit at the same ordinal position in the *ReceiveExit* list.
- A list with only one name is equivalent to specifying a single name in an MQCFST structure.
- You cannot specify both a list (MQCFSL) and a single entry (MQCFST) structure for the same channel attribute.
- The total length of all of the exit user data in the list (excluding trailing blanks in each string) must not exceed MQ_TOTAL_EXIT_DATA_LENGTH. An individual string must not exceed MQ_EXIT_DATA_LENGTH.

ModeName (MQCFST)
Mode name (parameter identifier: MQCACH_MODE_NAME).

This is the LU 6.2 mode name.

The maximum length of the string is MQ_MODE_NAME_LENGTH.

- On OpenVMS, OS/400, Tandem NSK, UNIX systems, and Windows NT, this can be set only to blanks. The actual name is taken instead

from the CPI-C Communications Side Object or (on Windows NT) from the CPI-C symbolic destination name properties.

- On 32-bit Windows, this parameter is accepted but ignored.

This parameter is valid only for channels with a *TransportType* of MQXPT_LU62. It is not valid for receiver channels.

TpName (MQCFST)
Transaction program name (parameter identifier: MQCACH_TP_NAME).

This is the LU 6.2 transaction program name.

The maximum length of the string is MQ_TP_NAME_LENGTH.

On OpenVMS, OS/400, Tandem NSK, UNIX systems, and Windows NT, this can be set only to blanks. The actual name is taken instead from the CPI-C Communications Side Object or (on Windows NT) from the CPI-C symbolic destination name properties.

- On 32-bit Windows, this parameter is accepted but ignored.

This parameter is valid only for channels with a *TransportType* of MQXPT_LU62. It is not valid for receiver channels.

ConnectionName (MQCFST)
Connection name (parameter identifier: MQCACH_CONNECTION_NAME).

The maximum length of the string is MQ_CONN_NAME_LENGTH.

Specify the name of the machine as required for the stated *TransportType*:

- For MQXPT_LU62 on OS/2, specify the fully-qualified name of the partner LU. On OS/400, and UNIX systems, specify the name of the CPI-C communications side object. On Windows NT specify the CPI-C symbolic destination name.
- For MQXPT_TCP specify either the host name or the network address of the remote machine.
- For MQXPT_NETBIOS specify the NetBIOS station name.
- For MQXPT_SPX specify the 4 byte network address, the 6 byte node address, and the 2 byte socket number. These should be entered in hexadecimal, with a period separating the network and node addresses. The socket number should be enclosed in brackets, for example:

  ```
  CONNAME('0a0b0c0d.804abcde23a1(5e86)')
  ```

  If the socket number is omitted, the MQSeries default value (5e86 hex) is assumed.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_REQUESTER or MQCHT_CLNTCONN.

XmitQName (MQCFST)
Transmission queue name (parameter identifier: MQCACH_XMIT_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

A transmission queue name is required (either previously defined or specified here) if *ChannelType* is MQCHT_SENDER or MQCHT_SERVER. It is not valid for other channel types.

MCAName (MQCFST)
Message channel agent name (parameter identifier: MQCACH_MCA_NAME).

This is reserved, and if specified can be set only to blanks.

The maximum length of the string is MQ_MCA_NAME_LENGTH.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, or MQCHT_REQUESTER.

BatchSize (MQCFIN)
Batch size (parameter identifier: MQIACH_BATCH_SIZE).

The maximum number of messages that should be sent down a channel before a checkpoint is taken.

The batch size which is actually used is the lowest of the following:

- The *BatchSize* of the sending channel
- The *BatchSize* of the receiving channel
- The maximum number of uncommitted messages at the sending queue manager
- The maximum number of uncommitted messages at the receiving queue manager

The maximum number of uncommitted messages is specified by the *MaxUncommittedMsgs* parameter of the Change Queue Manager command.

Specify a value in the range 1-9999.

This parameter is not valid for channels with a *ChannelType* of MQCHT_SVRCONN or MQCHT_CLNTCONN.

DiscInterval (MQCFIN)
Disconnection interval (parameter identifier: MQIACH_DISC_INTERVAL).

This defines the maximum number of seconds that the channel waits for messages to be put on a transmission queue before terminating the channel. A value of zero causes the message channel agent to wait indefinitely.

Specify a value in the range 0 through 999 999.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER or MQCHT_SERVER.

ShortRetryCount (MQCFIN)
Short retry count (parameter identifier: MQIACH_SHORT_RETRY).

The maximum number of attempts that are made by a sender or server channel to establish a connection to the remote machine, at intervals specified by *ShortRetryInterval* before the (normally longer) *LongRetryCount* and *LongRetryInterval* are used.

Retry attempts are made if the channel fails to connect initially (whether it is started automatically by the channel initiator or by an explicit command), and also if the connection fails after the channel has successfully

connected. However, if the cause of the failure is such that retry is unlikely to be successful, retries are not attempted.

Specify a value in the range 0 through 999 999 999.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER or MQCHT_SERVER.

ShortRetryInterval (MQCFIN)
Short timer (parameter identifier: MQIACH_SHORT_TIMER).

Specifies the short retry wait interval for a sender or server channel that is started automatically by the channel initiator. It defines the interval in seconds between attempts to establish a connection to the remote machine.

The time is approximate; zero means that another connection attempt is made as soon as possible.

Specify a value in the range 0 through 999 999. Values exceeding this are treated as 999 999.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER or MQCHT_SERVER.

LongRetryCount (MQCFIN)
Long retry count (parameter identifier: MQIACH_LONG_RETRY).

When a sender or server channel is attempting to connect to the remote machine, and the count specified by *ShortRetryCount* has been exhausted, this specifies the maximum number of further attempts that are made to connect to the remote machine, at intervals specified by *LongRetryInterval*.

If this count is also exhausted without success, an error is logged to the operator, and the channel is stopped. The channel must subsequently be restarted with a command (it is not started automatically by the channel initiator), and it then makes only one attempt to connect, as it is assumed that the problem has now been cleared by the administrator. The retry sequence is not carried out again until after the channel has successfully connected.

Specify a value in the range 0 through 999 999 999.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER or MQCHT_SERVER.

LongRetryInterval (MQCFIN)
Long timer (parameter identifier: MQIACH_LONG_TIMER).

Specifies the long retry wait interval for a sender or server channel that is started automatically by the channel initiator. It defines the interval in seconds between attempts to establish a connection to the remote machine, after the count specified by *ShortRetryCount* has been exhausted.

The time is approximate; zero means that another connection attempt is made as soon as possible.

Specify a value in the range 0 through 999 999. Values exceeding this are treated as 999 999.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER or MQCHT_SERVER.

DataConversion (MQCFIN)
Whether sender should convert application data (parameter identifier: MQIACH_DATA_CONVERSION).

This parameter is valid only for *ChannelType* values of MQCHT_SENDER or MQCHT_SERVER.

The value may be:

MQCDC_NO_SENDER_CONVERSION
No conversion by sender.

MQCDC_SENDER_CONVERSION
Conversion by sender.

This value is not supported on 32-bit Windows.

PutAuthority (MQCFIN)
Put authority (parameter identifier: MQIACH_PUT_AUTHORITY).

Specifies whether the user identifier in the context information associated with a message should be used to establish authority to put the message on the destination queue.

This parameter is valid only for channels with a *ChannelType* value of MQCHT_RECEIVER or MQCHT_REQUESTER

The value may be:

MQPA_DEFAULT
Default user identifier is used.

MQPA_CONTEXT
Context user identifier is used.

SeqNumberWrap (MQCFIN)
Sequence wrap number (parameter identifier: MQIACH_SEQUENCE_NUMBER_WRAP).

Specifies the maximum message sequence number. When the maximum is reached, sequence numbers wrap to start again at 1.

The maximum message sequence number is not negotiable; the local and remote channels must wrap at the same number.

Specify a value in the range 100 through 999 999 999.

This parameter is not valid for channels with a *ChannelType* of MQCHT_SVRCONN or MQCHT_CLNTCONN.

MCAType (MQCFIN)
Message channel agent type (parameter identifier: MQIACH_MCA_TYPE).

Specifies the type of the message channel agent program.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, or MQCHT_REQUESTER.

The value may be:

MQMCAT_PROCESS
Process.

MQMCAT_THREAD
Thread (OS/2 and Windows NT only).

MCAUserIdentifier (MQCFST)
Message channel agent user identifier (parameter identifier: MQCACH_MCA_USER_ID).

If this is nonblank, it is the user identifier which is to be used by the message channel agent for authorization to access MQSeries resources, including (if *PutAuthority* is MQPA_DEFAULT) authorization to put the message to the destination queue for receiver or requester channels.

If it is blank, the message channel agent uses its default user identifier.

This user identifier can be overridden by one supplied by a channel security exit.

This parameter is not valid for channels with a *ChannelType* of MQCHT_CLNTCONN.

The maximum length of the string is MQ_USER_ID_LENGTH.

UserIdentifier (MQCFST)
Task user identifier (parameter identifier: MQCACH_USER_ID).

This is used by the message channel agent when attempting to initiate a secure SNA session with a remote message channel agent. It is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_REQUESTER or MQCHT_CLNTCONN.

- This parameter is supported in the following environments: OpenVMS, OS/2, Tandem NSK, UNIX systems.
- On 32-bit Windows, the parameter is accepted but ignored.

The maximum length of the string is MQ_USER_ID_LENGTH. However, only the first 10 characters are used.

Password (MQCFST)
Password (parameter identifier: MQCACH_PASSWORD).

This is used by the message channel agent when attempting to initiate a secure SNA session with a remote message channel agent. It is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_REQUESTER or MQCHT_CLNTCONN.

- This parameter is supported in the following environments: OpenVMS, OS/2, Tandem NSK, UNIX systems.
- On 32-bit Windows, the parameter is accepted but ignored.

The maximum length of the string is MQ_PASSWORD_LENGTH. However, only the first 10 characters are used.

MsgRetryExit (MQCFST)
Message retry exit name (parameter identifier: MQCACH_MR_EXIT_NAME).

- This parameter is supported in the following environments: AIX, AT&T GIS UNIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.
- On 32-bit Windows, the parameter is accepted but must be blank.

If a nonblank name is defined, the exit is invoked prior to performing a wait before retrying a failing message.

The format of the string is the same as for *SecurityExit*.

The maximum length of the string is MQ_EXIT_NAME_LENGTH. Set unused character positions to blanks.

This parameter is valid only for *ChannelType* values of MQCHT_RECEIVER or MQCHT_REQUESTER.

MsgRetryUserData (MQCFST)
Message retry exit user data (parameter identifier: MQCACH_MR_EXIT_USER_DATA).

- This parameter is supported in the following environments: AIX, AT&T GIS UNIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.
- On 32-bit Windows, the parameter is accepted but ignored.

Specifies user data that is passed to the message retry exit.

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

This parameter is valid only for *ChannelType* values of MQCHT_RECEIVER or MQCHT_REQUESTER.

MsgRetryCount (MQCFIN)
Message retry count (parameter identifier: MQIACH_MR_COUNT).

- This parameter is supported in the following environments: AIX, AT&T GIS UNIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.
- On 32-bit Windows, the parameter is accepted but must be zero.

Specifies the number of times that a failing message should be retried.

Specify a value in the range 0 through 999 999 999.

This parameter is valid only for *ChannelType* values of MQCHT_RECEIVER or MQCHT_REQUESTER.

MsgRetryInterval (MQCFIN)
Message retry interval (parameter identifier: MQIACH_MR_INTERVAL).

- This parameter is supported in the following environments: AIX, AT&T GIS UNIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.
- On 32-bit Windows, the parameter is accepted but must be zero.

Specifies the minimum time interval in milliseconds between retries of failing messages.

Specify a value in the range 0 through 999 999 999.

This parameter is valid only for *ChannelType* values of MQCHT_RECEIVER or MQCHT_REQUESTER.

QMgrName (MQCFST)
Queue-manager name (parameter identifier: MQCA_Q_MGR_NAME).

For channels with a *ChannelType* of MQCHT_CLNTCONN, this is the name of a queue manager to which a client application can request connection.

On 32-bit Windows, this parameter is accepted but ignored.

For channels of other types, this parameter is not valid. The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

HeartbeatInterval (MQCFIN)
Heartbeat interval (parameter identifier: MQIACH_HB_INTERVAL).

The interpretation of this parameter depends on the channel type, as follows:

- For a channel type of MQCHT_SENDER, MQCHT_SERVER, MQCHT_RECEIVER or MQCHT_REQUESTER, this is the time in seconds between heartbeat flows passed from the sending MCA when there are no messages on the transmission queue. This gives the receiving MCA the opportunity to quiesce the channel. To be useful, *HeartbeatInterval* should be significantly less than *DiscInterval*. However, the only check is that the value is within the permitted range.

  This type of heartbeat is supported in the following environments: AIX, HP-UX, MVS/ESA, OS/2, OS/400, Sun Solaris, Windows NT.

- For a channel type of MQCHT_CLNTCONN or MQCHT_SVRCONN, this is the time in seconds between heartbeat flows passed from the server MCA when that MCA has issued an MQGET call with the MQGMO_WAIT option on behalf of a client application. This allows the server MCA to handle situations where the client connection fails during an MQGET with MQGMO_WAIT.

  This type of heartbeat is supported in the following environments: AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

The value must be in the range 0 through 999 999. A value of 0 means that no heartbeat exchange occurs. The value that is actually used is the larger of the values specified at the sending side and receiving side.

NonPersistentMsgSpeed (MQCFIN)
Speed at which nonpersistent messages are to be sent (parameter identifier: MQIACH_NPM_SPEED).

This parameter is supported in the following environments: AIX, HP-UX, OS/2, OS/400, Sun Solaris, 32-bit Windows, Windows NT.

Specifying MQNPMS_FAST means that nonpersistent messages on a channel need not wait for a syncpoint before being made available for retrieval. The advantage of this is that nonpersistent messages become available for retrieval far more quickly. The disadvantage is that because they do not wait for a syncpoint, they may be lost if there is a transmission failure.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_RECEIVER or MQCHT_REQUESTER. The value may be:

MQNPMS_NORMAL
Normal speed.

MQNPMS_FAST
Fast speed.

BatchInterval (MQCFIN)
Batch interval (parameter identifier: MQIACH_BATCH_INTERVAL).

This is the approximate time in milliseconds that a channel will keep a batch open, if fewer than *BatchSize* messages have been transmitted in the current batch.

This parameter is supported in the following environments: AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

If *BatchInterval* is greater than zero, the batch is terminated by whichever of the following occurs first:

- *BatchSize* messages have been sent, or
- *BatchInterval* milliseconds have elapsed since the start of the batch.

If *BatchInterval* is zero, the batch is terminated by whichever of the following occurs first:

- *BatchSize* messages have been sent, or
- the transmission queue becomes empty.

BatchInterval must be in the range zero through 999 999 999.

This parameter applies only to channels with a *ChannelType* of:

MQCHT_SENDER
MQCHT_SERVER

Error codes

In addition to the values for any command shown on page 136, for this command the following may be returned in the response format header:

Reason (MQLONG)
The value may be:

MQRCCF_BATCH_INT_ERROR
Batch interval not valid.

MQRCCF_BATCH_INT_WRONG_TYPE
Batch interval parameter not allowed for this channel type.

MQRCCF_BATCH_SIZE_ERROR
Batch size not valid.

MQRCCF_CFIN_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFIN_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIN_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFSL_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFSL_TOTAL_LENGTH_ERROR
Total string length error.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_CHANNEL_NAME_ERROR
Channel name error.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_CHANNEL_TYPE_ERROR
Channel type not valid.

MQRCCF_DISC_INT_ERROR
Disconnection interval not valid.

MQRCCF_DISC_INT_WRONG_TYPE
Disconnection interval not allowed for this channel type.

MQRCCF_HB_INTERVAL_ERROR
Heartbeat interval not valid.

MQRCCF_HB_INTERVAL_WRONG_TYPE
Heartbeat interval parameter not allowed for this channel type.

MQRCCF_LONG_RETRY_ERROR
Long retry count not valid.

MQRCCF_LONG_RETRY_WRONG_TYPE
Long retry parameter not allowed for this channel type.

MQRCCF_LONG_TIMER_ERROR
Long timer not valid.

MQRCCF_LONG_TIMER_WRONG_TYPE
Long timer parameter not allowed for this channel type.

MQRCCF_MAX_MSG_LENGTH_ERROR
Maximum message length not valid.

MQRCCF_MCA_NAME_ERROR
Message channel agent name error.

MQRCCF_MCA_NAME_WRONG_TYPE
Message channel agent name not allowed for this channel type.

MQRCCF_MCA_TYPE_ERROR
Message channel agent type not valid.

MQRCCF_MISSING_CONN_NAME
Connection name parameter required but missing.

MQRCCF_MR_COUNT_ERROR
Message retry count not valid.

MQRCCF_MR_COUNT_WRONG_TYPE
Message-retry count parameter not allowed for this channel type.

MQRCCF_MR_EXIT_NAME_ERROR
Channel message-retry exit name error.

MQRCCF_MR_EXIT_NAME_WRONG_TYPE
Message-retry exit parameter not allowed for this channel type.

MQRCCF_MR_INTERVAL_ERROR
Message retry interval not valid.

MQRCCF_MR_INTERVAL_WRONG_TYPE
Message-retry interval parameter not allowed for this channel type.

MQRCCF_MSG_EXIT_NAME_ERROR
Channel message exit name error.

MQRCCF_NPM_SPEED_ERROR
Nonpersistent message speed not valid.

MQRCCF_NPM_SPEED_WRONG_TYPE
Nonpersistent message speed parameter not allowed for this channel type.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_PARM_SEQUENCE_ERROR
Parameter sequence not valid.

MQRCCF_PUT_AUTH_ERROR
Put authority value not valid.

MQRCCF_PUT_AUTH_WRONG_TYPE
Put authority parameter not allowed for this channel type.

MQRCCF_RCV_EXIT_NAME_ERROR
Channel receive exit name error.

MQRCCF_SEC_EXIT_NAME_ERROR
Channel security exit name error.

MQRCCF_SEND_EXIT_NAME_ERROR
Channel send exit name error.

MQRCCF_SEQ_NUMBER_WRAP_ERROR
Sequence wrap number not valid.

MQRCCF_SHORT_RETRY_ERROR
Short retry count not valid.

MQRCCF_SHORT_RETRY_WRONG_TYPE
Short retry parameter not allowed for this channel type.

MQRCCF_SHORT_TIMER_ERROR
Short timer value not valid.

MQRCCF_SHORT_TIMER_WRONG_TYPE
Short timer parameter not allowed for this channel type.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

MQRCCF_XMIT_PROTOCOL_TYPE_ERR
Transmission protocol type not valid.

MQRCCF_XMIT_Q_NAME_ERROR
Transmission queue name error.

MQRCCF_XMIT_Q_NAME_WRONG_TYPE
Transmission queue name not allowed for this channel type.

Change Process

The Change Process (MQCMD_CHANGE_PROCESS) command changes the specified attributes of an existing MQSeries process definition.

| *This PCF is not supported if you are using MQSeries for Windows Version 2.1.*

For any optional parameters that are omitted, the value does not change.

Required parameters:

ProcessName

Optional parameters:

ProcessDesc, *ApplType*, *ApplId*, *EnvData UserData*

Required parameters

ProcessName (MQCFST)

The name of the process definition to be changed (parameter identifier: MQCA_PROCESS_NAME).

The maximum length of the string is MQ_PROCESS_NAME_LENGTH.

Optional parameters

ProcessDesc (MQCFST)

Description of process definition (parameter identifier: MQCA_PROCESS_DESC).

A plain-text comment that provides descriptive information about the process definition. It should contain only displayable characters.

The maximum length of the string is MQ_PROCESS_DESC_LENGTH.

If characters are used that are not in the coded character set identifier (CCSID) for the queue manager on which the command is executing, they may be translated incorrectly.

ApplType (MQCFIN)

Application type (parameter identifier: MQIA_APPL_TYPE).

Valid application types are:

MQAT_OS400
: OS/400 application.

MQAT_OS2
: OS/2 or Presentation Manager application.

MQAT_WINDOWS_NT
: Windows NT or 32-bit Windows application.

MQAT_DOS
: DOS client application.

MQAT_WINDOWS
: Windows client or 16-bit Windows application.

MQAT_UNIX
: UNIX application.

MQAT_AIX
AIX application (same value as MQAT_UNIX).

MQAT_CICS
CICS transaction.

| MQAT_VMS
| OpenVMS application.

| MQAT_NSK
| Tandem NSK application.

MQAT_DEFAULT
Default application type.

user-value: User-defined application type in the range 65 536 through 999 999 999 (not checked).

Only application types (other than user-defined types) that are supported on the platform at which the command is executed should be used:

| • On OpenVMS:

| MQAT_VMS (default),
| MQAT_DOS,
| MQAT_WINDOWS, and
| MQAT_DEFAULT are supported.

• On OS/2:

MQAT_OS2 (default),
MQAT_DOS,
MQAT_WINDOWS,
MQAT_AIX,
MQAT_CICS, and
MQAT_DEFAULT are supported.

• On OS/400:

MQAT_OS400 (default),
MQAT_CICS, and
MQAT_DEFAULT are supported.

| • On Tandem NSK:

| MQAT_NSK (default),
| MQAT_DOS,
| MQAT_WINDOWS, and
| MQAT_DEFAULT are supported.

• On UNIX systems:

MQAT_UNIX (default),
MQAT_OS2,
MQAT_DOS,
MQAT_WINDOWS,
MQAT_CICS, and
MQAT_DEFAULT are supported.

• On Windows NT:

MQAT_WINDOWS_NT (default),
MQAT_OS2

MQAT_DOS,
MQAT_WINDOWS,
MQAT_CICS, and
MQAT_DEFAULT are supported.

ApplId (MQCFST)
Application identifier (parameter identifier: MQCA_APPL_ID).

This is the name of the application to be started, on the platform for which the command is executing, and might typically be a program name and library name.

The maximum length of the string is MQ_PROCESS_APPL_ID_LENGTH.

EnvData (MQCFST)
Environment data (parameter identifier: MQCA_ENV_DATA).

A character string that contains environment information pertaining to the application to be started.

The maximum length of the string is MQ_PROCESS_ENV_DATA_LENGTH.

UserData (MQCFST)
User data (parameter identifier: MQCA_USER_DATA).

A character string that contains user information pertaining to the application (defined by *ApplId*) that is to be started.

The maximum length of the string is MQ_PROCESS_USER_DATA_LENGTH.

Error codes

In addition to the values for any command shown on page 136, for this command the following may be returned in the response format header:

Reason (MQLONG)
The value may be:

MQRC_UNKNOWN_OBJECT_NAME
(2085, X'825') Unknown object name.

MQRCCF_ATTR_VALUE_ERROR
Attribute value not valid.

MQRCCF_CFIN_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFIN_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIN_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_FORCE_VALUE_ERROR
Force value not valid.

MQRCCF_OBJECT_NAME_ERROR
Object name not valid.

MQRCCF_OBJECT_OPEN
Object is open.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_PARM_SEQUENCE_ERROR
Parameter sequence not valid.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

Change Queue

The Change Queue (MQCMD_CHANGE_Q) command changes the specified attributes of an existing MQSeries queue.

For any optional parameters that are omitted, the value does not change.

Required parameters:
QName, *QType*,

Optional parameters (any QType):
QDesc, *InhibitPut*, *DefPriority*, *DefPersistence*

Optional parameters (alias QType):
Force, *InhibitGet*, *BaseQName*, *Scope*

Optional parameters (local QType):
Force, *InhibitGet*, *ProcessName*, *MaxQDepth*, *MaxMsgLength*, *BackoutThreshold*, *BackoutRequeueName*, *Shareability*, *DefInputOpenOption*, *HardenGetBackout*, *MsgDeliverySequence*, *RetentionInterval*, *DistLists*, *Usage*, *InitiationQName*, *TriggerControl*, *TriggerType*, *TriggerMsgPriority*, *TriggerDepth*, *TriggerData*, *Scope*, *QDepthHighLimit*, *QDepthLowLimit*, *QDepthMaxEvent*, *QDepthHighEvent*, *QDepthLowEvent*, *QServiceInterval*, *QServiceIntervalEvent*

Optional parameters (remote QType):
Force, *RemoteQName*, *RemoteQMgrName*, *XmitQName*, *Scope*

Optional parameters (model QType):
InhibitGet, *ProcessName*, *MaxQDepth*, *MaxMsgLength*, *BackoutThreshold*, *BackoutRequeueName*, *Shareability*, *DefInputOpenOption*, *HardenGetBackout*, *MsgDeliverySequence*, *RetentionInterval*, *DistLists*, *Usage*, *InitiationQName*, *TriggerControl*, *TriggerType*, *TriggerMsgPriority*, *TriggerDepth*, *TriggerData*, *DefinitionType*, *QDepthHighLimit*, *QDepthLowLimit*, *QDepthMaxEvent*, *QDepthHighEvent*, *QDepthLowEvent*, *QServiceInterval*, *QServiceIntervalEvent*

Required parameters

QName (MQCFST)
Queue name (parameter identifier: MQCA_Q_NAME).

The name of the queue to be changed. The maximum length of the string is MQ_Q_NAME_LENGTH.

QType (MQCFIN)
Queue type (parameter identifier: MQIA_Q_TYPE).

The value specified must match the type of the queue being changed.

The value may be:

MQQT_ALIAS
Alias queue definition.

MQQT_LOCAL
Local queue.

MQQT_REMOTE
Local definition of a remote queue.

The following is supported on all platforms, but on OS/400 for receipt by MQSeries for AS/400 V4R2:

MQQT_MODEL
Model queue definition.

Optional parameters

Force (MQCFIN)
Force changes (parameter identifier: MQIACF_FORCE).

Specifies whether the command should be forced to complete when conditions are such that completing the command would affect an open queue. The conditions depend upon the type of the queue that is being changed:

Alias QType: *BaseQName* is specified with a queue name and an application has the alias queue open.

Local QType: Either of the following conditions indicate that a local queue would be affected:

- *Shareability* is specified as MQQA_NOT_SHAREABLE and more than one application has the local queue open for input.
- The *Usage* value is changed and one or more applications has the local queue open, or there are one or more messages on the queue. (The *Usage* value should not normally be changed while there are messages on the queue; the format of messages changes when they are put on a transmission queue.)

Remote QType: Either of the following conditions indicate that a remote queue would be affected:

- *XmitQName* is specified with a transmission-queue name (or blank) and an application has a remote queue open that would be affected by this change.
- Any of the *RemoteQName*, *RemoteQMgrName* or *XmitQName* parameters is specified with a queue or queue-manager name, and one or more applications has a queue open that resolved through this definition as a queue-manager alias.

Model QType: This parameter is not valid for model queues.

Note: A value of MQFC_YES is not required if this definition is in use as a reply-to queue definition only.

The value may be:

MQFC_YES
Force the change.

MQFC_NO
Do not force the change.

QDesc (MQCFST)
Queue description (parameter identifier: MQCA_Q_DESC).

Text that briefly describes the object.

The maximum length of the string is MQ_Q_DESC_LENGTH.

Use characters from the character set identified by the coded character set identifier (CCSID) for the message queue manager on which the command is executing to ensure that the text is translated correctly if it is sent to another queue manager.

InhibitPut (MQCFIN)
Whether put operations are allowed (parameter identifier: MQIA_INHIBIT_PUT).

Specifies whether messages can be put on the queue.

The value may be:

MQQA_PUT_ALLOWED
Put operations are allowed.

MQQA_PUT_INHIBITED
Put operations are inhibited.

DefPriority (MQCFIN)
Default priority (parameter identifier: MQIA_DEF_PRIORITY).

Specifies the default priority of messages put on the queue. The value must be in the range zero through to the maximum priority value that is supported (9).

DefPersistence (MQCFIN)
Default persistence (parameter identifier: MQIA_DEF_PERSISTENCE).

Specifies the default for message-persistence on the queue. Message persistence determines whether or not messages are preserved across restarts of the queue manager.

The value may be:

MQPER_PERSISTENT
Message is persistent.

MQPER_NOT_PERSISTENT
Message is not persistent.

InhibitGet (MQCFIN)
Whether get operations are allowed (parameter identifier: MQIA_INHIBIT_GET).

The value may be:

MQQA_GET_ALLOWED
Get operations are allowed.

MQQA_GET_INHIBITED
Get operations are inhibited.

BaseQName (MQCFST)
Queue name to which the alias resolves (parameter identifier: MQCA_BASE_Q_NAME).

This is the name of a local or remote queue that is defined to the local queue manager.

The maximum length of the string is MQ_Q_NAME_LENGTH.

ProcessName (MQCFST)
Name of process definition for the queue (parameter identifier: MQCA_PROCESS_NAME).

Specifies the local name of the MQSeries process that identifies the application that should be started when a trigger event occurs.

- On AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT, if the queue is a transmission queue the process name can be left as all blanks.
- On 32-bit Windows, this parameter is accepted but ignored.
- In other environments, the process name must be nonblank for a trigger event to occur (although it can be set after the queue has been created).

The maximum length of the string is MQ_PROCESS_NAME_LENGTH.

MaxQDepth (MQCFIN)
Maximum queue depth (parameter identifier: MQIA_MAX_Q_DEPTH).

The maximum number of messages allowed on the queue. Note that other factors may cause the queue to be treated as full; for example, it will appear to be full if there is no storage available for a message.

Specify a value in the range 0 through 640 000.

MaxMsgLength (MQCFIN)
Maximum message length (parameter identifier: MQIA_MAX_MSG_LENGTH).

Specifies the maximum length for messages on the queue. Because applications may use the value of this attribute to determine the size of buffer they need to retrieve messages from the queue, the value should be changed only if it is known that this will not cause an application to operate incorrectly.

You are recommended not to set a value that is greater than the queue manager's *MaxMsgLength* attribute.

The lower limit for this parameter is 0. The upper limit depends on the environment:

- On AIX, HP-UX, OS/2, Sun Solaris, and Windows NT, the maximum message length is 100 MB (104 857 600 bytes).
- On OpenVMS, OS/400, Tandem NSK, UNIX systems not listed above, and 32-bit Windows, the maximum message length is 4 MB (4 194 304 bytes).

BackoutThreshold (MQCFIN)
Backout threshold (parameter identifier: MQIA_BACKOUT_THRESHOLD).

That is, the number of times a message can be backed out before it is transferred to the backout queue specified by *BackoutRequeueName*.

If the value is subsequently reduced, any messages already on the queue that have been backed out at least as many times as the new value remain on the queue, but such messages are transferred if they are backed out again.

Specify a value in the range 0 through 999 999 999.

BackoutRequeueName (MQCFST)
Excessive backout requeue name (parameter identifier: MQCA_BACKOUT_REQ_Q_NAME).

Specifies the local name of the queue (not necessarily a local queue) to which a message is transferred if it is backed out more times than the value of *BackoutThreshold*.

The backout queue does not need to exist at this time but it must exist when the *BackoutThreshold* value is exceeded.

The maximum length of the string is MQ_Q_NAME_LENGTH.

Shareability (MQCFIN)
Whether queue can be shared (parameter identifier: MQIA_SHAREABILITY).

Specifies whether multiple instances of applications, can open this queue for input.

The value may be:

MQQA_SHAREABLE
Queue is shareable.

MQQA_NOT_SHAREABLE
Queue is not shareable.

DefInputOpenOption (MQCFIN)
Default input open option (parameter identifier: MQIA_DEF_INPUT_OPEN_OPTION).

Specifies the default share option for applications opening this queue for input.

The value may be:

MQOO_INPUT_EXCLUSIVE
Open queue to get messages with exclusive access.

MQOO_INPUT_SHARED
Open queue to get messages with shared access.

HardenGetBackout (MQCFIN)
Whether to harden backout count (parameter identifier: MQIA_HARDEN_GET_BACKOUT).

Specifies whether the count of backed out messages should be saved (hardened) across restarts of the message queue manager.

Note: MQSeries for AS/400 always hardens the count, regardless of the setting of this attribute.

The value may be:

MQQA_BACKOUT_HARDENED
Backout count remembered.

MQQA_BACKOUT_NOT_HARDENED
Backout count may not be remembered.

MsgDeliverySequence (MQCFIN)
Whether priority is relevant (parameter identifier: MQIA_MSG_DELIVERY_SEQUENCE).

The value may be:

MQMDS_PRIORITY
Messages are returned in priority order.

MQMDS_FIFO
Messages are returned in FIFO order (first in, first out).

RetentionInterval (MQCFIN)
Retention interval (parameter identifier: MQIA_RETENTION_INTERVAL).

The number of hours for which the queue may be needed, based on the date and time when the queue was created.

This information is available to a housekeeping application or an operator and may be used to determine when a queue is no longer required. The queue manager does not delete queues nor does it prevent queues from being deleted if their retention interval has not expired. It is the user's responsibility to take any required action.

Specify a value in the range 0 through 999 999 999.

DistLists (MQCFIN)
Distribution list support (parameter identifier: MQIA_DIST_LISTS).

Specifies whether distribution-list messages can be placed on the queue.

Note: This attribute is set by the sending message channel agent (MCA) which removes messages from the queue; this happens each time the sending MCA establishes a connection to a receiving MCA on a partnering queue manager. The attribute should not normally be set by administrators, although it can be set if the need arises.

This parameter is supported in the following environments: AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

The value may be:

MQDL_SUPPORTED
Distribution lists supported.

MQDL_NOT_SUPPORTED
Distribution lists not supported.

Usage (MQCFIN)
Usage (parameter identifier: MQIA_USAGE).

Specifies whether the queue is for normal usage or for transmitting messages to a remote message queue manager.

The value may be:

MQUS_NORMAL
Normal usage.

MQUS_TRANSMISSION
Transmission queue.

InitiationQName (MQCFST)
Initiation queue name (parameter identifier: MQCA_INITIATION_Q_NAME).

The local queue for trigger messages relating to this queue. The initiation queue must be on the same queue manager.

On 32-bit Windows, this parameter is accepted but ignored.

The maximum length of the string is MQ_Q_NAME_LENGTH.

TriggerControl (MQCFIN)
Trigger control (parameter identifier: MQIA_TRIGGER_CONTROL).

Specifies whether trigger messages are written to the initiation queue.

The value may be:

MQTC_OFF
Trigger messages not required.

MQTC_ON
Trigger messages required.

This value is not supported on 32-bit Windows.

TriggerType (MQCFIN)
Trigger type (parameter identifier: MQIA_TRIGGER_TYPE).

Specifies the condition that initiates a trigger event. When the condition is true, a trigger message is sent to the initiation queue.

On 32-bit Windows, this parameter is accepted but ignored.

The value may be:

MQTT_NONE
No trigger messages.

MQTT_EVERY
Trigger message for every message.

MQTT_FIRST
Trigger message when queue depth goes from 0 to 1.

MQTT_DEPTH
Trigger message when depth threshold exceeded.

TriggerMsgPriority (MQCFIN)
Threshold message priority for triggers (parameter identifier: MQIA_TRIGGER_MSG_PRIORITY).

Specifies the minimum priority that a message must have before it can cause, or be counted for, a trigger event. The value must be in the range of priority values that are supported (0 through 9).

On 32-bit Windows, this parameter is accepted but ignored.

TriggerDepth (MQCFIN)
Trigger depth (parameter identifier: MQIA_TRIGGER_DEPTH).

Specifies (when *TriggerType* is MQTT_DEPTH) the number of messages that will initiate a trigger message to the initiation queue. The value must be in the range 1 through 999 999 999.

On 32-bit Windows, this parameter is accepted but ignored.

TriggerData (MQCFST)
Trigger data (parameter identifier: MQCA_TRIGGER_DATA).

Specifies user data that the queue manager includes in the trigger message. This data is made available to the monitoring application that processes the initiation queue and to the application that is started by the monitor.

On 32-bit Windows, this parameter is accepted but ignored.

The maximum length of the string is MQ_TRIGGER_DATA_LENGTH.

RemoteQName (MQCFST)
Name of remote queue as known locally on the remote queue manager (parameter identifier: MQCA_REMOTE_Q_NAME).

If this definition is used for a local definition of a remote queue, *RemoteQName* must not be blank when the open occurs.

If this definition is used for a queue-manager alias definition, *RemoteQName* must be blank when the open occurs.

If this definition is used for a reply-to alias, this name is the name of the queue that is to be the reply-to queue.

The maximum length of the string is MQ_Q_NAME_LENGTH.

RemoteQMgrName (MQCFST)
Name of remote queue manager (parameter identifier: MQCA_REMOTE_Q_MGR_NAME).

If an application opens the local definition of a remote queue, *RemoteQMgrName* must not be blank or the name of the connected queue manager. If *XmitQName* is blank there must be a local queue of this name, which is to be used as the transmission queue.

If this definition is used for a queue-manager alias, *RemoteQMgrName* is the name of the queue manager, which can be the name of the connected queue manager. Otherwise, if *XmitQName* is blank, when the queue is opened there must be a local queue of this name, which is to be used as the transmission queue.

If this definition is used for a reply-to alias, this name is the name of the queue manager that is to be the reply-to queue manager.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

XmitQName (MQCFST)
Transmission queue name (parameter identifier: MQCA_XMIT_Q_NAME).

Specifies the local name of the transmission queue to be used for messages destined for either a remote queue or for a queue-manager alias definition.

If *XmitQName* is blank, a queue with the same name as *RemoteQMgrName* is used as the transmission queue.

This attribute is ignored if the definition is being used as a queue-manager alias and *RemoteQMgrName* is the name of the connected queue manager.

It is also ignored if the definition is used as a reply-to queue alias definition.

The maximum length of the string is MQ_Q_NAME_LENGTH.

DefinitionType (MQCFIN)
Queue definition type (parameter identifier: MQIA_DEFINITION_TYPE).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

The value may be:

MQQDT_PERMANENT_DYNAMIC
Dynamically defined permanent queue.

MQQDT_TEMPORARY_DYNAMIC
Dynamically defined temporary queue.

Scope (MQCFIN)
Scope of the queue definition (parameter identifier: MQIA_SCOPE).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

Specifies whether the scope of the queue definition does not extend beyond the queue manager which owns the queue, or whether the queue name is contained in a cell directory, so that it is known to all of the queue managers within the cell.

If this attribute is changed from MQSCO_CELL to MQSCO_Q_MGR, the entry for the queue is deleted from the cell directory.

Model and dynamic queues cannot be changed to have cell scope.

If it is changed from MQSCO_Q_MGR to MQSCO_CELL, an entry for the queue is created in the cell directory. If there is already a queue with the same name in the cell directory, the command fails. The command also fails if no name service supporting a cell directory has been configured.

The value may be:

MQSCO_Q_MGR
Queue-manager scope.

MQSCO_CELL
Cell scope.

This value is not supported on OS/400 and 32-bit Windows.

QDepthHighLimit (MQCFIN)
High limit for queue depth (parameter identifier: MQIA_Q_DEPTH_HIGH_LIMIT).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

The threshold against which the queue depth is compared to generate a Queue Depth High event.

This event indicates that an application has put a message to a queue, and this has caused the number of messages on the queue to become greater than or equal to the queue depth high threshold. See the *QDepthHighEvent* parameter.

The value is expressed as a percentage of the maximum queue depth (*MaxQDepth* attribute), and must be greater than or equal to zero and less than or equal to 100.

QDepthLowLimit (MQCFIN)
Low limit for queue depth (parameter identifier: MQIA_Q_DEPTH_LOW_LIMIT).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

The threshold against which the queue depth is compared to generate a Queue Depth Low event.

This event indicates that an application has retrieved a message from a queue, and this has caused the number of messages on the queue to become less than or equal to the queue depth low threshold. See the *QDepthLowEvent* parameter.

The value is expressed as a percentage of the maximum queue depth (*MaxQDepth* attribute), and must be greater than or equal to zero and less than or equal to 100.

QDepthMaxEvent (MQCFIN)
Controls whether Queue Full events are generated (parameter identifier: MQIA_Q_DEPTH_MAX_EVENT).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

A Queue Full event indicates that an **MQPUT** call to a queue has been rejected because the queue is full, that is, the queue depth has already reached its maximum value.

Note: The value of this attribute can change implicitly. See Chapter 3, “Understanding performance events” on page 17.

The value may be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

QDepthHighEvent (MQCFIN)
Controls whether Queue Depth High events are generated (parameter identifier: MQIA_Q_DEPTH_HIGH_EVENT).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

A Queue Depth High event indicates that an application has put a message on a queue, and this has caused the number of messages on the queue to become greater than or equal to the queue depth high threshold. See the *QDepthHighLimit* parameter.

Note: The value of this attribute can change implicitly. See Chapter 3, “Understanding performance events” on page 17.

The value may be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

QDepthLowEvent (MQCFIN)
Controls whether Queue Depth Low events are generated (parameter identifier: MQIA_Q_DEPTH_LOW_EVENT).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

A Queue Depth Low event indicates that an application has retrieved a message from a queue, and this has caused the number of messages on the queue to become less than or equal to the queue depth low threshold. See the *QDepthLowLimit* parameter.

Note: The value of this attribute can change implicitly. See Chapter 3, "Understanding performance events" on page 17.

The value may be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

QServiceInterval (MQCFIN)
Target for queue service interval (parameter identifier: MQIA_Q_SERVICE_INTERVAL).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

The service interval used for comparison to generate Queue Service Interval High and Queue Service Interval OK events. See the *QServiceIntervalEvent* parameter.

The value is in units of milliseconds, and must be greater than or equal to zero, and less than or equal to 999 999 999.

QServiceIntervalEvent (MQCFIN)
Controls whether Service Interval High or Service Interval OK events are generated (parameter identifier: MQIA_Q_SERVICE_INTERVAL_EVENT).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

A Queue Service Interval High event is generated when a check indicates that no messages have been retrieved from or put to the queue for at least the time indicated by the *QServiceInterval* attribute.

A Queue Service Interval OK event is generated when a check indicates that a message has been retrieved from the queue within the time indicated by the *QServiceInterval* attribute.

Note: The value of this attribute can change implicitly. See Chapter 3, "Understanding performance events" on page 17.

The value may be:

MQQSIE_HIGH
Queue Service Interval High events enabled.

- Queue Service Interval High events are **enabled** and
- Queue Service Interval OK events are **disabled**.

MQQSIE_OK
Queue Service Interval OK events enabled.

- Queue Service Interval High events are **disabled** and
- Queue Service Interval OK events are **enabled**.

MQQSIE_NONE
No queue service interval events enabled.

- Queue Service Interval High events are **disabled** and
- Queue Service Interval OK events are also **disabled**.

Error codes

In addition to the values for any command shown on page 136, for this command the following may be returned in the response format header:

Reason (MQLONG)
The value may be:

MQRC_UNKNOWN_OBJECT_NAME
(2085, X'825') Unknown object name.

MQRCCF_ATTR_VALUE_ERROR
Attribute value not valid.

MQRCCF_CELL_DIR_NOT_AVAILABLE
Cell directory is not available.

MQRCCF_CFIN_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFIN_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIN_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_FORCE_VALUE_ERROR
Force value not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_DYNAMIC_Q_SCOPE_ERROR
Dynamic queue scope error.

MQRCCF_OBJECT_NAME_ERROR
Object name not valid.

MQRCCF_OBJECT_OPEN
Object is open.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_PARM_SEQUENCE_ERROR
Parameter sequence not valid.

MQRCCF_Q_ALREADY_IN_CELL
Queue already exists in cell.

MQRCCF_Q_TYPE_ERROR
Queue type not valid.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

Change Queue Manager

The Change Queue Manager (MQCMD_CHANGE_Q_MGR) command changes the specified attributes of the queue manager.

For any optional parameters that are omitted, the value does not change.

Required parameters:
None

Optional parameters:
Force, *QMgrDesc*, *TriggerInterval*, *DeadLetterQName*, *MaxHandles*, *MaxUncommittedMsgs*, *DefXmitQName*, *AuthorityEvent*, *InhibitEvent*, *LocalEvent*, *RemoteEvent*, *StartStopEvent*, *PerformanceEvent*, *MaxMsgLength*, *ChannelAutoDef*, *ChannelAutoDefEvent*, *ChannelAutoDefExit*

Optional parameters

Force (MQCFIN)
Force changes (parameter identifier: MQIACF_FORCE).

Specifies whether the command should be forced to complete if both of the following are true:

- *DefXmitQName* is specified, and
- An application has a remote queue open, the resolution for which would be affected by this change.

QMgrDesc (MQCFST)
Queue manager description (parameter identifier: MQCA_Q_MGR_DESC).

This is text that briefly describes the object.

The maximum length of the string is MQ_Q_MGR_DESC_LENGTH.

Use characters from the character set identified by the coded character set identifier (CCSID) for the queue manager on which the command is executing, to ensure that the text is translated correctly.

TriggerInterval (MQCFIN)
Trigger interval (parameter identifier: MQIA_TRIGGER_INTERVAL).

Specifies the trigger time interval, expressed in milliseconds, for use only with queues where *TriggerType* has a value of MQTT_FIRST.

In this case trigger messages are normally only generated when a suitable message arrives on the queue, and the queue was previously empty. Under certain circumstances, however, an additional trigger message can be generated with MQTT_FIRST triggering, even if the queue was not empty. These additional trigger messages are not generated more often than every *TriggerInterval* milliseconds.

Specify a value in the range 0 through 999 999 999.

On 32-bit Windows, this parameter is accepted but ignored.

DeadLetterQName (MQCFST)
Dead letter (undelivered message) queue name (parameter identifier: MQCA_DEAD_LETTER_Q_NAME).

Specifies the name of the local queue that is to be used for undelivered messages. Messages are put on this queue if they cannot be routed to their correct destination. The maximum length of the string is MQ_Q_NAME_LENGTH.

On 32-bit Windows, this parameter can be set only to blanks.

MaxHandles (MQCFIN)
Maximum number of handles (parameter identifier: MQIA_MAX_HANDLES).

The maximum number of handles that any one job can have open at the same time.

Specify a value in the range 0 through 999 999 999.

MaxUncommittedMsgs (MQCFIN)
Maximum uncommitted messages (parameter identifier: MQIA_MAX_UNCOMMITTED_MSGS).

Specifies the maximum number of uncommitted messages. That is:

- The number of messages that can be retrieved, plus
- The number of messages that can be put, plus
- Any trigger messages generated within this unit of work

under any one syncpoint. This limit does not apply to messages that are retrieved or put outside syncpoint.

Specify a value in the range 1 through 10 000.

DefXmitQName (MQCFST)
Default transmission queue name (parameter identifier: MQCA_DEF_XMIT_Q_NAME).

This is the name of the default transmission queue that is used for the transmission of messages to remote queue managers, if there is no other indication of which transmission queue to use.

The maximum length of the string is MQ_Q_NAME_LENGTH.

AuthorityEvent (MQCFIN)
Controls whether authorization (Not Authorized) events are generated (parameter identifier: MQIA_AUTHORITY_EVENT).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

The value may be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

On 32-bit Windows, this value is not supported.

InhibitEvent (MQCFIN)
Controls whether inhibit (Inhibit Get and Inhibit Put) events are generated (parameter identifier: MQIA_INHIBIT_EVENT).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

The value may be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

LocalEvent (MQCFIN)
Controls whether local error events are generated (parameter identifier: MQIA_LOCAL_EVENT).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

The value may be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

RemoteEvent (MQCFIN)
Controls whether remote error events are generated (parameter identifier: MQIA_REMOTE_EVENT).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

The value may be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

StartStopEvent (MQCFIN)
Controls whether start and stop events are generated (parameter identifier: MQIA_START_STOP_EVENT).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

The value may be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

PerformanceEvent (MQCFIN)
Controls whether performance-related events are generated (parameter identifier: MQIA_PERFORMANCE_EVENT).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

The value may be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

MaxMsgLength (MQCFIN)
Maximum message length (parameter identifier: MQIA_MAX_MSG_LENGTH).

Specifies the maximum length of messages allowed on queues on the queue manager. No message that is larger than either the queue's *MaxMsgLength* or the queue manager's *MaxMsgLength* can be put on a queue.

If you reduce the maximum message length for the queue manager, you should also reduce the maximum message length of the SYSTEM.DEFAULT.LOCAL.QUEUE definition, and your other queues, to ensure that the queue manager's limit is not less than that of any of the queues in the system. If you do not do this, and applications inquire only the value of the queue's *MaxMsgLength*, they may not work correctly.

The lower limit for this parameter is 32 KB (32 768 bytes). The upper limit depends on the environment:

- On AIX, HP-UX, OS/2, Sun Solaris, and Windows NT, the maximum message length is 100 MB (104 857 600 bytes).
- On OS/400, the maximum message length is 4 MB (4 194 304 bytes).

ChannelAutoDef (MQCFIN)
Controls whether receiver and server-connection channels can be auto-defined (parameter identifier: MQIA_CHANNEL_AUTO_DEF).

This parameter is supported in the following environments: AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

The value may be:

MQCHAD_DISABLED
Channel auto-definition disabled.

MQCHAD_ENABLED
Channel auto-definition enabled.

ChannelAutoDefEvent (MQCFIN)
Controls whether channel auto-definition events are generated (parameter identifier: MQIA_CHANNEL_AUTO_DEF_EVENT).

Relevant only if channel auto-definition is enabled (see *ChannelAutoDef*).

This parameter is supported in the following environments: AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

The value may be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

ChannelAutoDefExit (MQCFST)
Channel auto-definition exit name (parameter identifier: MQCA_CHANNEL_AUTO_DEF_EXIT).

If a nonblank name is defined, and channel auto-definition is enabled (see *ChannelAutoDef*), this exit is invoked when an inbound request for an undefined channel is received.

The format of the name is the same as for the *SecurityExit* parameter described in “Change Channel” on page 139.

The maximum length of the string is MQ_EXIT_NAME_LENGTH.

This parameter is supported in the following environments: AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

Error codes

In addition to the values for any command shown on page 136, for this command the following may be returned in the response format header:

Reason (MQLONG)
The value may be:

MQRCCF_ATTR_VALUE_ERROR
Attribute value not valid.

MQRCCF_CFIN_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFIN_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIN_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_CHAD_ERROR
Channel automatic definition error.

MQRCCF_CHAD_EVENT_ERROR
Channel automatic definition event error.

MQRCCF_CHAD_EVENT_WRONG_TYPE
Channel automatic definition event parameter not allowed for this channel type.

MQRCCF_CHAD_EXIT_ERROR
Channel automatic definition exit name error.

MQRCCF_CHAD_EXIT_WRONG_TYPE
Channel automatic definition exit parameter not allowed for this channel type.

MQRCCF_CHAD_WRONG_TYPE
: Channel automatic definition parameter not allowed for this channel type.

MQRCCF_FORCE_VALUE_ERROR
: Force value not valid.

MQRCCF_OBJECT_NAME_ERROR
: Object name not valid.

MQRCCF_OBJECT_OPEN
: Object is open.

MQRCCF_PARM_COUNT_TOO_BIG
: Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
: Parameter count too small.

MQRCCF_PARM_SEQUENCE_ERROR
: Parameter sequence not valid.

MQRCCF_STRUCTURE_TYPE_ERROR
: Structure type not valid.

MQRCCF_UNKNOWN_Q_MGR
: Queue manager not known.

Clear Queue

The Clear Queue (MQCMD_CLEAR_Q) command deletes all of the messages from a local queue.

The command fails if the queue contains uncommitted messages.

Required parameters:
QName

Optional parameters:
None

Required parameters

QName (MQCFST)
Queue name (parameter identifier: MQCA_Q_NAME).

The name of the local queue to be cleared. The maximum length of the string is MQ_Q_NAME_LENGTH.

Note: The target queue must be type local.

Error codes

In addition to the values for any command shown on page 136, for this command the following may be returned in the response format header:

Reason (MQLONG)
The value may be:

MQRC_Q_NOT_EMPTY
(2055, X'807') Queue contains one or more messages or uncommitted put or get requests.

(For this command this reason only occurs if there are uncommitted updates.)

MQRC_UNKNOWN_OBJECT_NAME
(2085, X'825') Unknown object name.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_OBJECT_OPEN
Object is open.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_Q_WRONG_TYPE
Action not valid for the queue of specified type.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

Copy Channel

The Copy Channel (MQCMD_COPY_CHANNEL) command creates a new channel definition using, for attributes not specified in the command, the attribute values of an existing channel definition.

Required parameters:

FromChannelName, *ToChannelName*, *ChannelType*

Optional parameters (any ChannelType):

Replace, *TransportType*, *ChannelDesc*, *SecurityExit*, *MsgExit*, *SendExit*, *ReceiveExit*, *MaxMsgLength*, *SecurityUserData*, *MsgUserData*, *SendUserData*, *ReceiveUserData*

Optional parameters (sender or server ChannelType):

ModeName, *TpName*, *ConnectionName*, *XmitQName*, *MCAName*, *BatchSize*, *DiscInterval*, *ShortRetryCount*, *ShortRetryInterval*, *LongRetryCount*, *LongRetryInterval*, *SeqNumberWrap*, *DataConversion*, *MCAType*, *MCAUserIdentifier*, *UserIdentifier*, *Password*, *HeartbeatInterval*, *NonPersistentMsgSpeed BatchInterval*

Optional parameters (receiver ChannelType):

BatchSize, *PutAuthority*, *SeqNumberWrap*, *MCAUserIdentifier*, *MsgRetryExit*, *MsgRetryUserData*, *MsgRetryCount*, *MsgRetryInterval*, *HeartbeatInterval*, *NonPersistentMsgSpeed*

Optional parameters (requester ChannelType):

ModeName, *TpName*, *ConnectionName*, *MCAName*, *BatchSize*, *PutAuthority*, *SeqNumberWrap*, *MCAType*, *MCAUserIdentifier*, *UserIdentifier*, *Password*, *MsgRetryExit*, *MsgRetryUserData*, *MsgRetryCount*, *MsgRetryInterval*, *HeartbeatInterval*, *NonPersistentMsgSpeed*

Optional parameters (server-connection ChannelType):

MCAUserIdentifier

Optional parameters (client-connection ChannelType):

ModeName, *TpName*, *QMgrName*, *ConnectionName*, *UserIdentifier*, *Password*

Required parameters

FromChannelName (MQCFST)

From channel name (parameter identifier: MQCACF_FROM_CHANNEL_NAME).

The name of the existing channel definition that contains values for the attributes that are not specified in this command.

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

ToChannelName (MQCFST)

To channel name (parameter identifier: MQCACF_TO_CHANNEL_NAME).

The name of the new channel definition.

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Channel names must be unique; if a channel definition with this name already exists, the value of *Replace* must be MQRP_YES. The channel type of the existing channel definition must be the same as the channel type of the new channel definition otherwise it cannot be replaced.

ChannelType (MQCFIN)
Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

Specifies the type of the channel being copied. The value may be:

MQCHT_SENDER
Sender.

MQCHT_SERVER
Server.

MQCHT_RECEIVER
Receiver.

MQCHT_REQUESTER
Requester.

MQCHT_SVRCONN
Server-connection (for use by clients).

This value is not supported in the following environment: 32-bit Windows.

MQCHT_CLNTCONN
Client connection.

This value is not supported in the following environments: OS/400, 32-bit Windows.

Optional parameters

Replace (MQCFIN)
Replace channel definition (parameter identifier: MQIACF_REPLACE).

The value may be:

MQRP_YES
Replace existing definition.

MQRP_NO
Do not replace existing definition.

TransportType (MQCFIN)
Transmission protocol type (parameter identifier: MQIACH_XMIT_PROTOCOL_TYPE).

No check is made that the correct transport type has been specified if the channel is initiated from the other end. The value may be:

MQXPT_LU62
LU 6.2.

This value is not supported on 32-bit Windows.

MQXPT_TCP
TCP/IP.

This is the *only* value supported on 32-bit Windows.

MQXPT_NETBIOS
NetBIOS.

This value is supported in the following environments: OS/2, 32-bit Windows, Windows NT.

MQXPT_SPX
SPX.

This value is supported in the following environments: OS/2, Windows NT, Windows client, DOS client.

MQXPT_DECNET
DECnet.

This value is supported in the following environment: OpenVMS.

ModeName (MQCFST)
Mode name (parameter identifier: MQCACH_MODE_NAME).

LU 6.2 mode name.

The maximum length of the string is MQ_MODE_NAME_LENGTH.

- On OpenVMS, OS/400, Tandem NSK, UNIX systems, and Windows NT, this can be set only to blanks. The actual name is taken instead from the CPI-C Communications Side Object or (on Windows NT) from the CPI-C symbolic destination name properties.
- On 32-bit Windows, this parameter is accepted but ignored.

This parameter is valid only for channels with a *TransportType* of MQXPT_LU62. It is not valid for receiver channels.

TpName (MQCFST)
Transaction program name (parameter identifier: MQCACH_TP_NAME).

LU 6.2 transaction program name.

The maximum length of the string is MQ_TP_NAME_LENGTH.

- On OpenVMS, OS/400, Tandem NSK, UNIX systems, and Windows NT, this can be set only to blanks. The actual name is taken instead from the CPI-C Communications Side Object or (on Windows NT) from the CPI-C symbolic destination name properties.
- On 32-bit Windows, this parameter is accepted but ignored.

This parameter is valid only for channels with a *TransportType* of MQXPT_LU62. It is not valid for receiver channels.

QMgrName (MQCFST)
Queue-manager name (parameter identifier: MQCA_Q_MGR_NAME).

For channels with a *ChannelType* of MQCHT_CLNTCONN, this is the name of a queue manager to which a client application can request connection.

On 32-bit Windows, this parameter is accepted but ignored.

For channels of other types, this parameter is not valid. The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

ChannelDesc (MQCFST)
Channel description (parameter identifier: MQCACH_DESC).

The maximum length of the string is MQ_CHANNEL_DESC_LENGTH.

Use characters from the character set identified by the coded character set identifier (CCSID) for the message queue manager on which the command is executing, to ensure that the text is translated correctly.

BatchSize (MQCFIN)
Batch size (parameter identifier: MQIACH_BATCH_SIZE).

The maximum number of messages that should be sent down a channel before a checkpoint is taken.

The batch size which is actually used is the lowest of the following:

- The *BatchSize* of the sending channel
- The *BatchSize* of the receiving channel
- The maximum number of uncommitted messages at the sending queue manager
- The maximum number of uncommitted messages at the receiving queue manager

The maximum number of uncommitted messages is specified by the *MaxUncommittedMsgs* parameter of the Change Queue Manager command.

Specify a value in the range one 1-9999.

This parameter is not valid for channels with a *ChannelType* of MQCHT_SVRCONN or MQCHT_CLNTCONN.

SecurityExit (MQCFST)
Security exit name (parameter identifier: MQCACH_SEC_EXIT_NAME).

If a nonblank name is defined, the security exit is invoked at the following times:

- Immediately after establishing a channel.

  Before any messages are transferred, the exit is given the opportunity to instigate security flows to validate connection authorization.

- Upon receipt of a response to a security message flow.

  Any security message flows received from the remote processor on the remote machine are passed to the exit.

The exit is given the entire application message and message descriptor for modification.

The format of the string depends on the platform, as follows:

- On UNIX systems, it is of the form

  `libraryname(functionname)`

- On OS/2, Windows NT, and Windows 3.1, it is of the form

  `dllname(functionname)`

  where *dllname* is specified without the suffix ".DLL".

- On OS/400, it is of the form

  `progname libname`

  where *progname* occupies the first 10 characters, and *libname* the second 10 characters (both blank-padded to the right if necessary).

- On OpenVMS, it is of the form

| `imagename(functionname)`

The maximum length of the string is MQ_EXIT_NAME_LENGTH. Set unused character positions to blanks.

MsgExit (MQCFST)
Message exit name (parameter identifier: MQCACH_MSG_EXIT_NAME).

If a nonblank name is defined, the exit is invoked immediately after a message has been retrieved from the transmission queue. The exit is given the entire application message and message descriptor for modification.

For channels with a *ChannelType* of MQCHT_SVRCONN or MQCHT_CLNTCONN, this parameter is not relevant, since message exits are not invoked for such channels.

The format of the string is the same as for *SecurityExit*.

The maximum length of the string is MQ_EXIT_NAME_LENGTH. Set unused character positions to blanks.

In the following environments, a list of exit names can be specified by using an MQCFSL structure instead of an MQCFST structure: AIX, | HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

- The exits are invoked in the order specified in the list.
- A list with only one name is equivalent to specifying a single name in an MQCFST structure.
- You cannot specify both a list (MQCFSL) and a single entry (MQCFST) structure for the same channel attribute.
- The total length of all of the exit names in the list (excluding trailing blanks in each name) must not exceed MQ_TOTAL_EXIT_NAME_LENGTH. An individual string must not exceed MQ_EXIT_NAME_LENGTH.

SendExit (MQCFST)
Send exit name (parameter identifier: MQCACH_SEND_EXIT_NAME).

If a nonblank name is defined, the exit is invoked immediately before data is sent out on the network. The exit is given the complete transmission buffer before it is transmitted; the contents of the buffer can be modified as required.

The format of the string is the same as for *SecurityExit*.

The maximum length of the string is MQ_EXIT_NAME_LENGTH. Set unused character positions to blanks.

In the following environments, a list of exit names can be specified by using an MQCFSL structure instead of an MQCFST structure: AIX, | HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

- The exits are invoked in the order specified in the list.
- A list with only one name is equivalent to specifying a single name in an MQCFST structure.
- You cannot specify both a list (MQCFSL) and a single entry (MQCFST) structure for the same channel attribute.

- The total length of all of the exit names in the list (excluding trailing blanks in each name) must not exceed MQ_TOTAL_EXIT_NAME_LENGTH. An individual string must not exceed MQ_EXIT_NAME_LENGTH.

ReceiveExit (MQCFST)
Receive exit name (parameter identifier: MQCACH_RCV_EXIT_NAME).

If a nonblank name is defined, the exit is invoked before data received from the network is processed. The complete transmission buffer is passed to the exit and the contents of the buffer can be modified as required.

The format of the string is the same as for *SecurityExit*.

The maximum length of the string is MQ_EXIT_NAME_LENGTH. Set unused character positions to blanks.

In the following environments, a list of exit names can be specified by using an MQCFSL structure instead of an MQCFST structure: AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

- The exits are invoked in the order specified in the list.
- A list with only one name is equivalent to specifying a single name in an MQCFST structure.
- You cannot specify both a list (MQCFSL) and a single entry (MQCFST) structure for the same channel attribute.
- The total length of all of the exit names in the list (excluding trailing blanks in each name) must not exceed MQ_TOTAL_EXIT_NAME_LENGTH. An individual string must not exceed MQ_EXIT_NAME_LENGTH.

SeqNumberWrap (MQCFIN)
Sequence wrap number (parameter identifier: MQIACH_SEQUENCE_NUMBER_WRAP).

Specifies the maximum message sequence number. When the maximum is reached, sequence numbers wrap to start again at 1.

The maximum message sequence number is not negotiable; the local and remote channels must wrap at the same number.

Specify a value in the range 100 through 999 999 999.

This parameter is not valid for channels with a *ChannelType* of MQCHT_SVRCONN or MQCHT_CLNTCONN.

MaxMsgLength (MQCFIN)
Maximum message length (parameter identifier: MQIACH_MAX_MSG_LENGTH).

Specifies the maximum message length that can be transmitted on the channel. This is compared with the value for the remote channel and the actual maximum is the lowest of the two values.

The value zero means the maximum message length for the queue manager.

The lower limit for this parameter is 0. The upper limit depends on the environment:

- On AIX, HP-UX, OS/2, Sun Solaris, and Windows NT, the maximum message length is 100 MB (104 857 600 bytes).
- On OpenVMS, OS/400, Tandem NSK, UNIX systems not listed above, and 32-bit Windows, the maximum message length is 4 MB (4 194 304 bytes).

SecurityUserData (MQCFST)
Security exit user data (parameter identifier: MQCACH_SEC_EXIT_USER_DATA).

Specifies user data that is passed to the security exit. The maximum length of the string is MQ_EXIT_DATA_LENGTH.

MsgUserData (MQCFST)
Message exit user data (parameter identifier: MQCACH_MSG_EXIT_USER_DATA).

Specifies user data that is passed to the message exit. The maximum length of the string is MQ_EXIT_DATA_LENGTH.

In the following environments, a list of exit user data strings can be specified by using an MQCFSL structure instead of an MQCFST structure: AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

- Each exit user data string is passed to the exit at the same ordinal position in the *MsgExit* list.
- A list with only one name is equivalent to specifying a single name in an MQCFST structure.
- You cannot specify both a list (MQCFSL) and a single entry (MQCFST) structure for the same channel attribute.
- The total length of all of the exit user data in the list (excluding trailing blanks in each string) must not exceed MQ_TOTAL_EXIT_DATA_LENGTH. An individual string must not exceed MQ_EXIT_DATA_LENGTH.

SendUserData (MQCFST)
Send exit user data (parameter identifier: MQCACH_SEND_EXIT_USER_DATA).

Specifies user data that is passed to the send exit. The maximum length of the string is MQ_EXIT_DATA_LENGTH.

In the following environments, a list of exit user data strings can be specified by using an MQCFSL structure instead of an MQCFST structure: AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

- Each exit user data string is passed to the exit at the same ordinal position in the *SendExit* list.
- A list with only one name is equivalent to specifying a single name in an MQCFST structure.
- You cannot specify both a list (MQCFSL) and a single entry (MQCFST) structure for the same channel attribute.

- The total length of all of the exit user data in the list (excluding trailing blanks in each string) must not exceed MQ_TOTAL_EXIT_DATA_LENGTH. An individual string must not exceed MQ_EXIT_DATA_LENGTH.

ReceiveUserData (MQCFST)
Receive exit user data (parameter identifier: MQCACH_RCV_EXIT_USER_DATA).

Specifies user data that is passed to the receive exit. The maximum length of the string is MQ_EXIT_DATA_LENGTH.

In the following environments, a list of exit user data strings can be specified by using an MQCFSL structure instead of an MQCFST structure:
| AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

- Each exit user data string is passed to the exit at the same ordinal position in the *ReceiveExit* list.
- A list with only one name is equivalent to specifying a single name in an MQCFST structure.
- You cannot specify both a list (MQCFSL) and a single entry (MQCFST) structure for the same channel attribute.
- The total length of all of the exit user data in the list (excluding trailing blanks in each string) must not exceed MQ_TOTAL_EXIT_DATA_LENGTH. An individual string must not exceed MQ_EXIT_DATA_LENGTH.

ConnectionName (MQCFST)
Connection name (parameter identifier: MQCACH_CONNECTION_NAME).

The maximum length of the string is MQ_CONN_NAME_LENGTH.

Specify the name of the machine as required for the stated *TransportType*:

- For MQXPT_LU62 on OS/2, specify the fully-qualified name of the partner LU. On OS/400, and UNIX systems, specify the name of the CPI-C communications side object. On Windows NT specify the CPI-C symbolic destination name.
- For MQXPT_TCP specify either the host name or the network address of the remote machine.
- For MQXPT_NETBIOS specify the NetBIOS station name.
- For MQXPT_SPX specify the 4-byte network address, the 6-byte node address, and the 2-byte socket number. These should be entered in hexadecimal, with a period separating the network and node addresses. The socket number should be enclosed in brackets, for example:

  ```
  CONNAME('0a0b0c0d.804abcde23a1(5e86)')
  ```

  If the socket number is omitted, the MQSeries default value (5e86 hex) is assumed.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_REQUESTER or MQCHT_CLNTCONN.

XmitQName (MQCFST)
Transmission queue name (parameter identifier: MQCACH_XMIT_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

A transmission queue name is required (either previously defined or specified here) if *ChannelType* is MQCHT_SENDER or MQCHT_SERVER. It is not valid for other channel types.

MCAName (MQCFST)
Message channel agent name (parameter identifier: MQCACH_MCA_NAME).

This is reserved, and if specified can be set only to blanks.

The maximum length of the string is MQ_MCA_NAME_LENGTH.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, or MQCHT_REQUESTER.

DiscInterval (MQCFIN)
Disconnection interval (parameter identifier: MQIACH_DISC_INTERVAL).

This defines the maximum number of seconds that the channel waits for messages to be put on a transmission queue before terminating the channel.

Specify a value in the range 0 through 999 999.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER or MQCHT_SERVER.

ShortRetryCount (MQCFIN)
Short retry count (parameter identifier: MQIACH_SHORT_RETRY).

The maximum number of attempts that are made by a sender or server channel to establish a connection to the remote machine, at intervals specified by *ShortRetryInterval* before the (normally longer) *LongRetryCount* and *LongRetryInterval* are used.

Retry attempts are made if the channel fails to connect initially (whether it is started automatically by the channel initiator or by an explicit command), and also if the connection fails after the channel has successfully connected. However, if the cause of the failure is such that retry is unlikely to be successful, retries are not attempted.

Specify a value in the range 0 through 999 999 999.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER or MQCHT_SERVER.

ShortRetryInterval (MQCFIN)
Short timer (parameter identifier: MQIACH_SHORT_TIMER).

Specifies the short retry wait interval for a sender or server channel that is started automatically by the channel initiator. It defines the interval in seconds between attempts to establish a connection to the remote machine.

The time is approximate; zero means that another connection attempt is made as soon as possible.

Specify a value in the range 0 through 999 999. Values exceeding this are treated as 999 999.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER or MQCHT_SERVER.

LongRetryCount (MQCFIN)
Long retry count (parameter identifier: MQIACH_LONG_RETRY).

When a sender or server channel is attempting to connect to the remote machine, and the count specified by *ShortRetryCount* has been exhausted, this specifies the maximum number of further attempts that are made to connect to the remote machine, at intervals specified by *LongRetryInterval*.

If this count is also exhausted without success, an error is logged to the operator, and the channel is stopped. The channel must subsequently be restarted with a command (it is not started automatically by the channel initiator), and it then makes only one attempt to connect, as it is assumed that the problem has now been cleared by the administrator. The retry sequence is not carried out again until after the channel has successfully connected.

Specify a value in the range 0 through 999 999 999.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER or MQCHT_SERVER.

LongRetryInterval (MQCFIN)
Long timer (parameter identifier: MQIACH_LONG_TIMER).

Specifies the long retry wait interval for a sender or server channel that is started automatically by the channel initiator. It defines the interval in seconds between attempts to establish a connection to the remote machine, after the count specified by *ShortRetryCount* has been exhausted.

The time is approximate; zero means that another connection attempt is made as soon as possible.

Specify a value in the range 0 through 999 999. Values exceeding this are treated as 999 999.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER or MQCHT_SERVER.

DataConversion (MQCFIN)
Whether sender should convert application data (parameter identifier: MQIACH_DATA_CONVERSION).

This parameter is valid only for *ChannelType* values of MQCHT_SENDER or MQCHT_SERVER.

The value may be:

MQCDC_NO_SENDER_CONVERSION
No conversion by sender.

MQCDC_SENDER_CONVERSION
Conversion by sender.

This value is not supported on 32-bit Windows.

PutAuthority (MQCFIN)
Put authority (parameter identifier: MQIACH_PUT_AUTHORITY).

Specifies whether the user identifier in the context information associated with a message should be used to establish authority to put the message on the destination queue.

This parameter is valid only for channels with a *ChannelType* value of MQCHT_RECEIVER or MQCHT_REQUESTER The value may be:

MQPA_DEFAULT
Default user identifier is used.

MQPA_CONTEXT
Context user identifier is used.

MCAType (MQCFIN)
Message channel agent type (parameter identifier: MQIACH_MCA_TYPE).

Specifies the type of the message channel agent program.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, or MQCHT_REQUESTER.

The value may be:

MQMCAT_PROCESS
Process.

MQMCAT_THREAD
Thread (OS/2 and Windows NT only).

MCAUserIdentifier (MQCFST)
Message channel agent user identifier (parameter identifier: MQCACH_MCA_USER_ID).

If this is nonblank, it is the user identifier which is to be used by the message channel agent for authorization to access MQ resources, including (if *PutAuthority* is MQPA_DEFAULT) authorization to put the message to the destination queue for receiver or requester channels.

If it is blank, the message channel agent uses its default user identifier.

This user identifier can be overridden by one supplied by a channel security exit.

This parameter is not valid for channels with a *ChannelType* of MQCHT_CLNTCONN.

The maximum length of the string is MQ_USER_ID_LENGTH.

UserIdentifier (MQCFST)
Task user identifier (parameter identifier: MQCACH_USER_ID).

This is used by the message channel agent when attempting to initiate a secure SNA session with a remote message channel agent.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_REQUESTER or MQCHT_CLNTCONN.

- This parameter is supported in the following environments: OpenVMS, OS/2, Tandem NSK, UNIX systems.

- On 32-bit Windows, the parameter is accepted but ignored.

The maximum length of the string is MQ_USER_ID_LENGTH. However, only the first 10 characters are used.

Password (MQCFST)
Password (parameter identifier: MQCACH_PASSWORD).

This is used by the message channel agent when attempting to initiate a secure SNA session with a remote message channel agent.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_REQUESTER or MQCHT_CLNTCONN.

- This parameter is supported in the following environments: OpenVMS, OS/2, Tandem NSK, UNIX systems.
- On 32-bit Windows, the parameter is accepted but ignored.

The maximum length of the string is MQ_PASSWORD_LENGTH. However, only the first 10 characters are used.

MsgRetryExit (MQCFST)
Message retry exit name (parameter identifier: MQCACH_MR_EXIT_NAME).

- This parameter is supported in the following environments: AIX, AT&T GIS UNIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.
- On 32-bit Windows, the parameter is accepted but must be blank.

If a nonblank name is defined, the exit is invoked prior to performing a wait before retrying a failing message.

The format of the string is the same as for *SecurityExit*.

The maximum length of the string is MQ_EXIT_NAME_LENGTH. Set unused character positions to blanks.

This parameter is valid only for *ChannelType* values of MQCHT_RECEIVER or MQCHT_REQUESTER.

MsgRetryUserData (MQCFST)
Message retry exit user data (parameter identifier: MQCACH_MR_EXIT_USER_DATA).

- This parameter is supported in the following environments: AIX, AT&T GIS UNIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.
- On 32-bit Windows, the parameter is accepted but ignored.

Specifies user data that is passed to the message retry exit.

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

This parameter is valid only for *ChannelType* values of MQCHT_RECEIVER or MQCHT_REQUESTER.

MsgRetryCount (MQCFIN)
Message retry count (parameter identifier: MQIACH_MR_COUNT).

- This parameter is supported in the following environments: AIX, AT&T GIS UNIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

- On 32-bit Windows, the parameter is accepted but must be zero.

Specifies the number of times that a failing message should be retried.

Specify a value in the range 0 through 999 999 999.

This parameter is valid only for *ChannelType* values of MQCHT_RECEIVER or MQCHT_REQUESTER.

MsgRetryInterval (MQCFIN)
Message retry interval (parameter identifier: MQIACH_MR_INTERVAL).

- This parameter is supported in the following environments: AIX, AT&T GIS UNIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.
- On 32-bit Windows, the parameter is accepted but must be zero.

Specifies the minimum time interval in milliseconds between retries of failing messages.

Specify a value in the range 0 through 999 999 999.

This parameter is valid only for *ChannelType* values of MQCHT_RECEIVER or MQCHT_REQUESTER.

HeartbeatInterval (MQCFIN)
Heartbeat interval (parameter identifier: MQIACH_HB_INTERVAL).

The interpretation of this parameter depends on the channel type, as follows:

- For a channel type of MQCHT_SENDER, MQCHT_SERVER, MQCHT_RECEIVER or MQCHT_REQUESTER, this is the time in seconds between heartbeat flows passed from the sending MCA when there are no messages on the transmission queue. This gives the receiving MCA the opportunity to quiesce the channel. To be useful, *HeartbeatInterval* should be significantly less than *DiscInterval*. However, the only check is that the value is within the permitted range.

  This type of heartbeat is supported in the following environments: AIX, HP-UX, MVS/ESA, OS/2, OS/400, Sun Solaris, Windows NT.

- For a channel type of MQCHT_CLNTCONN or MQCHT_SVRCONN, this is the time in seconds between heartbeat flows passed from the server MCA when that MCA has issued an MQGET call with the MQGMO_WAIT option on behalf of a client application. This allows the server MCA to handle situations where the client connection fails during an MQGET with MQGMO_WAIT.

  This type of heartbeat is supported in the following environments: AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

The value must be in the range 0 through 999 999. A value of 0 means that no heartbeat exchange occurs. The value that is actually used is the larger of the values specified at the sending side and receiving side.

NonPersistentMsgSpeed (MQCFIN)
Speed at which nonpersistent messages are to be sent (parameter identifier: MQIACH_NPM_SPEED).

This parameter is supported in the following environments: AIX, HP-UX, OS/2, OS/400, Sun Solaris, 32-bit Windows, Windows NT.

Specifying MQNPMS_FAST means that nonpersistent messages on a channel need not wait for a syncpoint before being made available for retrieval. The advantage of this is that nonpersistent messages become available for retrieval far more quickly. The disadvantage is that because they do not wait for a syncpoint, they may be lost if there is a transmission failure.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_RECEIVER or MQCHT_REQUESTER. The value may be:

MQNPMS_NORMAL
Normal speed.

MQNPMS_FAST
Fast speed.

BatchInterval (MQCFIN)
Batch interval (parameter identifier: MQIACH_BATCH_INTERVAL).

This is the approximate time in milliseconds that a channel will keep a batch open, if fewer than *BatchSize* messages have been transmitted in the current batch.

This parameter is supported in the following environments: AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

If *BatchInterval* is greater than zero, the batch is terminated by whichever of the following occurs first:

- *BatchSize* messages have been sent, or
- *BatchInterval* milliseconds have elapsed since the start of the batch.

If *BatchInterval* is zero, the batch is terminated by whichever of the following occurs first:

- *BatchSize* messages have been sent, or
- The transmission queue becomes empty.

BatchInterval must be in the range 0 through 999 999 999.

This parameter applies only to channels with a *ChannelType* of:

MQCHT_SENDER
MQCHT_SERVER

Error codes

In addition to the values for any command shown on page 136, for this command the following may be returned in the response format header:

Reason (MQLONG)
The value may be:

MQRCCF_BATCH_INT_ERROR
Batch interval not valid.

MQRCCF_BATCH_INT_WRONG_TYPE
Batch interval parameter not allowed for this channel type.

MQRCCF_BATCH_SIZE_ERROR
Batch size not valid.

MQRCCF_CFIN_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFIN_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIN_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFSL_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFSL_TOTAL_LENGTH_ERROR
Total string length error.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_CHANNEL_ALREADY_EXISTS
Channel already exists.

MQRCCF_CHANNEL_NAME_ERROR
Channel name error.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_CHANNEL_TYPE_ERROR
Channel type not valid.

MQRCCF_CONN_NAME_ERROR
Error in connection name parameter.

MQRCCF_DISC_INT_ERROR
Disconnection interval not valid.

MQRCCF_DISC_INT_WRONG_TYPE
Disconnection interval not allowed for this channel type.

MQRCCF_HB_INTERVAL_ERROR
Heartbeat interval not valid.

MQRCCF_HB_INTERVAL_WRONG_TYPE
Heartbeat interval parameter not allowed for this channel type.

MQRCCF_LONG_RETRY_ERROR
Long retry count not valid.

MQRCCF_LONG_RETRY_WRONG_TYPE
Long retry parameter not allowed for this channel type.

MQRCCF_LONG_TIMER_ERROR
Long timer not valid.

MQRCCF_LONG_TIMER_WRONG_TYPE
Long timer parameter not allowed for this channel type.

MQRCCF_MAX_MSG_LENGTH_ERROR
Maximum message length not valid.

MQRCCF_MCA_NAME_ERROR
Message channel agent name error.

MQRCCF_MCA_NAME_WRONG_TYPE
Message channel agent name not allowed for this channel type.

MQRCCF_MCA_TYPE_ERROR
Message channel agent type not valid.

MQRCCF_MISSING_CONN_NAME
Connection name parameter required but missing.

MQRCCF_MR_COUNT_ERROR
Message retry count not valid.

MQRCCF_MR_COUNT_WRONG_TYPE
Message-retry count parameter not allowed for this channel type.

MQRCCF_MR_EXIT_NAME_ERROR
Channel message-retry exit name error.

MQRCCF_MR_EXIT_NAME_WRONG_TYPE
Message-retry exit parameter not allowed for this channel type.

MQRCCF_MR_INTERVAL_ERROR
Message retry interval not valid.

MQRCCF_MR_INTERVAL_WRONG_TYPE
Message-retry interval parameter not allowed for this channel type.

MQRCCF_MSG_EXIT_NAME_ERROR
Channel message exit name error.

MQRCCF_NPM_SPEED_ERROR
Nonpersistent message speed not valid.

MQRCCF_NPM_SPEED_WRONG_TYPE
Nonpersistent message speed parameter not allowed for this channel type.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_PARM_SEQUENCE_ERROR
Parameter sequence not valid.

MQRCCF_PUT_AUTH_ERROR
Put authority value not valid.

MQRCCF_PUT_AUTH_WRONG_TYPE
Put authority parameter not allowed for this channel type.

MQRCCF_RCV_EXIT_NAME_ERROR
Channel receive exit name error.

MQRCCF_REPLACE_VALUE_ERROR
Replace value not valid.

MQRCCF_SEC_EXIT_NAME_ERROR
Channel security exit name error.

MQRCCF_SEND_EXIT_NAME_ERROR
Channel send exit name error.

MQRCCF_SEQ_NUMBER_WRAP_ERROR
Sequence wrap number not valid.

MQRCCF_SHORT_RETRY_ERROR
Short retry count not valid.

MQRCCF_SHORT_RETRY_WRONG_TYPE
Short retry parameter not allowed for this channel type.

MQRCCF_SHORT_TIMER_ERROR
Short timer value not valid.

MQRCCF_SHORT_TIMER_WRONG_TYPE
Short timer parameter not allowed for this channel type.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

MQRCCF_XMIT_PROTOCOL_TYPE_ERR
Transmission protocol type not valid.

MQRCCF_XMIT_Q_NAME_ERROR
Transmission queue name error.

MQRCCF_XMIT_Q_NAME_WRONG_TYPE
Transmission queue name not allowed for this channel type.

Copy Process

The Copy Process (MQCMD_COPY_PROCESS) command creates a new MQSeies process definition, using, for attributes not specified in the command, the attribute values of an existing process definition.

| *This PCF is not supported if you are using MQSeries for Windows Version 2.1.*

Required parameters:

FromProcessName, *ToProcessName*

Optional parameters:

Replace, *ProcessDesc*, *ApplType*, *ApplId*, *EnvData*, *UserData*

Required parameters

FromProcessName (MQCFST)

The name of the process definition to be copied from (parameter identifier: MQCACF_FROM_PROCESS_NAME).

Specifies the name of the existing process definition that contains values for the attributes not specified in this command.

The maximum length of the string is MQ_PROCESS_NAME_LENGTH.

ToProcessName (MQCFST)

To process name (parameter identifier: MQCACF_TO_PROCESS_NAME).

The name of the new process definition. If a process definition with this name already exists, *Replace* must be specified as MQRP_YES.

The maximum length of the string is MQ_PROCESS_NAME_LENGTH.

Optional parameters

Replace (MQCFIN)

Replace attributes (parameter identifier: MQIACF_REPLACE).

If a process definition with the same name as *ToProcessName* already exists, this specifies whether it is to be replaced.

The value may be:

MQRP_YES
: Replace existing definition.

MQRP_NO
: Do not replace existing definition.

ProcessDesc (MQCFST)

Description of process definition (parameter identifier: MQCA_PROCESS_DESC).

A plain-text comment that provides descriptive information about the process definition. It should contain only displayable characters.

The maximum length of the string is MQ_PROCESS_DESC_LENGTH.

If characters that are not in the coded character set identifier (CCSID) for the queue manager on which the command is executing are used, they may be translated incorrectly.

ApplType (MQCFIN)
Application type (parameter identifier: MQIA_APPL_TYPE).

Valid application types are:

MQAT_OS400
OS/400 application.

MQAT_OS2
OS/2 or Presentation Manager application.

MQAT_WINDOWS_NT
Windows NT or 32-bit Windows application.

MQAT_DOS
DOS client application.

MQAT_WINDOWS
Windows client or 16-bit Windows application.

MQAT_UNIX
UNIX application.

MQAT_AIX
AIX application (same value as MQAT_UNIX).

MQAT_CICS
CICS transaction.

| MQAT_VMS
| OpenVMS application.

| MQAT_NSK
| Tandem NSK application.

MQAT_DEFAULT
Default application type.

user-value: User-defined application type in the range 65 536 through 999 999 999 (not checked).

Only application types (other than user-defined types) that are supported on the platform at which the command is executed should be used:

| • On OpenVMS:
|
| MQAT_VMS (default),
| MQAT_DOS,
| MQAT_WINDOWS, and
| MQAT_DEFAULT are supported.

• On OS/400:

MQAT_OS400 (default),
MQAT_CICS, and
MQAT_DEFAULT are supported.

• On OS/2:

MQAT_OS2 (default),
MQAT_DOS,

MQAT_WINDOWS,
MQAT_AIX,
MQAT_CICS, and
MQAT_DEFAULT are supported.

- On Tandem NSK:

  MQAT_NSK (default),
  MQAT_DOS,
  MQAT_WINDOWS, and
  MQAT_DEFAULT are supported.

- On Windows NT:

  MQAT_WINDOWS_NT (default),
  MQAT_OS2
  MQAT_DOS,
  MQAT_WINDOWS,
  MQAT_CICS, and
  MQAT_DEFAULT are supported.

- On UNIX systems:

  MQAT_UNIX (default),
  MQAT_OS2,
  MQAT_DOS,
  MQAT_WINDOWS,
  MQAT_CICS, and
  MQAT_DEFAULT are supported.

ApplId (MQCFST)
Application identifier (parameter identifier: MQCA_APPL_ID).

This is the name of the application to be started, on the platform for which the command is executing, and might typically be a program name and library name.

The maximum length of the string is MQ_PROCESS_APPL_ID_LENGTH.

EnvData (MQCFST)
Environment data (parameter identifier: MQCA_ENV_DATA).

A character string that contains environment information pertaining to the application to be started.

The maximum length of the string is MQ_PROCESS_ENV_DATA_LENGTH.

UserData (MQCFST)
User data (parameter identifier: MQCA_USER_DATA).

A character string that contains user information pertaining to the application (defined by *ApplId*) that is to be started.

The maximum length of the string is MQ_PROCESS_USER_DATA_LENGTH.

Error codes

In addition to the values for any command shown on page 136, for this command the following may be returned in the response format header:

Reason (MQLONG)
The value may be:

MQRC_UNKNOWN_OBJECT_NAME
(2085, X'825') Unknown object name.

MQRCCF_ATTR_VALUE_ERROR
Attribute value not valid.

MQRCCF_CFIN_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFIN_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIN_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_OBJECT_ALREADY_EXISTS
Object already exists.

MQRCCF_OBJECT_NAME_ERROR
Object name not valid.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_PARM_SEQUENCE_ERROR
Parameter sequence not valid.

MQRCCF_REPLACE_VALUE_ERROR
Replace value not valid.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

Copy Queue

The Copy Queue (MQCMD_COPY_Q) command creates a new queue definition, of the same type, using, for attributes not specified in the command, the attribute values of an existing queue definition.

Required parameters:
FromQName, *ToQName*, *QType*

Optional parameters (any QType):
Replace, *QDesc*, *InhibitPut*, *DefPriority*, *DefPersistence*

Optional parameters (alias QType):
InhibitGet, *BaseQName*, *Scope*

Optional parameters (local QType):
InhibitGet, *ProcessName*, *MaxQDepth*, *MaxMsgLength*, *BackoutThreshold*, *BackoutRequeueName*, *Shareability*, *DefInputOpenOption*, *HardenGetBackout*, *MsgDeliverySequence*, *RetentionInterval*, *DistLists*, *Usage*, *InitiationQName*, *TriggerControl*, *TriggerType*, *TriggerMsgPriority*, *TriggerDepth*, *TriggerData*, *Scope*, *QDepthHighLimit*, *QDepthLowLimit*, *QDepthMaxEvent*, *QDepthHighEvent*, *QDepthLowEvent*, *QServiceInterval*, *QServiceIntervalEvent*

Optional parameters (remote QType):
RemoteQName, *RemoteQMgrName*, *XmitQName*, *Scope*

Optional parameters (model QType):
InhibitGet, *ProcessName*, *MaxQDepth*, *MaxMsgLength*, *BackoutThreshold*, *BackoutRequeueName*, *Shareability*, *DefInputOpenOption*, *HardenGetBackout*, *MsgDeliverySequence*, *RetentionInterval*, *DistLists*, *Usage*, *InitiationQName*, *TriggerControl*, *TriggerType*, *TriggerMsgPriority*, *TriggerDepth*, *TriggerData*, *DefinitionType*, *QDepthHighLimit*, *QDepthLowLimit*, *QDepthMaxEvent*, *QDepthHighEvent*, *QDepthLowEvent*, *QServiceInterval*, *QServiceIntervalEvent*

Required parameters

FromQName (MQCFST)
From queue name (parameter identifier: MQCACF_FROM_Q_NAME).

Specifies the name of the existing queue definition.

The maximum length of the string is MQ_Q_NAME_LENGTH.

ToQName (MQCFST)
To queue name (parameter identifier: MQCACF_TO_Q_NAME).

Specifies the name of the new queue definition.

The maximum length of the string is MQ_Q_NAME_LENGTH.

Queue names must be unique; if a queue definition already exists with the name and type of the new queue, *Replace* must be specified as MQRP_YES. If a queue definition exists with the same name as and a different type from the new queue, the command will fail.

QType (MQCFIN)
Queue type (parameter identifier: MQIA_Q_TYPE).

The value specified must match the type of the queue being copied.

The value may be:

MQQT_ALIAS
Alias queue definition.

MQQT_LOCAL
Local queue.

MQQT_REMOTE
Local definition of a remote queue.

The following is supported on all platforms, but on OS/400 for receipt by MQSeries for AS/400 V4R2:

MQQT_MODEL
Model queue definition.

Optional parameters

Replace (MQCFIN)
Replace attributes (parameter identifier: MQIACF_REPLACE).

If the object already exists, the effect is similar to issuing the Change Queue command without the MQFC_YES option on the *Force* parameter, and with *all* of the other attributes specified. In particular, note that any messages which are on the existing queue are retained.

(The difference between the Change Queue command without MQFC_YES on the *Force* parameter, and the Copy Queue command with MQRP_YES on the *Replace* parameter, is that the Change Queue command does not change unspecified attributes, but Copy Queue with MQRP_YES sets *all* the attributes. When you use MQRP_YES, unspecified attributes are taken from the queue specified by *FromQName*, and the existing attributes of the object being replaced, if one exists, are ignored.)

The command fails if both of the following are true:

- The command sets attributes that would require the use of MQFC_YES on the *Force* parameter if you were using the Change Queue command
- The object is open

The Change Queue command with MQFC_YES on the *Force* parameter succeeds in this situation.

If MQSCO_CELL is specified on the *Scope* parameter on OS/2 or UNIX systems, and there is already a queue with the same name in the cell directory, the command fails, whether or not MQRP_YES is specified.

The value may be:

MQRP_YES
Replace existing definition.

MQRP_NO
Do not replace existing definition.

QDesc (MQCFST)
Queue description (parameter identifier: MQCA_Q_DESC).

Text that briefly describes the object. The maximum length of the string is MQ_Q_DESC_LENGTH.

Use characters from the character set identified by the coded character set identifier (CCSID) for the queue manager on which the command is executing to ensure that the text is translated correctly.

InhibitPut (MQCFIN)
Whether put operations are allowed (parameter identifier: MQIA_INHIBIT_PUT).

Specifies whether messages can be put on the queue.

The value may be:

MQQA_PUT_ALLOWED
Put operations are allowed.

MQQA_PUT_INHIBITED
Put operations are inhibited.

DefPriority (MQCFIN)
Default priority (parameter identifier: MQIA_DEF_PRIORITY).

Specifies the default priority of messages put on the queue. The value must be in the range zero through to the maximum priority value that is supported (this is 9).

DefPersistence (MQCFIN)
Default persistence (parameter identifier: MQIA_DEF_PERSISTENCE).

Specifies the default for message-persistence on the queue. Message persistence determines whether or not messages are preserved across restarts of the queue manager.

The value may be:

MQPER_PERSISTENT
Message is persistent.

MQPER_NOT_PERSISTENT
Message is not persistent.

InhibitGet (MQCFIN)
Whether get operations are allowed (parameter identifier: MQIA_INHIBIT_GET).

The value may be:

MQQA_GET_ALLOWED
Get operations are allowed.

MQQA_GET_INHIBITED
Get operations are inhibited.

BaseQName (MQCFST)
Queue name to which the alias resolves (parameter identifier: MQCA_BASE_Q_NAME).

This is the name of a local or remote queue that is defined to the local queue manager. The maximum length of the string is MQ_Q_NAME_LENGTH.

ProcessName (MQCFST)
Name of process definition for the queue (parameter identifier: MQCA_PROCESS_NAME).

Specifies the local name of the MQSeries process that identifies the application that should be started when a trigger event occurs.

- On AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT, if the queue is a transmission queue the process name can be left as all blanks.
- On 32-bit Windows, this parameter is accepted but ignored.
- In other environments, the process name must be nonblank for a trigger event to occur (although it can be set after the queue has been created).

The maximum length of the string is MQ_PROCESS_NAME_LENGTH.

MaxQDepth (MQCFIN)
Maximum queue depth (parameter identifier: MQIA_MAX_Q_DEPTH).

The maximum number of messages allowed on the queue. Note that other factors may cause the queue to be treated as full; for example, it will be appear to be full if there is no storage available for a message.

Specify a value in the range 0 through 640 000.

MaxMsgLength (MQCFIN)
Maximum message length (parameter identifier: MQIA_MAX_MSG_LENGTH).

Specifies the maximum length for messages on the queue. Because applications may use the value of this attribute to determine the size of buffer they need to retrieve messages from the queue, the value should be changed only if it is known that this will not cause an application to operate incorrectly.

You are recommended not to set a value that is greater than the queue manager's *MaxMsgLength* attribute.

The lower limit for this parameter is 0. The upper limit depends on the environment:

- On AIX, HP-UX, OS/2, Sun Solaris, and Windows NT, the maximum message length is 100 MB (104 857 600 bytes).
- On OpenVMS, OS/400, Tandem NSK, UNIX systems not listed above, and 32-bit Windows, the maximum message length is 4 MB (4 194 304 bytes).

BackoutThreshold (MQCFIN)
Backout threshold (parameter identifier: MQIA_BACKOUT_THRESHOLD).

That is, the number of times a message can be backed out before it is transferred to the backout queue specified by *BackoutRequeueName*.

If the value is subsequently reduced, any messages already on the queue that have been backed out at least as many times as the new value remain on the queue, but such messages are transferred if they are backed out again.

Specify a value in the range 0 through 999 999 999.

BackoutRequeueName (MQCFST)
Excessive backout requeue name (parameter identifier: MQCA_BACKOUT_REQ_Q_NAME).

Specifies the local name of the queue (not necessarily a local queue) to which a message is transferred if it is backed out more times than the value of *BackoutThreshold*.

The backout queue does not need to exist at this time but it must exist when the *BackoutThreshold* value is exceeded.

The maximum length of the string is MQ_Q_NAME_LENGTH.

Shareability (MQCFIN)
Whether queue can be shared (parameter identifier: MQIA_SHAREABILITY).

Specifies whether multiple instances of applications, can open this queue for input.

The value may be:

MQQA_SHAREABLE
Queue is shareable.

MQQA_NOT_SHAREABLE
Queue is not shareable.

DefInputOpenOption (MQCFIN)
Default input open option (parameter identifier: MQIA_DEF_INPUT_OPEN_OPTION).

Specifies the default share option for applications opening this queue for input.

The value may be:

MQOO_INPUT_EXCLUSIVE
Open queue to get messages with exclusive access.

MQOO_INPUT_SHARED
Open queue to get messages with shared access.

HardenGetBackout (MQCFIN)
Whether to harden backout count (parameter identifier: MQIA_HARDEN_GET_BACKOUT).

Specifies whether the count of backed out messages should be saved (hardened) across restarts of the queue manager.

Note: MQSeries for AS/400 always hardens the count, regardless of the setting of this attribute.

The value may be:

MQQA_BACKOUT_HARDENED
Backout count remembered.

MQQA_BACKOUT_NOT_HARDENED
Backout count may not be remembered.

MsgDeliverySequence (MQCFIN)
Whether priority is relevant (parameter identifier: MQIA_MSG_DELIVERY_SEQUENCE).

The value may be:

MQMDS_PRIORITY
Messages are returned in priority order.

MQMDS_FIFO
Messages are returned in FIFO order (first in, first out).

RetentionInterval (MQCFIN)
Retention interval (parameter identifier: MQIA_RETENTION_INTERVAL).

The number of hours for which the queue may be needed, based on the date and time when the queue was created.

This information is available to a housekeeping application or an operator and may be used to determine when a queue is no longer required. The queue manager does not delete queues nor does it prevent queues from being deleted if their retention interval has not expired. It is the user's responsibility to take any required action.

Specify a value in the range 0 through 999 999 999.

DistLists (MQCFIN)
Distribution list support (parameter identifier: MQIA_DIST_LISTS).

Specifies whether distribution-list messages can be placed on the queue.

Note: This attribute is set by the sending message channel agent (MCA) which removes messages from the queue; this happens each time the sending MCA establishes a connection to a receiving MCA on a partnering queue manager. The attribute should not normally be set by administrators, although it can be set if the need arises.

This parameter is supported in the following environments: AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

The value may be:

MQDL_SUPPORTED
Distribution lists supported.

MQDL_NOT_SUPPORTED
Distribution lists not supported.

Usage (MQCFIN)
Usage (parameter identifier: MQIA_USAGE).

Specifies whether the queue is for normal usage or for transmitting messages to a remote queue manager.

The value may be:

MQUS_NORMAL
Normal usage.

MQUS_TRANSMISSION
Transmission queue.

InitiationQName (MQCFST)
Initiation queue name (parameter identifier: MQCA_INITIATION_Q_NAME).

The local queue for trigger messages relating to the new, or changed, queue. The initiation queue must be on the same queue manager.

On 32-bit Windows, this parameter is accepted but ignored.

The maximum length of the string is MQ_Q_NAME_LENGTH.

TriggerControl (MQCFIN)
Trigger control (parameter identifier: MQIA_TRIGGER_CONTROL).

Specifies whether trigger messages are written to the initiation queue.

The value may be:

MQTC_OFF
Trigger messages not required.

MQTC_ON
Trigger messages required.

This value is not supported on 32-bit Windows.

TriggerType (MQCFIN)
Trigger type (parameter identifier: MQIA_TRIGGER_TYPE).

Specifies the condition that initiates a trigger event. When the condition is true, a trigger message is sent to the initiation queue.

On 32-bit Windows, this parameter is accepted but ignored.

The value may be:

MQTT_NONE
No trigger messages.

MQTT_EVERY
Trigger message for every message.

MQTT_FIRST
Trigger message when queue depth goes from 0 to 1.

MQTT_DEPTH
Trigger message when depth threshold exceeded.

TriggerMsgPriority (MQCFIN)
Threshold message priority for triggers (parameter identifier: MQIA_TRIGGER_MSG_PRIORITY).

Specifies the minimum priority that a message must have before it can cause, or be counted for, a trigger event. The value must be in the range of priority values that are supported (0 through 9).

On 32-bit Windows, this parameter is accepted but ignored.

TriggerDepth (MQCFIN)
Trigger depth (parameter identifier: MQIA_TRIGGER_DEPTH).

Specifies (when *TriggerType* is MQTT_DEPTH) the number of messages that will initiate a trigger message to the initiation queue.

Specify a value in the range 1 through 999 999 999.

On 32-bit Windows, this parameter is accepted but ignored.

TriggerData (MQCFST)
Trigger data (parameter identifier: MQCA_TRIGGER_DATA).

Specifies user data that the queue manager includes in the trigger message. This data is made available to the monitoring application that processes the initiation queue and to the application that is started by the monitor.

On 32-bit Windows, this parameter is accepted but ignored.

The maximum length of the string is MQ_TRIGGER_DATA_LENGTH.

RemoteQName (MQCFST)
Name of remote queue as known locally on the remote queue manager (parameter identifier: MQCA_REMOTE_Q_NAME).

If this definition is used for a local definition of a remote queue, *RemoteQName* must not be blank when the open occurs.

If this definition is used for a queue-manager alias definition, *RemoteQName* must be blank when the open occurs.

If this definition is used for a reply-to alias, this name is the name of the queue that is to be the reply-to queue.

The maximum length of the string is MQ_Q_NAME_LENGTH.

RemoteQMgrName (MQCFST)
Name of remote queue manager (parameter identifier: MQCA_REMOTE_Q_MGR_NAME).

If an application opens the local definition of a remote queue, *RemoteQMgrName* must not be blank or the name of the connected queue manager. If *XmitQName* is blank there must be a local queue of this name, which is to be used as the transmission queue.

If this definition is used for a queue-manager alias, *RemoteQMgrName* is the name of the queue manager, which can be the name of the connected queue manager. Otherwise, if *XmitQName* is blank, when the queue is opened there must be a local queue of this name, which is to be used as the transmission queue.

If this definition is used for a reply-to alias, this name is the name of the queue manager that is to be the reply-to queue manager.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

XmitQName (MQCFST)
Transmission queue name (parameter identifier: MQCA_XMIT_Q_NAME).

Specifies the local name of the transmission queue to be used for messages destined for the remote queue, for either a remote queue or for a queue-manager alias definition.

If *XmitQName* is blank, a queue with the same name as *RemoteQMgrName* is used as the transmission queue.

This attribute is ignored if the definition is being used as a queue-manager alias and *RemoteQMgrName* is the name of the connected queue manager.

It is also ignored if the definition is used as a reply-to queue alias definition.

The maximum length of the string is MQ_Q_NAME_LENGTH.

DefinitionType (MQCFIN)
Queue definition type (parameter identifier: MQIA_DEFINITION_TYPE).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

The value may be:

MQQDT_PERMANENT_DYNAMIC
Dynamically defined permanent queue.

MQQDT_TEMPORARY_DYNAMIC
Dynamically defined temporary queue.

Scope (MQCFIN)
Scope of the queue definition (parameter identifier: MQIA_SCOPE).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

Specifies whether the scope of the queue definition does not extend beyond the queue manager which owns the queue, or whether the queue name is contained in a cell directory, so that it is known to all of the queue managers within the cell.

Model and dynamic queues cannot have cell scope.

The command fails if the new queue has a *Scope* attribute of MQSCO_CELL, but no name service supporting a cell directory has been configured.

The value may be:

MQSCO_Q_MGR
Queue-manager scope.

MQSCO_CELL
Cell scope.

This value is not supported on OS/400 and 32-bit Windows.

QDepthHighLimit (MQCFIN)
High limit for queue depth (parameter identifier: MQIA_Q_DEPTH_HIGH_LIMIT).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

The threshold against which the queue depth is compared to generate a Queue Depth High event.

This event indicates that an application has put a message to a queue, and this has caused the number of messages on the queue to become greater than or equal to the queue depth high threshold. See the *QDepthHighEvent* parameter.

The value is expressed as a percentage of the maximum queue depth (*MaxQDepth* attribute), and must be greater than or equal to zero and less than or equal to 100.

QDepthLowLimit (MQCFIN)
Low limit for queue depth (parameter identifier: MQIA_Q_DEPTH_LOW_LIMIT).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

The threshold against which the queue depth is compared to generate a Queue Depth Low event.

This event indicates that an application has retrieved a message from a queue, and this has caused the number of messages on the queue to become less than or equal to the queue depth low threshold. See the *QDepthLowEvent* parameter.

The value is expressed as a percentage of the maximum queue depth (*MaxQDepth* attribute), and must be greater than or equal to zero and less than or equal to 100.

QDepthMaxEvent (MQCFIN)
Controls whether Queue Full events are generated (parameter identifier: MQIA_Q_DEPTH_MAX_EVENT).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

A Queue Full event indicates that an **MQPUT** call to a queue has been rejected because the queue is full, that is, the queue depth has already reached its maximum value.

Note: The value of this attribute can change implicitly. See Chapter 3, “Understanding performance events” on page 17.

The value may be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

QDepthHighEvent (MQCFIN)
Controls whether Queue Depth High events are generated (parameter identifier: MQIA_Q_DEPTH_HIGH_EVENT).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

A Queue Depth High event indicates that an application has put a message on a queue, and this has caused the number of messages on the queue to become greater than or equal to the queue depth high threshold (see the *QDepthHighLimit* parameter).

Note: The value of this attribute can change implicitly. See Chapter 3, “Understanding performance events” on page 17.

The value may be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

QDepthLowEvent (MQCFIN)
Controls whether Queue Depth Low events are generated (parameter identifier: MQIA_Q_DEPTH_LOW_EVENT).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

A Queue Depth Low event indicates that an application has retrieved a message from a queue, and this has caused the number of messages on the queue to become less than or equal to the queue depth low threshold (see the *QDepthLowLimit* parameter).

Note: The value of this attribute can change implicitly. See Chapter 3, "Understanding performance events" on page 17.

The value may be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

QServiceInterval (MQCFIN)
Target for queue service interval (parameter identifier: MQIA_Q_SERVICE_INTERVAL).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

The service interval used for comparison to generate Queue Service Interval High and Queue Service Interval OK events. See the *QServiceIntervalEvent* parameter.

The value is in units of milliseconds, and must be greater than or equal to zero, and less than or equal to 999 999 999.

QServiceIntervalEvent (MQCFIN)
Controls whether Queue Service Interval High or Queue Service Interval OK events are generated (parameter identifier: MQIA_Q_SERVICE_INTERVAL_EVENT).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

A Service Interval High event is generated when a check indicates that no messages have been retrieved from the queue for at least the time indicated by the *QServiceInterval* attribute.

A Queue Service Interval OK event is generated when a check indicates that messages have been retrieved from the queue within the time indicated by the *QServiceInterval* attribute.

Note: The value of this attribute can change implicitly. See Chapter 3, "Understanding performance events" on page 17.

The value may be:

MQQSIE_HIGH
Queue Service Interval High events enabled.

- Queue Service Interval High events are **enabled** and

- Queue Service Interval OK events are **disabled**.

MQQSIE_OK
Queue Service Interval OK events enabled.

- Queue Service Interval High events are **disabled** and
- Queue Service Interval OK events are **enabled**.

MQQSIE_NONE
No queue service interval events enabled.

- Queue Service Interval High events are **disabled** and
- Queue Service Interval OK events are also **disabled**.

Error codes

In addition to the values for any command shown on page 136, for this command the following may be returned in the response format header:

Reason (MQLONG)
The value may be:

MQRC_UNKNOWN_OBJECT_NAME
(2085, X'825') Unknown object name.

MQRCCF_ATTR_VALUE_ERROR
Attribute value not valid.

MQRCCF_CELL_DIR_NOT_AVAILABLE
Cell directory is not available.

MQRCCF_CFIN_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFIN_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIN_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_DYNAMIC_Q_SCOPE_ERROR
Dynamic queue scope error.

MQRCCF_LIKE_OBJECT_WRONG_TYPE
New and existing objects have different type.

MQRCCF_OBJECT_ALREADY_EXISTS
Object already exists.

MQRCCF_OBJECT_NAME_ERROR
Object name not valid.

MQRCCF_OBJECT_OPEN
: Object is open.

MQRCCF_OBJECT_WRONG_TYPE
: Object has wrong type.

MQRCCF_PARM_COUNT_TOO_BIG
: Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
: Parameter count too small.

MQRCCF_PARM_SEQUENCE_ERROR
: Parameter sequence not valid.

MQRCCF_Q_ALREADY_IN_CELL
: Queue already exists in cell.

MQRCCF_Q_TYPE_ERROR
: Queue type not valid.

MQRCCF_REPLACE_VALUE_ERROR
: Replace value not valid.

MQRCCF_STRUCTURE_TYPE_ERROR
: Structure type not valid.

Create Channel

The Create Channel (MQCMD_CREATE_CHANNEL) command creates an MQSeries channel definition. Any attributes that are not defined explicitly are set to the default values on the destination queue manager. If a system default channel exists for the type of channel being created, the default values are taken from there.

Required parameters:

ChannelName, *ChannelType*

Optional parameters (any ChannelType):

Replace, *TransportType*, *ChannelDesc*, *SecurityExit*, *MsgExit*, *SendExit*, *ReceiveExit*, *MaxMsgLength*, *SecurityUserData*, *MsgUserData*, *SendUserData*, *ReceiveUserData*

Optional parameters (sender or server ChannelType):

ModeName, *TpName*, *ConnectionName*, *XmitQName*, *MCAName*, *BatchSize*, *DiscInterval*, *ShortRetryCount*, *ShortRetryInterval*, *LongRetryCount*, *LongRetryInterval*, *SeqNumberWrap*, *DataConversion*, *MCAType*, *MCAUserIdentifier*, *UserIdentifier*, *Password*, *HeartbeatInterval*, *NonPersistentMsgSpeed BatchInterval*

Optional parameters (receiver ChannelType):

BatchSize, *PutAuthority*, *SeqNumberWrap*, *MCAUserIdentifier*, *MsgRetryExit*, *MsgRetryUserData*, *MsgRetryCount*, *MsgRetryInterval*, *HeartbeatInterval*, *NonPersistentMsgSpeed*

Optional parameters (requester ChannelType):

ModeName, *TpName*, *ConnectionName*, *MCAName*, *BatchSize*, *PutAuthority*, *SeqNumberWrap*, *MCAType*, *MCAUserIdentifier*, *UserIdentifier*, *Password*, *MsgRetryExit*, *MsgRetryUserData*, *MsgRetryCount*, *MsgRetryInterval*, *HeartbeatInterval*, *NonPersistentMsgSpeed*

Optional parameters (server-connection ChannelType):

MCAUserIdentifier,

Optional parameters (client-connection ChannelType):

ModeName, *TpName*, *QMgrName*, *ConnectionName*, *UserIdentifier*, *Password*

Required parameters

ChannelName (MQCFST)

Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The name of the new channel definition. The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Channel names must be unique; if a channel definition with this name already exists, the value of *Replace* must be MQRP_YES. The channel type of the existing channel definition must be the same as the channel type of the new channel definition otherwise it cannot be replaced.

ChannelType (MQCFIN)

Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

Specifies the type of the channel being defined. The value may be:

MQCHT_SENDER
: Sender.

MQCHT_SERVER
Server.

MQCHT_RECEIVER
Receiver.

MQCHT_REQUESTER
Requester.

MQCHT_SVRCONN
Server-connection (for use by clients).

This value is not supported in the following environment: 32-bit Windows.

MQCHT_CLNTCONN
Client connection.

This value is not supported in the following environments: OS/400, 32-bit Windows.

Optional parameters

Replace (MQCFIN)
Replace channel definition (parameter identifier: MQIACF_REPLACE).

The value may be:

MQRP_YES
Replace existing definition.

MQRP_NO
Do not replace existing definition.

TransportType (MQCFIN)
Transmission protocol type (parameter identifier: MQIACH_XMIT_PROTOCOL_TYPE).

No check is made that the correct transport type has been specified if the channel is initiated from the other end. The value may be:

MQXPT_LU62
LU 6.2.

This value is not supported on 32-bit Windows.

MQXPT_TCP
TCP/IP.

This is the *only* value supported on 32-bit Windows.

MQXPT_NETBIOS
NetBIOS.

This value is supported in the following environments: OS/2, 32-bit Windows, Windows NT.

MQXPT_SPX
SPX.

This value is supported in the following environments: OS/2, Windows NT, Windows client, DOS client.

MQXPT_DECNET
DECnet.

This value is supported in the following environment: OpenVMS.

ModeName (MQCFST)
Mode name (parameter identifier: MQCACH_MODE_NAME).

This is the LU 6.2 mode name. The maximum length of the string is MQ_MODE_NAME_LENGTH.

- On OpenVMS, OS/400, Tandem NSK, UNIX systems, and Windows NT, this can be set only to blanks. The actual name is taken instead from the CPI-C Communications Side Object or (on Windows NT) from the CPI-C symbolic destination name properties.
- On 32-bit Windows, this parameter is accepted but ignored.

This parameter is valid only for channels with a *TransportType* of MQXPT_LU62. It is not valid for receiver channels.

TpName (MQCFST)
Transaction program name (parameter identifier: MQCACH_TP_NAME).

LU 6.2 transaction program name. The maximum length of the string is MQ_TP_NAME_LENGTH.

On OpenVMS, OS/400, Tandem NSK, UNIX systems, and Windows NT, this can be set only to blanks. The actual name is taken instead from the CPI-C Communications Side Object or (on Windows NT) from the CPI-C symbolic destination name properties.

- On 32-bit Windows, this parameter is accepted but ignored.

This parameter is valid only for channels with a *TransportType* of MQXPT_LU62. It is not valid for receiver channels.

QMgrName (MQCFST)
Queue-manager name (parameter identifier: MQCA_Q_MGR_NAME).

For channels with a *ChannelType* of MQCHT_CLNTCONN, this is the name of a queue manager to which a client application can request connection.

On 32-bit Windows, this parameter is accepted but ignored.

For channels of other types, this parameter is not valid. The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

ChannelDesc (MQCFST)
Channel description (parameter identifier: MQCACH_DESC).

The maximum length of the string is MQ_CHANNEL_DESC_LENGTH.

Use characters from the character set, identified by the coded character set identifier (CCSID) for the message queue manager on which the command is executing, to ensure that the text is translated correctly.

BatchSize (MQCFIN)
Batch size (parameter identifier: MQIACH_BATCH_SIZE).

The maximum number of messages that can be sent down a channel before a checkpoint is taken.

The batch size which is actually used is the lowest of the following:

- The *BatchSize* of the sending channel
- The *BatchSize* of the receiving channel
- The maximum number of uncommitted messages at the sending queue manager
- The maximum number of uncommitted messages at the receiving queue manager

The maximum number of uncommitted messages is specified by the *MaxUncommittedMsgs* parameter of the Change Queue Manager command.

Specify a value in the range 1-9999.

This parameter is not valid for channels with a *ChannelType* of MQCHT_SVRCONN or MQCHT_CLNTCONN.

SecurityExit (MQCFST)
Security exit name (parameter identifier: MQCACH_SEC_EXIT_NAME).

If a nonblank name is defined, the security exit is invoked at the following times:

- Immediately after establishing a channel.

  Before any messages are transferred, the exit is given the opportunity to instigate security flows to validate connection authorization.

- Upon receipt of a response to a security message flow.

  Any security message flows received from the remote processor on the remote machine are passed to the exit.

The exit is given the entire application message and message descriptor for modification.

The format of the string depends on the platform, as follows:

- On UNIX systems, it is of the form

  `libraryname(functionname)`

- On OS/2 and Windows, it is of the form

  `dllname(functionname)`

  where *dllname* is specified without the suffix ".DLL".

- On OS/400, it is of the form

  `progname libname`

  where *progname* occupies the first 10 characters, and *libname* the second 10 characters (both blank-padded to the right if necessary).

- On OpenVMS, it is of the form

  `imagename(functionname)`

The maximum length of the string is MQ_EXIT_NAME_LENGTH. Set unused character positions to blanks.

MsgExit (MQCFST)
Message exit name (parameter identifier: MQCACH_MSG_EXIT_NAME).

If a nonblank name is defined, the exit is invoked immediately after a message has been retrieved from the transmission queue. The exit is

given the entire application message and message descriptor for modification.

For channels with a channel type (*ChannelType*) of MQCHT_SVRCONN or MQCHT_CLNTCONN, this parameter is not relevant, since message exits are not invoked for such channels.

The format of the string is the same as for *SecurityExit*.

The maximum length of the string is MQ_EXIT_NAME_LENGTH. Set unused character positions to blanks.

In the following environments, a list of exit names can be specified by using an MQCFSL structure instead of an MQCFST structure: AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

- The exits are invoked in the order specified in the list.
- A list with only one name is equivalent to specifying a single name in an MQCFST structure.
- You cannot specify both a list (MQCFSL) and a single entry (MQCFST) structure for the same channel attribute.
- The total length of all of the exit names in the list (excluding trailing blanks in each name) must not exceed MQ_TOTAL_EXIT_NAME_LENGTH. An individual string must not exceed MQ_EXIT_NAME_LENGTH.

SendExit (MQCFST)
Send exit name (parameter identifier: MQCACH_SEND_EXIT_NAME).

If a nonblank name is defined, the exit is invoked immediately before data is sent out on the network. The exit is given the complete transmission buffer before it is transmitted; the contents of the buffer can be modified as required.

The format of the string is the same as for *SecurityExit*.

The maximum length of the string is MQ_EXIT_NAME_LENGTH. Set unused character positions to blanks.

In the following environments, a list of exit names can be specified by using an MQCFSL structure instead of an MQCFST structure: AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

- The exits are invoked in the order specified in the list.
- A list with only one name is equivalent to specifying a single name in an MQCFST structure.
- You cannot specify both a list (MQCFSL) and a single entry (MQCFST) structure for the same channel attribute.
- The total length of all of the exit names in the list (excluding trailing blanks in each name) must not exceed MQ_TOTAL_EXIT_NAME_LENGTH. An individual string must not exceed MQ_EXIT_NAME_LENGTH.

ReceiveExit (MQCFST)
Receive exit name (parameter identifier: MQCACH_RCV_EXIT_NAME).

If a nonblank name is defined, the exit is invoked before data received from the network is processed. The complete transmission buffer is passed to the exit and the contents of the buffer can be modified as required.

The format of the string is the same as for *SecurityExit*.

The maximum length of the string is MQ_EXIT_NAME_LENGTH. Set unused character positions to blanks.

In the following environments, a list of exit names can be specified by using an MQCFSL structure instead of an MQCFST structure: AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

- The exits are invoked in the order specified in the list.
- A list with only one name is equivalent to specifying a single name in an MQCFST structure.
- You cannot specify both a list (MQCFSL) and a single entry (MQCFST) structure for the same channel attribute.
- The total length of all of the exit names in the list (excluding trailing blanks in each name) must not exceed MQ_TOTAL_EXIT_NAME_LENGTH. An individual string must not exceed MQ_EXIT_NAME_LENGTH.

SeqNumberWrap (MQCFIN)
Sequence wrap number (parameter identifier: MQIACH_SEQUENCE_NUMBER_WRAP).

Specifies the maximum message sequence number. When the maximum is reached, sequence numbers wrap to start again at 1.

The maximum message sequence number is not negotiable; the local and remote channels must wrap at the same number.

Specify a value in the range 100 through 999 999 999.

This parameter is not valid for channels with a *ChannelType* of MQCHT_SVRCONN or MQCHT_CLNTCONN.

MaxMsgLength (MQCFIN)
Maximum message length (parameter identifier: MQIACH_MAX_MSG_LENGTH).

Specifies the maximum message length that can be transmitted on the channel. This is compared with the value for the remote channel and the actual maximum is the lowest of the two values.

The value zero means the maximum message length for the queue manager.

The lower limit for this parameter is 0. The upper limit depends on the environment:

- On AIX, HP-UX, OS/2, Sun Solaris, and Windows NT, the maximum message length is 100 MB (104 857 600 bytes).

- On OpenVMS, OS/400, Tandem NSK, UNIX systems not listed above, and 32-bit Windows, the maximum message length is 4 MB (4 194 304 bytes).

SecurityUserData (MQCFST)
Security exit user data (parameter identifier: MQCACH_SEC_EXIT_USER_DATA).

Specifies user data that is passed to the security exit. The maximum length of the string is MQ_EXIT_DATA_LENGTH.

MsgUserData (MQCFST)
Message exit user data (parameter identifier: MQCACH_MSG_EXIT_USER_DATA).

Specifies user data that is passed to the message exit. The maximum length of the string is MQ_EXIT_DATA_LENGTH.

In the following environments, a list of exit user data strings can be specified by using an MQCFSL structure instead of an MQCFST structure: AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

- Each exit user data string is passed to the exit at the same ordinal position in the *MsgExit* list.
- A list with only one name is equivalent to specifying a single name in an MQCFST structure.
- You cannot specify both a list (MQCFSL) and a single entry (MQCFST) structure for the same channel attribute.
- The total length of all of the exit user data in the list (excluding trailing blanks in each string) must not exceed MQ_TOTAL_EXIT_DATA_LENGTH. An individual string must not exceed MQ_EXIT_DATA_LENGTH.

SendUserData (MQCFST)
Send exit user data (parameter identifier: MQCACH_SEND_EXIT_USER_DATA).

Specifies user data that is passed to the send exit. The maximum length of the string is MQ_EXIT_DATA_LENGTH.

In the following environments, a list of exit user data strings can be specified by using an MQCFSL structure instead of an MQCFST structure: AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

- Each exit user data string is passed to the exit at the same ordinal position in the *SendExit* list.
- A list with only one name is equivalent to specifying a single name in an MQCFST structure.
- You cannot specify both a list (MQCFSL) and a single entry (MQCFST) structure for the same channel attribute.
- The total length of all of the exit user data in the list (excluding trailing blanks in each string) must not exceed MQ_TOTAL_EXIT_DATA_LENGTH. An individual string must not exceed MQ_EXIT_DATA_LENGTH.

ReceiveUserData (MQCFST)
Receive exit user data (parameter identifier: MQCACH_RCV_EXIT_USER_DATA).

Specifies user data that is passed to the receive exit. The maximum length of the string is MQ_EXIT_DATA_LENGTH.

In the following environments, a list of exit user data strings can be specified by using an MQCFSL structure instead of an MQCFST structure: AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

- Each exit user data string is passed to the exit at the same ordinal position in the *ReceiveExit* list.
- A list with only one name is equivalent to specifying a single name in an MQCFST structure.
- You cannot specify both a list (MQCFSL) and a single entry (MQCFST) structure for the same channel attribute.
- The total length of all of the exit user data in the list (excluding trailing blanks in each string) must not exceed MQ_TOTAL_EXIT_DATA_LENGTH. An individual string must not exceed MQ_EXIT_DATA_LENGTH.

ConnectionName (MQCFST)
Connection name (parameter identifier: MQCACH_CONNECTION_NAME).

The maximum length of the string is MQ_CONN_NAME_LENGTH.

Specify the name of the machine as required for the stated *TransportType*:

- For MQXPT_LU62 on OS/2, specify the fully-qualified name of the partner LU. On OS/400, and UNIX systems, specify the name of the CPI-C communications side object. On Windows NT specify the CPI-C symbolic destination name.
- For MQXPT_TCP specify either the host name or the network address of the remote machine.
- For MQXPT_NETBIOS specify the NetBIOS station name.
- For MQXPT_SPX specify the 4 byte network address, the 6 byte node address, and the 2 byte socket number. These should be entered in hexadecimal, with a period separating the network and node addresses. The socket number should be enclosed in brackets, for example:

  ```
  CONNAME('0a0b0c0d.804abcde23a1(5e86)')
  ```

  If the socket number is omitted, the MQSeries default value (5e86 hex) is assumed.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_REQUESTER or MQCHT_CLNTCONN.

XmitQName (MQCFST)
Transmission queue name (parameter identifier: MQCACH_XMIT_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

A transmission queue name is required (either previously defined or specified here) if *ChannelType* is MQCHT_SENDER or MQCHT_SERVER. It is not valid for other channel types.

MCAName (MQCFST)
Message channel agent name (parameter identifier: MQCACH_MCA_NAME).

This is reserved, and if specified can be set only to blanks.

The maximum length of the string is MQ_MCA_NAME_LENGTH.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, or MQCHT_REQUESTER.

DiscInterval (MQCFIN)
Disconnection interval (parameter identifier: MQIACH_DISC_INTERVAL).

This defines the maximum number of seconds that the channel waits for messages to be put on a transmission queue before terminating the channel.

Specify a value in the range 0 through 999 999.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER or MQCHT_SERVER.

ShortRetryCount (MQCFIN)
Short retry count (parameter identifier: MQIACH_SHORT_RETRY).

The maximum number of attempts that are made by a sender or server channel to establish a connection to the remote machine, at intervals specified by *ShortRetryInterval* before the (normally longer) *LongRetryCount* and *LongRetryInterval* are used.

Retry attempts are made if the channel fails to connect initially (whether it is started automatically by the channel initiator or by an explicit command), and also if the connection fails after the channel has successfully connected. However, if the cause of the failure is such that retry is unlikely to be successful, retries are not attempted.

Specify a value in the range 0 through 999 999 999.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER or MQCHT_SERVER.

ShortRetryInterval (MQCFIN)
Short timer (parameter identifier: MQIACH_SHORT_TIMER).

Specifies the short retry wait interval for a sender or server channel that is started automatically by the channel initiator. It defines the interval in seconds between attempts to establish a connection to the remote machine.

The time is approximate; zero means that another connection attempt is made as soon as possible.

Specify a value in the range 0 through 999 999. Values exceeding this are treated as 999 999.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER or MQCHT_SERVER.

LongRetryCount (MQCFIN)
Long retry count (parameter identifier: MQIACH_LONG_RETRY).

When a sender or server channel is attempting to connect to the remote machine, and the count specified by *ShortRetryCount* has been exhausted, this specifies the maximum number of further attempts that are made to connect to the remote machine, at intervals specified by *LongRetryInterval*.

If this count is also exhausted without success, an error is logged to the operator, and the channel is stopped. The channel must subsequently be restarted with a command (it is not started automatically by the channel initiator), and it then makes only one attempt to connect, as it is assumed that the problem has now been cleared by the administrator. The retry sequence is not carried out again until after the channel has successfully connected.

Specify a value in the range 0 through 999 999 999.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER or MQCHT_SERVER.

LongRetryInterval (MQCFIN)
Long timer (parameter identifier: MQIACH_LONG_TIMER).

Specifies the long retry wait interval for a sender or server channel that is started automatically by the channel initiator. It defines the interval in seconds between attempts to establish a connection to the remote machine, after the count specified by *ShortRetryCount* has been exhausted.

The time is approximate; zero means that another connection attempt is made as soon as possible.

Specify a value in the range 0 through 999 999. Values exceeding this are treated as 999 999.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER or MQCHT_SERVER.

DataConversion (MQCFIN)
Whether sender should convert application data (parameter identifier: MQIACH_DATA_CONVERSION).

This parameter is valid only for *ChannelType* values of MQCHT_SENDER or MQCHT_SERVER.

The value may be:

MQCDC_NO_SENDER_CONVERSION
No conversion by sender.

MQCDC_SENDER_CONVERSION
Conversion by sender.

This value is not supported on 32-bit Windows.

PutAuthority (MQCFIN)
Put authority (parameter identifier: MQIACH_PUT_AUTHORITY).

Specifies whether the user identifier in the context information associated

with a message should be used to establish authority to put the message on the destination queue.

This parameter is valid only for *ChannelType* values of MQCHT_RECEIVER or MQCHT_REQUESTER. The value may be:

MQPA_DEFAULT
Default user identifier is used.

MQPA_CONTEXT
Context user identifier is used.

MCAType (MQCFIN)
Message channel agent type (parameter identifier: MQIACH_MCA_TYPE).

Specifies the type of the message channel agent program.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, or MQCHT_REQUESTER.

The value may be:

MQMCAT_PROCESS
Process.

MQMCAT_THREAD
Thread (OS/2 and Windows NT only).

MCAUserIdentifier (MQCFST)
Message channel agent user identifier (parameter identifier: MQCACH_MCA_USER_ID).

If this is nonblank, it is the user identifier which is to be used by the message channel agent for authorization to access MQ resources, including (if *PutAuthority* is MQPA_DEFAULT) authorization to put the message to the destination queue for receiver or requester channels.

If it is blank, the message channel agent uses its default user identifier.

This user identifier can be overridden by one supplied by a channel security exit.

This parameter is not valid for channels with a *ChannelType* of MQCHT_CLNTCONN.

The maximum length of the string is MQ_USER_ID_LENGTH.

UserIdentifier (MQCFST)
Task user identifier (parameter identifier: MQCACH_USER_ID).

This is used by the message channel agent when attempting to initiate a secure SNA session with a remote message channel agent.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_REQUESTER or MQCHT_CLNTCONN.

- This parameter is supported in the following environments: OpenVMS, OS/2, Tandem NSK, UNIX systems.
- On 32-bit Windows, the parameter is accepted but ignored.

The maximum length of the string is MQ_USER_ID_LENGTH. However, only the first 10 characters are used.

Password (MQCFST)
Password (parameter identifier: MQCACH_PASSWORD).

This is used by the message channel agent when attempting to initiate a secure SNA session with a remote message channel agent.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_REQUESTER or MQCHT_CLNTCONN.

- This parameter is supported in the following environments: OpenVMS, OS/2, Tandem NSK, UNIX systems.
- On 32-bit Windows, the parameter is accepted but ignored.

The maximum length of the string is MQ_PASSWORD_LENGTH. However, only the first 10 characters are used.

MsgRetryExit (MQCFST)
Message retry exit name (parameter identifier: MQCACH_MR_EXIT_NAME).

- This parameter is supported in the following environments: AIX, AT&T GIS UNIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.
- On 32-bit Windows, the parameter is accepted but must be blank.

If a nonblank name is defined, the exit is invoked prior to performing a wait before retrying a failing message.

The format of the string is the same as for *SecurityExit*.

The maximum length of the string is MQ_EXIT_NAME_LENGTH. Set unused character positions to blanks.

This parameter is valid only for *ChannelType* values of MQCHT_RECEIVER or MQCHT_REQUESTER.

MsgRetryUserData (MQCFST)
Message retry exit user data (parameter identifier: MQCACH_MR_EXIT_USER_DATA).

- This parameter is supported in the following environments: AIX, AT&T GIS UNIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.
- On 32-bit Windows, the parameter is accepted but ignored.

Specifies user data that is passed to the message retry exit.

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

This parameter is valid only for *ChannelType* values of MQCHT_RECEIVER or MQCHT_REQUESTER.

MsgRetryCount (MQCFIN)
Message retry count (parameter identifier: MQIACH_MR_COUNT).

- This parameter is supported in the following environments: AIX, AT&T GIS UNIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.
- On 32-bit Windows, the parameter is accepted but must be zero.

Specifies the number of times that a failing message should be retried.

Specify a value in the range 0 through 999 999 999.

This parameter is valid only for *ChannelType* values of MQCHT_RECEIVER or MQCHT_REQUESTER.

MsgRetryInterval (MQCFIN)
Message retry interval (parameter identifier: MQIACH_MR_INTERVAL).

- This parameter is supported in the following environments: AIX, AT&T GIS UNIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.
- On 32-bit Windows, the parameter is accepted but must be zero.

Specifies the minimum time interval in milliseconds between retries of failing messages.

Specify a value in the range 0 through 999 999 999.

This parameter is valid only for *ChannelType* values of MQCHT_RECEIVER or MQCHT_REQUESTER.

HeartbeatInterval (MQCFIN)
Heartbeat interval (parameter identifier: MQIACH_HB_INTERVAL).

The interpretation of this parameter depends on the channel type, as follows:

- For a channel type of MQCHT_SENDER, MQCHT_SERVER, MQCHT_RECEIVER or MQCHT_REQUESTER, this is the time in seconds between heartbeat flows passed from the sending MCA when there are no messages on the transmission queue. This gives the receiving MCA the opportunity to quiesce the channel. To be useful, *HeartbeatInterval* should be significantly less than *DiscInterval*. However, the only check is that the value is within the permitted range.

  This type of heartbeat is supported in the following environments: AIX, HP-UX, MVS/ESA, OS/2, OS/400, Sun Solaris, Windows NT.

- For a channel type of MQCHT_CLNTCONN or MQCHT_SVRCONN, this is the time in seconds between heartbeat flows passed from the server MCA when that MCA has issued an MQGET call with the MQGMO_WAIT option on behalf of a client application. This allows the server MCA to handle situations where the client connection fails during an MQGET with MQGMO_WAIT.

  This type of heartbeat is supported in the following environments: AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

The value must be in the range 0 through 999 999. A value of 0 means that no heartbeat exchange occurs. The value that is actually used is the larger of the values specified at the sending side and receiving side.

NonPersistentMsgSpeed (MQCFIN)
Speed at which non-persistent messages are to be sent (parameter identifier: MQIACH_NPM_SPEED).

This parameter is supported in the following environments: AIX, HP-UX, OS/2, OS/400, Sun Solaris, 32-bit Windows, Windows NT.

Specifying MQNPMS_FAST means that non-persistent messages on a channel need not wait for a syncpoint before being made available for retrieval. The advantage of this is that non-persistent messages become available for retrieval far more quickly. The disadvantage is that because

they do not wait for a syncpoint, they may be lost if there is a transmission failure.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_RECEIVER or MQCHT_REQUESTER. The value may be:

MQNPMS_NORMAL
Normal speed.

MQNPMS_FAST
Fast speed.

BatchInterval (MQCFIN)
Batch interval (parameter identifier: MQIACH_BATCH_INTERVAL).

This is the approximate time in milliseconds that a channel will keep a batch open, if fewer than *BatchSize* messages have been transmitted in the current batch.

This parameter is supported in the following environments: AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

If *BatchInterval* is greater than zero, the batch is terminated by whichever of the following occurs first:

- *BatchSize* messages have been sent, or
- *BatchInterval* milliseconds have elapsed since the start of the batch.

If *BatchInterval* is zero, the batch is terminated by whichever of the following occurs first:

- *BatchSize* messages have been sent, or
- the transmission queue becomes empty.

BatchInterval must be in the range 0 through 999 999 999.

This parameter applies only to channels with a *ChannelType* of:

MQCHT_SENDER
MQCHT_SERVER

Error codes

In addition to the values for any command shown on page 136, for this command the following may be returned in the response format header:

Reason (MQLONG)
The value may be:

MQRCCF_BATCH_INT_ERROR
Batch interval not valid.

MQRCCF_BATCH_INT_WRONG_TYPE
Batch interval parameter not allowed for this channel type.

MQRCCF_BATCH_SIZE_ERROR
Batch size not valid.

MQRCCF_CFIN_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFIN_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIN_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFSL_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFSL_TOTAL_LENGTH_ERROR
Total string length error.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_CHANNEL_ALREADY_EXISTS
Channel already exists.

MQRCCF_CHANNEL_NAME_ERROR
Channel name error.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_CHANNEL_TYPE_ERROR
Channel type not valid.

MQRCCF_CONN_NAME_ERROR
Error in connection name parameter.

MQRCCF_DISC_INT_ERROR
Disconnection interval not valid.

MQRCCF_DISC_INT_WRONG_TYPE
Disconnection interval not allowed for this channel type.

MQRCCF_HB_INTERVAL_ERROR
Heartbeat interval not valid.

MQRCCF_HB_INTERVAL_WRONG_TYPE
Heartbeat interval parameter not allowed for this channel type.

MQRCCF_LONG_RETRY_ERROR
Long retry count not valid.

MQRCCF_LONG_RETRY_WRONG_TYPE
Long retry parameter not allowed for this channel type.

MQRCCF_LONG_TIMER_ERROR
Long timer not valid.

MQRCCF_LONG_TIMER_WRONG_TYPE
Long timer parameter not allowed for this channel type.

MQRCCF_MAX_MSG_LENGTH_ERROR
Maximum message length not valid.

MQRCCF_MCA_NAME_ERROR
Message channel agent name error.

MQRCCF_MCA_NAME_WRONG_TYPE
Message channel agent name not allowed for this channel type.

MQRCCF_MCA_TYPE_ERROR
Message channel agent type not valid.

MQRCCF_MISSING_CONN_NAME
Connection name parameter required but missing.

MQRCCF_MR_COUNT_ERROR
Message retry count not valid.

MQRCCF_MR_COUNT_WRONG_TYPE
Message-retry count parameter not allowed for this channel type.

MQRCCF_MR_EXIT_NAME_ERROR
Channel message-retry exit name error.

MQRCCF_MR_EXIT_NAME_WRONG_TYPE
Message-retry exit parameter not allowed for this channel type.

MQRCCF_MR_INTERVAL_ERROR
Message retry interval not valid.

MQRCCF_MR_INTERVAL_WRONG_TYPE
Message-retry interval parameter not allowed for this channel type.

MQRCCF_MSG_EXIT_NAME_ERROR
Channel message exit name error.

MQRCCF_NPM_SPEED_ERROR
Nonpersistent message speed not valid.

MQRCCF_NPM_SPEED_WRONG_TYPE
Nonpersistent message speed parameter not allowed for this channel type.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_PARM_SEQUENCE_ERROR
Parameter sequence not valid.

MQRCCF_PUT_AUTH_ERROR
Put authority value not valid.

MQRCCF_PUT_AUTH_WRONG_TYPE
Put authority parameter not allowed for this channel type.

MQRCCF_RCV_EXIT_NAME_ERROR
Channel receive exit name error.

MQRCCF_REPLACE_VALUE_ERROR
Replace value not valid.

MQRCCF_SEC_EXIT_NAME_ERROR
Channel security exit name error.

MQRCCF_SEND_EXIT_NAME_ERROR
Channel send exit name error.

MQRCCF_SEQ_NUMBER_WRAP_ERROR
Sequence wrap number not valid.

MQRCCF_SHORT_RETRY_ERROR
Short retry count not valid.

MQRCCF_SHORT_RETRY_WRONG_TYPE
Short retry parameter not allowed for this channel type.

MQRCCF_SHORT_TIMER_ERROR
Short timer value not valid.

MQRCCF_SHORT_TIMER_WRONG_TYPE
Short timer parameter not allowed for this channel type.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

MQRCCF_XMIT_PROTOCOL_TYPE_ERR
Transmission protocol type not valid.

MQRCCF_XMIT_Q_NAME_ERROR
Transmission queue name error.

MQRCCF_XMIT_Q_NAME_WRONG_TYPE
Transmission queue name not allowed for this channel type.

Create Process

The Create Process (MQCMD_CREATE_PROCESS) command creates a new MQSeries process definition. Any attributes that are not defined explicitly are set to the default values on the destination queue manager.

| *This PCF is not supported if you are using MQSeries for Windows Version 2.1.*

Required parameters:
ProcessName

Optional parameters:
Replace, *ProcessDesc*, *ApplType*, *ApplId*, *EnvData*, *UserData*

Required parameters

ProcessName (MQCFST)
The new process definition to be created (parameter identifier: MQCA_PROCESS_NAME).

If a process definition with this name already exists, *Replace* must be specified as MQRP_YES.

The maximum length of the string is MQ_PROCESS_NAME_LENGTH.

Optional parameters

Replace (MQCFIN)
Replace process definition (parameter identifier: MQIACF_REPLACE).

If a process definition with the same name as *ProcessName* already exists, this specifies whether it is to be replaced.

The value may be:

MQRP_YES
Replace existing definition.

MQRP_NO
Do not replace existing definition.

ProcessDesc (MQCFST)
Description of process definition (parameter identifier: MQCA_PROCESS_DESC).

A plain-text comment that provides descriptive information about the process definition. It should contain only displayable characters.

If characters that are not in the coded character set identifier (CCSID) for the queue manager on which the command is executing are used, they may be translated incorrectly.

The maximum length of the string is MQ_PROCESS_DESC_LENGTH.

ApplType (MQCFIN)
Application type (parameter identifier: MQIA_APPL_TYPE).

Valid application types are:

MQAT_OS400
OS/400 application.

MQAT_OS2
OS/2 or Presentation Manager application.

MQAT_WINDOWS_NT
Windows NT or 32-bit Windows application.

MQAT_DOS
DOS client application.

MQAT_WINDOWS
Windows client or 16-bit Windows application.

MQAT_UNIX
UNIX application.

MQAT_AIX
AIX application (same value as MQAT_UNIX).

MQAT_CICS
CICS transaction.

MQAT_VMS
OpenVMS application.

MQAT_NSK
Tandem NSK application.

MQAT_DEFAULT
Default application type.

user-value: User defined application type in the range 65 536 through 999 999 999 (not checked).

Only application types (other than user-defined types) that are supported on the platform at which the command is executed should be used:

- On OpenVMS:

  MQAT_VMS (default),
  MQAT_DOS,
  MQAT_WINDOWS, and
  MQAT_DEFAULT are supported.

- On OS/400:

  MQAT_OS400 (default),
  MQAT_CICS and
  MQAT_DEFAULT are supported.

- On OS/2:,

  MQAT_OS2 (default),
  MQAT_DOS,
  MQAT_WINDOWS,
  MQAT_AIX,
  MQAT_CICS and
  MQAT_DEFAULT are supported.

- On Tandem NSK:

MQAT_NSK (default),
MQAT_DOS,
MQAT_WINDOWS, and
MQAT_DEFAULT are supported.

- On Windows NT:

  MQAT_WINDOWS_NT (default),
  MQAT_OS2
  MQAT_DOS,
  MQAT_WINDOWS,
  MQAT_CICS and
  MQAT_DEFAULT are supported.

- On UNIX systems:

  MQAT_UNIX (default),
  MQAT_OS2,
  MQAT_DOS,
  MQAT_WINDOWS,
  MQAT_CICS and
  MQAT_DEFAULT are supported.

ApplId (MQCFST)
Application identifier (parameter identifier: MQCA_APPL_ID).

This is the name of the application to be started, on the platform for which the command is executing, and might typically be a program name and library name.

The maximum length of the string is MQ_PROCESS_APPL_ID_LENGTH.

EnvData (MQCFST)
Environment data (parameter identifier: MQCA_ENV_DATA).

A character string that contains environment information pertaining to the application to be started.

The maximum length of the string is MQ_PROCESS_ENV_DATA_LENGTH.

UserData (MQCFST)
User data (parameter identifier: MQCA_USER_DATA).

A character string that contains user information pertaining to the application (defined by *ApplId*) that is to be started.

The maximum length of the string is MQ_PROCESS_USER_DATA_LENGTH.

Error codes

In addition to the values for any command shown on page 136, for this command the following may be returned in the response format header:

Reason (MQLONG)
The value may be:

MQRC_UNKNOWN_OBJECT_NAME
(2085, X'825') Unknown object name.

MQRCCF_ATTR_VALUE_ERROR
Attribute value not valid.

MQRCCF_CFIN_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFIN_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIN_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_OBJECT_ALREADY_EXISTS
Object already exists.

MQRCCF_OBJECT_NAME_ERROR
Object name not valid.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_PARM_SEQUENCE_ERROR
Parameter sequence not valid.

MQRCCF_REPLACE_VALUE_ERROR
Replace value not valid.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

Create Queue

The Create Queue (MQCMD_CREATE_Q) command creates a queue definition with the specified attributes. All attributes that are not specified are set to the default value for the type of queue that is created.

Required parameters:

QName, *QType*

Optional parameters (any QType):

Replace, *QDesc*, *InhibitPut*, *DefPriority*, *DefPersistence*

Optional parameters (alias QType):

InhibitGet, *BaseQName*, *Scope*

Optional parameters (local QType):

InhibitGet, *ProcessName*, *MaxQDepth*, *MaxMsgLength*, *BackoutThreshold*, *BackoutRequeueName*, *Shareability*, *DefInputOpenOption*, *HardenGetBackout*, *MsgDeliverySequence*, *RetentionInterval*, *DistLists*, *Usage*, *InitiationQName*, *TriggerControl*, *TriggerType*, *TriggerMsgPriority*, *TriggerDepth*, *TriggerData*, *Scope*, *QDepthHighLimit*, *QDepthLowLimit*, *QDepthMaxEvent*, *QDepthHighEvent*, *QDepthLowEvent*, *QServiceInterval*, *QServiceIntervalEvent*

Optional parameters (remote QType):

RemoteQName, *RemoteQMgrName*, *XmitQName*, *Scope*

Optional parameters (model QType):

InhibitGet, *ProcessName*, *MaxQDepth*, *MaxMsgLength*, *BackoutThreshold*, *BackoutRequeueName*, *Shareability*, *DefInputOpenOption*, *HardenGetBackout*, *MsgDeliverySequence*, *RetentionInterval*, *DistLists*, *Usage*, *InitiationQName*, *TriggerControl*, *TriggerType*, *TriggerMsgPriority*, *TriggerDepth*, *TriggerData*, *DefinitionType*, *QDepthHighLimit*, *QDepthLowLimit*, *QDepthMaxEvent*, *QDepthHighEvent*, *QDepthLowEvent*, *QServiceInterval*, *QServiceIntervalEvent*

Required parameters

QName (MQCFST)

Queue name (parameter identifier: MQCA_Q_NAME).

The name of the queue to be created. The maximum length of the string is MQ_Q_NAME_LENGTH.

Queue name must be unique; if a queue definition already exists with the name and type of the new queue, *Replace* must be specified as MQRP_YES. If a queue definition exists with the same name as and a different type from the new queue, the command will fail.

QType (MQCFIN)

Queue type (parameter identifier: MQIA_Q_TYPE).

The value may be:

MQQT_ALIAS
: Alias queue definition.

MQQT_LOCAL
: Local queue.

MQQT_REMOTE
Local definition of a remote queue.

The following is supported on all platforms, but on OS/400 for receipt by MQSeries for AS/400 V4R2:

MQQT_MODEL
Model queue definition.

Optional parameters

Replace (MQCFIN)
Replace attributes (parameter identifier: MQIACF_REPLACE).

If the object already exists, the effect is similar to issuing the Change Queue command without the MQFC_YES option on the *Force* parameter, and with *all* of the other attributes specified. In particular, note that any messages which are on the existing queue are retained.

(The difference between the Change Queue command without MQFC_YES on the *Force* parameter, and the Create Queue command with MQRP_YES on the *Replace* parameter, is that the Change Queue command does not change unspecified attributes, but Create Queue with MQRP_YES sets *all* the attributes. When you use MQRP_YES, unspecified attributes are taken from the default definition, and the attributes of the object being replaced, if one exists, are ignored.)

The command fails if both of the following are true:

- The command sets attributes that would require the use of MQFC_YES on the *Force* parameter if you were using the Change Queue command
- The object is open

The Change Queue command with MQFC_YES on the *Force* parameter succeeds in this situation.

If MQSCO_CELL is specified on the *Scope* parameter on OS/2 or UNIX systems, and there is already a queue with the same name in the cell directory, the command fails, whether or not MQRP_YES is specified.

The value may be:

MQRP_YES
Replace existing definition.

MQRP_NO
Do not replace existing definition.

QDesc (MQCFST)
Queue description (parameter identifier: MQCA_Q_DESC).

Text that briefly describes the object. The maximum length of the string is MQ_Q_DESC_LENGTH.

Use characters from the character set identified by the coded character set identifier (CCSID) for the queue manager on which the command is executing to ensure that the text is translated correctly.

InhibitPut (MQCFIN)
Whether put operations are allowed (parameter identifier: MQIA_INHIBIT_PUT).

Specifies whether messages can be put on the queue.

The value may be:

MQQA_PUT_ALLOWED
Put operations are allowed.

MQQA_PUT_INHIBITED
Put operations are inhibited.

DefPriority (MQCFIN)
Default priority (parameter identifier: MQIA_DEF_PRIORITY).

Specifies the default priority of messages put on the queue. The value must be in the range zero through to the maximum priority value that is supported (9).

DefPersistence (MQCFIN)
Default persistence (parameter identifier: MQIA_DEF_PERSISTENCE).

Specifies the default for message-persistence on the queue. Message persistence determines whether or not messages are preserved across restarts of the queue manager.

The value may be:

MQPER_PERSISTENT
Message is persistent.

MQPER_NOT_PERSISTENT
Message is not persistent.

InhibitGet (MQCFIN)
Whether get operations are allowed (parameter identifier: MQIA_INHIBIT_GET).

The value may be:

MQQA_GET_ALLOWED
Get operations are allowed.

MQQA_GET_INHIBITED
Get operations are inhibited.

BaseQName (MQCFST)
Queue name to which the alias resolves (parameter identifier: MQCA_BASE_Q_NAME).

This is the name of a queue that is defined to the local queue manager.

The maximum length of the string is MQ_Q_NAME_LENGTH.

ProcessName (MQCFST)
Name of process definition for the queue (parameter identifier: MQCA_PROCESS_NAME).

Specifies the local name of the MQSeries process that identifies the application that should be started when a trigger event occurs.

- On AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT, if the queue is a transmission queue the process name can be left as all blanks.
- On 32-bit Windows, this parameter is accepted but ignored.
- In other environments, the process name must be nonblank for a trigger event to occur (although it can be set after the queue has been created).

MaxQDepth (MQCFIN)
Maximum queue depth (parameter identifier: MQIA_MAX_Q_DEPTH).

The maximum number of messages allowed on the queue. Note that other factors may cause the queue to be treated as full; for example, it will be appear to be full if there is no storage available for a message.

Specify a value in the range 0 through 640 000.

MaxMsgLength (MQCFIN)
Maximum message length (parameter identifier: MQIA_MAX_MSG_LENGTH).

Specifies the maximum length for messages on the queue. Because applications may use the value of this attribute to determine the size of buffer they need to retrieve messages from the queue, the value should be changed only if it is known that this will not cause an application to operate incorrectly.

You are recommended not to set a value that is greater than the queue manager's *MaxMsgLength* attribute.

The lower limit for this parameter is 0. The upper limit depends on the environment:

- On AIX, HP-UX, OS/2, Sun Solaris, and Windows NT, the maximum message length is 100 MB (104 857 600 bytes).
- On OpenVMS, OS/400, Tandem NSK, UNIX systems not listed above, and 32-bit Windows, the maximum message length is 4 MB (4 194 304 bytes).

BackoutThreshold (MQCFIN)
Backout threshold (parameter identifier: MQIA_BACKOUT_THRESHOLD).

That is, the number of times a message can be backed out before it is transferred to the backout queue specified by *BackoutRequeueName*.

If the value is subsequently reduced, any messages already on the queue that have been backed out at least as many times as the new value remain on the queue, but such messages are transferred if they are backed out again.

Specify a value in the range 0 through 999 999 999.

BackoutRequeueName (MQCFST)
Excessive backout requeue name (parameter identifier: MQCA_BACKOUT_REQ_Q_NAME).

Specifies the local name of the queue (not necessarily a local queue) to which a message is transferred if it is backed out more times than the value of *BackoutThreshold*.

The backout queue does not need to exist at this time but it must exist when the *BackoutThreshold* value is exceeded.

The maximum length of the string is MQ_Q_NAME_LENGTH.

Shareability (MQCFIN)
Whether queue can be shared (parameter identifier: MQIA_SHAREABILITY).

Specifies whether multiple instances of applications, can open this queue for input.

The value may be:

MQQA_SHAREABLE
Queue is shareable.

MQQA_NOT_SHAREABLE
Queue is not shareable.

DefInputOpenOption (MQCFIN)
Default input open option (parameter identifier: MQIA_DEF_INPUT_OPEN_OPTION).

Specifies the default share option for applications opening this queue for input.

The value may be:

MQOO_INPUT_EXCLUSIVE
Open queue to get messages with exclusive access.

MQOO_INPUT_SHARED
Open queue to get messages with shared access.

HardenGetBackout (MQCFIN)
Whether to harden backout (parameter identifier: MQIA_HARDEN_GET_BACKOUT).

Specifies whether the count of backed out messages should be saved (hardened) across restarts of the queue manager.

Note: MQSeries for AS/400 always hardens the count, regardless of the setting of this attribute.

The value may be:

MQQA_BACKOUT_HARDENED
Backout count remembered.

MQQA_BACKOUT_NOT_HARDENED
Backout count may not be remembered.

MsgDeliverySequence (MQCFIN)
Whether priority is relevant (parameter identifier: MQIA_MSG_DELIVERY_SEQUENCE).

The value may be:

MQMDS_PRIORITY
Messages are returned in priority order.

MQMDS_FIFO
Messages are returned in FIFO order (first in, first out).

RetentionInterval (MQCFIN)
Retention interval (parameter identifier: MQIA_RETENTION_INTERVAL).

The number of hours for which the queue may be needed, based on the date and time when the queue was created.

This information is available to a housekeeping application or an operator and may be used to determine when a queue is no longer required. The queue manager does not delete queues nor does it prevent queues from being deleted if their retention interval has not expired. It is the user's responsibility to take any required action.

Specify a value in the range 0 through 999 999 999.

DistLists (MQCFIN)
Distribution list support (parameter identifier: MQIA_DIST_LISTS).

Specifies whether distribution-list messages can be placed on the queue.

Note: This attribute is set by the sending message channel agent (MCA) which removes messages from the queue; this happens each time the sending MCA establishes a connection to a receiving MCA on a partnering queue manager. The attribute should not normally be set by administrators, although it can be set if the need arises.

This parameter is supported in the following environments: AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

The value may be:

MQDL_SUPPORTED
Distribution lists supported.

MQDL_NOT_SUPPORTED
Distribution lists not supported.

Usage (MQCFIN)
Usage (parameter identifier: MQIA_USAGE).

Specifies whether the queue is for normal usage or for transmitting messages to a remote message queue manager.

The value may be:

MQUS_NORMAL
Normal usage.

MQUS_TRANSMISSION
Transmission queue.

InitiationQName (MQCFST)
Initiation queue name (parameter identifier: MQCA_INITIATION_Q_NAME).

The local queue for trigger messages relating to the new, or changed, queue. The initiation queue must be on the same queue manager.

On 32-bit Windows, this parameter is accepted but ignored.

The maximum length of the string is MQ_Q_NAME_LENGTH.

TriggerControl (MQCFIN)
Trigger control (parameter identifier: MQIA_TRIGGER_CONTROL).

Specifies whether trigger messages are written to the initiation queue.

The value may be:

MQTC_OFF
Trigger messages not required.

MQTC_ON
Trigger messages required.

This value is not supported on 32-bit Windows.

TriggerType (MQCFIN)
Trigger type (parameter identifier: MQIA_TRIGGER_TYPE).

Specifies the condition that initiates a trigger event. When the condition is true, a trigger message is sent to the initiation queue.

On 32-bit Windows, this parameter is accepted but ignored.

The value may be:

MQTT_NONE
No trigger messages.

MQTT_EVERY
Trigger message for every message.

MQTT_FIRST
Trigger message when queue depth goes from 0 to 1.

MQTT_DEPTH
Trigger message when depth threshold exceeded.

TriggerMsgPriority (MQCFIN)
Threshold message priority for triggers (parameter identifier: MQIA_TRIGGER_MSG_PRIORITY).

Specifies the minimum priority that a message must have before it can cause, or be counted for, a trigger event. The value must be in the range of priority values that are supported (0 through 9).

On 32-bit Windows, this parameter is accepted but ignored.

TriggerDepth (MQCFIN)
Trigger depth (parameter identifier: MQIA_TRIGGER_DEPTH).

Specifies (when *TriggerType* is MQTT_DEPTH) the number of messages that will initiate a trigger message to the initiation queue. The value must be in the range 1 through 999 999 999.

On 32-bit Windows, this parameter is accepted but ignored.

TriggerData (MQCFST)
Trigger data (parameter identifier: MQCA_TRIGGER_DATA).

Specifies user data that the queue manager includes in the trigger message. This data is made available to the monitoring application that processes the initiation queue and to the application that is started by the monitor.

On 32-bit Windows, this parameter is accepted but ignored.

The maximum length of the string is MQ_TRIGGER_DATA_LENGTH.

RemoteQName (MQCFST)
Name of remote queue as known locally on the remote queue manager (parameter identifier: MQCA_REMOTE_Q_NAME).

If this definition is used for a local definition of a remote queue, *RemoteQName* must not be blank when the open occurs.

If this definition is used for a queue-manager alias definition, *RemoteQName* must be blank when the open occurs.

If this definition is used for a reply-to alias, this name is the name of the queue that is to be the reply-to queue.

The maximum length of the string is MQ_Q_NAME_LENGTH.

RemoteQMgrName (MQCFST)
Name of remote queue manager (parameter identifier: MQCA_REMOTE_Q_MGR_NAME).

If an application opens the local definition of a remote queue, *RemoteQMgrName* must not be blank or the name of the connected queue manager. If *XmitQName* is blank there must be a local queue of this name, which is to be used as the transmission queue.

If this definition is used for a queue-manager alias, *RemoteQMgrName* is the name of the queue manager, which can be the name of the connected queue manager. Otherwise, if *XmitQName* is blank, when the queue is opened there must be a local queue of this name, which is to be used as the transmission queue.

If this definition is used for a reply-to alias, this name is the name of the queue manager that is to be the reply-to queue manager.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

XmitQName (MQCFST)
Transmission queue name (parameter identifier: MQCA_XMIT_Q_NAME).

Specifies the local name of the transmission queue to be used for messages destined for the remote queue, for either a remote queue or for a queue-manager alias definition.

If *XmitQName* is blank, a queue with the same name as *RemoteQMgrName* is used as the transmission queue.

This attribute is ignored if the definition is being used as a queue-manager alias and *RemoteQMgrName* is the name of the connected queue manager.

It is also ignored if the definition is used as a reply-to queue alias definition.

The maximum length of the string is MQ_Q_NAME_LENGTH.

DefinitionType (MQCFIN)
Queue definition type (parameter identifier: MQIA_DEFINITION_TYPE).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

The value may be:

MQQDT_PERMANENT_DYNAMIC
Dynamically defined permanent queue.

MQQDT_TEMPORARY_DYNAMIC
Dynamically defined temporary queue.

Scope (MQCFIN)
Scope of the queue definition (parameter identifier: MQIA_SCOPE).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

Specifies whether the scope of the queue definition does not extend beyond the queue manager which owns the queue, or whether the queue name is contained in a cell directory, so that it is known to all of the queue managers within the cell.

Model and dynamic queues cannot have cell scope.

The command fails if the new queue has a *Scope* attribute of MQSCO_CELL, but no name service supporting a cell directory has been configured.

The value may be:

MQSCO_Q_MGR
Queue-manager scope.

MQSCO_CELL
Cell scope.

This value is not supported on OS/400 and 32-bit Windows.

QDepthHighLimit (MQCFIN)
High limit for queue depth (parameter identifier: MQIA_Q_DEPTH_HIGH_LIMIT).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

The threshold against which the queue depth is compared to generate a Queue Depth High event.

This event indicates that an application has put a message to a queue, and this has caused the number of messages on the queue to become greater than or equal to the queue depth high threshold. See the *QDepthHighEvent* parameter.

The value is expressed as a percentage of the maximum queue depth (*MaxQDepth* attribute), and must be greater than or equal to zero and less than or equal to 100.

QDepthLowLimit (MQCFIN)
Low limit for queue depth (parameter identifier: MQIA_Q_DEPTH_LOW_LIMIT).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

The threshold against which the queue depth is compared to generate a Queue Depth Low event.

This event indicates that an application has retreived a message from a queue, and this has caused the number of messages on the queue to become less than or equal to the queue depth low threshold. See the *QDepthLowEvent* parameter.

The value is expressed as a percentage of the maximum queue depth (*MaxQDepth* attribute), and must be greater than or equal to zero and less than or equal to 100.

QDepthMaxEvent (MQCFIN)
Controls whether Queue Full events are generated (parameter identifier: MQIA_Q_DEPTH_MAX_EVENT).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

A Queue Full event indicates that an **MQPUT** call to a queue has been rejected because the queue is full, that is, the queue depth has already reached its maximum value.

Note: The value of this attribute can change implicitly. See Chapter 3, “Understanding performance events” on page 17.

The value may be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

QDepthHighEvent (MQCFIN)
Controls whether Queue Depth High events are generated (parameter identifier: MQIA_Q_DEPTH_HIGH_EVENT).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

A Queue Depth High event indicates that an application has put a message on a queue, and this has caused the number of messages on the queue to become greater than or equal to the queue depth high threshold (see the *QDepthHighLimit* parameter).

Note: The value of this attribute can change implicitly. See Chapter 3, “Understanding performance events” on page 17.

The value may be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

QDepthLowEvent (MQCFIN)
Controls whether Queue Depth Low events are generated (parameter identifier: MQIA_Q_DEPTH_LOW_EVENT).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

A Queue Depth Low event indicates that an application has retrieved a message from a queue, and this has caused the number of messages on the queue to become less than or equal to the queue depth low threshold (see the *QDepthLowLimit* parameter).

Note: The value of this attribute can change implicitly. See Chapter 3, "Understanding performance events" on page 17.

The value may be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

QServiceInterval (MQCFIN)
Target for queue service interval (parameter identifier: MQIA_Q_SERVICE_INTERVAL).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

The service interval used for comparison to generate Queue Service Interval High and Queue Service Interval OK events. See the *QServiceIntervalEvent* parameter.

The value is in units of milliseconds, and must be greater than or equal to zero, and less than or equal to 999 999 999.

QServiceIntervalEvent (MQCFIN)
Controls whether Queue Service Interval High or Queue Service Interval OK events are generated (parameter identifier: MQIA_Q_SERVICE_INTERVAL_EVENT).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

A Queue Service Interval High event is generated when a check indicates that no messages have been retrieved from the queue for at least the time indicated by the *QServiceInterval* attribute.

A Queue Service Interval OK event is generated when a check indicates that messages have been retrieved from the queue within the time indicated by the *QServiceInterval* attribute.

Note: The value of this attribute can change implicitly. See Chapter 3, "Understanding performance events" on page 17.

The value may be:

MQQSIE_HIGH
Queue Service Interval High events enabled.

- Queue Service Interval High events are **enabled** and
- Queue Service Interval OK events are **disabled**.

MQQSIE_OK
Queue Service Interval OK events enabled.

- Queue Service Interval High events are **disabled** and
- Queue Service Interval OK events are **enabled**.

MQQSIE_NONE
No queue service interval events enabled.

- Queue Service Interval High events are **disabled** and
- Queue Service Interval OK events are also **disabled**.

Error codes

In addition to the values for any command shown on page 136, for this command the following may be returned in the response format header:

Reason (MQLONG)
The value may be:

MQRC_UNKNOWN_OBJECT_NAME
(2085, X'825') Unknown object name.

MQRCCF_ATTR_VALUE_ERROR
Attribute value not valid.

MQRCCF_CELL_DIR_NOT_AVAILABLE
Cell directory is not available.

MQRCCF_CFIN_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFIN_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIN_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_DYNAMIC_Q_SCOPE_ERROR
Dynamic queue scope error.

MQRCCF_LIKE_OBJECT_WRONG_TYPE
New and existing objects have different type.

MQRCCF_OBJECT_ALREADY_EXISTS
Object already exists.

MQRCCF_OBJECT_NAME_ERROR
Object name not valid.

MQRCCF_OBJECT_OPEN
Object is open.

MQRCCF_OBJECT_WRONG_TYPE
Object has wrong type.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_PARM_SEQUENCE_ERROR
Parameter sequence not valid.

MQRCCF_Q_ALREADY_IN_CELL
Queue already exists in cell.

MQRCCF_Q_TYPE_ERROR
Queue type not valid.

MQRCCF_REPLACE_VALUE_ERROR
Replace value not valid.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

Delete Channel

The Delete Channel (MQCMD_DELETE_CHANNEL) command deletes the specified channel definition.

Required parameters:

ChannelName

Optional parameters:

ChannelTable

Required parameters

ChannelName (MQCFST)

Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The name of the channel definition to be deleted. The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Optional parameters

ChannelTable (MQCFIN)

Channel table (parameter identifier: MQIACH_CHANNEL_TABLE).

Specifies the ownership of the channel definition table that contains the specified channel definition.

The value may be:

MQCHTAB_Q_MGR
: Queue-manager table.

  This is the default. This table contains channel definitions for channels of all types except MQCHT_CLNTCONN.

MQCHTAB_CLNTCONN
: Client-connection table.

  This table only contains channel definitions for channels of type MQCHT_CLNTCONN.

  On OS/400 and 32-bit Windows, this value is not supported.

Error codes

In addition to the values for any command shown on page 136, for this command the following may be returned in the response format header:

Reason (MQLONG)

The value may be:

MQRCCF_CFST_DUPLICATE_PARM
: Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
: Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
: Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
: String length not valid.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_CHANNEL_TABLE_ERROR
Channel table value not valid.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

Delete Process

The Delete Process (MQCMD_DELETE_PROCESS) command deletes an existing MQSeries process definition.

| *This PCF is not supported if you are using MQSeries for Windows Version 2.1.*

Required parameters:
ProcessName

Optional parameters:
None

Required parameters

ProcessName (MQCFST)
Process name (parameter identifier: MQCA_PROCESS_NAME).

The process definition to be deleted. The maximum length of the string is MQ_PROCESS_NAME_LENGTH.

Error codes

In addition to the values for any command shown on page 136, for this command the following may be returned in the response format header:

Reason (MQLONG)
The value may be:

MQRC_UNKNOWN_OBJECT_NAME
(2085, X'825') Unknown object name.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_OBJECT_OPEN
Object is open.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

Delete Queue

The Delete Queue (MQCMD_DELETE_Q) command deletes an MQSeries queue.

Required parameters:

QName

Optional parameters (any QType):

QType

Optional parameters (local QType only):

Purge

Required parameters

QName (MQCFST)

Queue name (parameter identifier: MQCA_Q_NAME).

The name of the queue to be deleted.

If the *Scope* attribute of the queue is MQSCO_CELL, the entry for the queue is deleted from the cell directory.

The maximum length of the string is MQ_Q_NAME_LENGTH.

Optional parameters

QType (MQCFIN)

Queue type (parameter identifier: MQIA_Q_TYPE).

If this parameter is present, the queue must be of the specified type.

The value may be:

MQQT_ALIAS
: Alias queue definition.

MQQT_LOCAL
: Local queue.

MQQT_REMOTE
: Local definition of a remote queue.

The following is supported on all platforms, but on OS/400 for receipt by MQSeries for AS/400 V4R2:

MQQT_MODEL
: Model queue definition.

Purge (MQCFIN)

Purge queue (parameter identifier: MQIACF_PURGE).

If there are messages on the queue MQPO_YES must be specified, otherwise the command will fail. If this parameter is not present the queue is not purged.

Valid only for queue of type local.

The value may be:

MQPO_YES
: Purge the queue.

MQPO_NO
Do not purge the queue.

Error codes

In addition to the values for any command shown on page 136, for this command the following may be returned in the response format header:

Reason (MQLONG)
The value may be:

MQRC_Q_NOT_EMPTY
(2055, X'807') Queue contains one or more messages or uncommitted put or get requests.

MQRC_UNKNOWN_OBJECT_NAME
(2085, X'825') Unknown object name.

MQRCCF_CFIN_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFIN_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIN_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_OBJECT_OPEN
Object is open.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_PURGE_VALUE_ERROR
Purge value not valid.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

Escape

The Escape (MQCMD_ESCAPE) command conveys any MQSeries command (MQSC) to a remote queue manager. Use it when the queue manager (or application) sending the command does not support the functionality of the particular MQSeries command, and so does not recognize it and cannot construct the required PCF command.

The Escape command can also be used to send a command for which no Programmable Command Format has been defined.

The only type of command that can be carried is one that is identified as an MQSC, that is recognised at the receiving queue manager.

| *This PCF is not supported if you are using MQSeries for Windows Version 2.1.*

Required parameters:
EscapeType, *EscapeText*

Optional parameters:
None

Required parameters

EscapeType (MQCFIN)
Escape type (parameter identifier: MQIACF_ESCAPE_TYPE).

The only value supported is:

MQET_MQSC
MQSeries command.

EscapeText (MQCFST)
Escape text (parameter identifier: MQCACF_ESCAPE_TEXT).

A string to hold a command. The length of the string is limited only by the size of the message.

Error codes

In addition to the values for any command shown on page 136, for this command the following may be returned in the response format header:

Reason (MQLONG)
The value may be:

MQRCCF_ESCAPE_TYPE_ERROR
Escape type not valid.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_PARM_SEQUENCE_ERROR
Parameter sequence not valid.

Escape (Response)

The response to the Escape (MQCMD_ESCAPE) command consists of the response header followed by two parameter structures, one containing the escape type, and the other containing the text response. More than one such message may be issued, depending upon the command contained in the Escape request.

The *Command* field in the response header MQCFH contains the MQCMD_* command identifier of the text command contained in the *EscapeText* parameter in the original Escape command. For example, if *EscapeText* in the original Escape command specified PING QMGR, *Command* in the response has the value MQCMD_PING_Q_MGR.

If it is possible to determine the outcome of the command, the *CompCode* in the response header identifies whether the command was successful. The success or otherwise can therefore be determined without the recipient of the response having to parse the text of the response.

If it is not possible to determine the outcome of the command, *CompCode* in the response header has the value MQCC_UNKNOWN, and *Reason* is MQRC_NONE.

This command is not supported on 32-bit Windows.

Always returned:
EscapeType, *EscapeText*

Returned if requested:
None

Parameters

EscapeType (MQCFIN)
Escape type (parameter identifier: MQIACF_ESCAPE_TYPE).

The only value supported is:

MQET_MQSC
MQSeries command.

EscapeText (MQCFST)
Escape text (parameter identifier: MQCACF_ESCAPE_TEXT).

A string holding the response to the original command.

Inquire Channel

The Inquire Channel (MQCMD_INQUIRE_CHANNEL) command inquires about the attributes of MQSeries channel definitions.

Required parameters:

ChannelName

Optional parameters:

ChannelType, *ChannelAttrs*

Required parameters

ChannelName (MQCFST)

Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

Generic channel names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all channels having names that start with the selected character string. An asterisk on its own matches all possible names.

The channel name is always returned, regardless of the attributes requested.

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Optional parameters

ChannelType (MQCFIN)

Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

If this parameter is present, eligible channels are limited to those of the specified type. Any attribute selector specified in the *ChannelAttrs* list which is only valid for channels of a different type or types is ignored; no error is raised.

If this parameter is not present (or if MQCHT_ALL is specified), channels of all types except MQCHT_CLNTCONN are eligible. Each attribute specified must be a valid channel attribute selector (that is, it must be one of those in the following list), but it may not be applicable to all (or any) of the channels actually returned. Channel attribute selectors that are valid but not applicable to the channel are ignored, no error messages occur, and no attribute is returned.

The value may be:

MQCHT_SENDER
: Sender.

MQCHT_SERVER
: Server.

MQCHT_RECEIVER
: Receiver.

MQCHT_REQUESTER
: Requester.

MQCHT_SVRCONN
Server-connection (for use by clients).

This value is not supported in the following environment: 32-bit Windows.

MQCHT_CLNTCONN
Client connection.

This value is not supported in the following environments: OS/400, 32-bit Windows.

MQCHT_ALL
All types.

The default value if this parameter is not specified is MQCHT_ALL.

Note: If this parameter is present, it must occur immediately after the *ChannelName* parameter. Failure to do this can result in a MQRCCF_MSG_LENGTH_ERROR error message.

ChannelAttrs (MQCFIL)
Channel attributes (parameter identifier: MQIACF_CHANNEL_ATTRS).

The attribute list may specify the following on its own (this is the default value used if the parameter is not specified):

MQIACF_ALL
All attributes.

or a combination of the following:

Relevant for any channel type:

MQCACH_CHANNEL_NAME
Channel name.

MQIACH_CHANNEL_TYPE
Channel type.

MQIACH_XMIT_PROTOCOL_TYPE
Transport (transmission protocol) type.

MQCACH_DESC
Description.

MQCACH_SEC_EXIT_NAME
Security exit name.

MQCACH_MSG_EXIT_NAME
Message exit name.

MQCACH_SEND_EXIT_NAME
Send exit name.

MQCACH_RCV_EXIT_NAME
Receive exit name.

MQIACH_MAX_MSG_LENGTH
Maximum message length.

MQCACH_SEC_EXIT_USER_DATA
Security exit user data.

MQCACH_MSG_EXIT_USER_DATA
Message exit user data.

MQCACH_SEND_EXIT_USER_DATA
Send exit user data.

MQCACH_RCV_EXIT_USER_DATA
Receive exit user data.

Relevant for sender or server channel types:

MQCACH_XMIT_Q_NAME
Transmission queue name.

MQCACH_MCA_NAME
Message channel agent name.

MQCACH_MODE_NAME
Mode name.

MQCACH_TP_NAME
Transaction program name.

MQIACH_BATCH_SIZE
Batch size.

MQIACH_DISC_INTERVAL
Disconnection interval.

MQIACH_SHORT_RETRY
Short retry count.

MQIACH_SHORT_TIMER
Short timer.

MQIACH_LONG_RETRY
Long retry count.

MQIACH_LONG_TIMER
Long timer.

MQIACH_SEQUENCE_NUMBER_WRAP
Sequence number wrap.

MQIACH_DATA_CONVERSION
Whether sender should convert application data.

MQIACH_MCA_TYPE
MCA type.

MQCACH_MCA_USER_ID
MCA user identifier.

| The following is supported on OpenVMS, OS/2, OS/400, Tandem NSK, UNIX systems, 32-bit Windows, and Windows NT:

MQCACH_CONNECTION_NAME
Connection name.

| The following are supported on OpenVMS, OS/2, Tandem NSK, UNIX systems, and Windows NT:

MQCACH_USER_ID
User identifier.

MQCACH_PASSWORD
Password.

The following are supported on AIX, HP-UX, OS/2, OS/400, Sun Solaris, and Windows NT:

MQIACH_BATCH_INTERVAL
Batch wait interval (seconds).

MQIACH_HB_INTERVAL
Heartbeat interval (seconds).

The following is supported on AIX, HP-UX, OS/2, OS/400, Sun Solaris, 32-bit Windows, and Windows NT:

MQIACH_NPM_SPEED
Speed of nonpersistent messages.

Relevant for requester channel type:

MQCACH_MCA_NAME
Message channel agent name.

MQCACH_MODE_NAME
Mode name.

MQCACH_TP_NAME
Transaction program name.

MQIACH_BATCH_SIZE
Batch size.

MQIACH_SEQUENCE_NUMBER_WRAP
Sequence number wrap.

MQIACH_PUT_AUTHORITY
Put authority.

MQCACH_MR_EXIT_NAME
Message-retry exit name.

MQCACH_MR_EXIT_USER_DATA
Message-retry exit user data.

MQIACH_MR_COUNT
Message retry count.

MQIACH_MR_INTERVAL
Message retry interval (milliseconds).

MQIACH_MCA_TYPE
MCA type.

MQCACH_MCA_USER_ID
MCA user identifier.

The following is supported on OpenVMS, OS/2, OS/400, Tandem NSK, UNIX systems, 32-bit Windows, and Windows NT:

MQCACH_CONNECTION_NAME
Connection name.

The following are supported on OpenVMS, OS/2, Tandem NSK, UNIX systems, and Windows NT:

MQCACH_USER_ID
User identifier.

MQCACH_PASSWORD
Password.

The following is supported on AIX, HP-UX, OS/2, OS/400, Sun Solaris, and Windows NT:

MQIACH_HB_INTERVAL
Heartbeat interval (seconds).

The following is supported on AIX, HP-UX, OS/2, OS/400, Sun Solaris, 32-bit Windows, and Windows NT:

MQIACH_NPM_SPEED
Speed of nonpersistent messages.

Relevant for receiver channel type:

MQIACH_BATCH_SIZE
Batch size.

MQIACH_SEQUENCE_NUMBER_WRAP
Sequence number wrap.

MQIACH_PUT_AUTHORITY
Put authority.

MQCACH_MR_EXIT_NAME
Message-retry exit name.

MQCACH_MR_EXIT_USER_DATA
Message-retry exit user data.

MQIACH_MR_COUNT
Message retry count.

MQIACH_MR_INTERVAL
Message retry interval (milliseconds).

MQCACH_MCA_USER_ID
MCA user identifier.

The following is supported on AIX, HP-UX, OS/2, OS/400, Sun Solaris, and Windows NT:

MQIACH_HB_INTERVAL
Heartbeat interval (seconds).

The following is supported on AIX, HP-UX, OS/2, OS/400, Sun Solaris, 32-bit Windows, and Windows NT:

MQIACH_NPM_SPEED
Speed of nonpersistent messages.

Relevant for server-connection channel type

The following is supported on OpenVMS, OS/2, OS/400, Tandem NSK, UNIX systems, and Windows NT:

MQCACH_MCA_USER_ID
MCA user identifier.

Relevant for client-connection channel type

The following are supported on OpenVMS, OS/2, Tandem NSK, UNIX systems, and Windows NT:

MQCACH_MODE_NAME
Mode name.

MQCACH_TP_NAME
Transaction program name.

MQCA_Q_MGR_NAME
Name of local queue manager.

MQCACH_CONNECTION_NAME
Connection name.

The following are supported on OpenVMS, OS/2 Tandem NSK, and UNIX systems:

MQCACH_USER_ID
User identifier.

MQCACH_PASSWORD
Password.

Error codes

In addition to the values for any command shown on page 136, for this command the following may be returned in the response format header:

Reason (MQLONG)
The value may be:

MQRC_SELECTOR_ERROR
(2067, X'813') Attribute selector not valid.

MQRCCF_CFIL_COUNT_ERROR
Count of parameter values not valid.

MQRCCF_CFIL_DUPLICATE_VALUE
Duplicate parameter.

MQRCCF_CFIL_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIL_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFIN_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFIN_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIN_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_CHANNEL_NAME_ERROR
Channel name error.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_CHANNEL_TYPE_ERROR
Channel type not valid.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

Inquire Channel (Response)

The response to the Inquire Channel (MQCMD_INQUIRE_CHANNEL) command consists of the response header followed by the *ChannelName* structure and the requested combination of attribute parameter structures (where applicable). If a generic channel name was specified, one such message is generated for each channel found.

Always returned:

ChannelName

Returned if requested:

ChannelType, *TransportType*, *ModeName*, *TpName*, *QMgrName*, *XmitQName*, *ConnectionName*, *MCAName*, *ChannelDesc*, *BatchSize*, *DiscInterval*, *ShortRetryCount*, *ShortRetryInterval*, *LongRetryCount*, *LongRetryInterval*, *DataConversion*, *SecurityExit*, *MsgExit*, *SendExit*, *ReceiveExit*, *PutAuthority*, *SeqNumberWrap*, *MaxMsgLength*, *SecurityUserData*, *MsgUserData*, *SendUserData*, *ReceiveUserData*, *MCAType*, *MCAUserIdentifier*, *UserIdentifier*, *Password*, *MsgRetryExit*, *MsgRetryUserData*, *MsgRetryCount*, *MsgRetryInterval*, *HeartbeatInterval*, *NonPersistentMsgSpeed* *BatchInterval*

Response data

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

ChannelType (MQCFIN)
Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

The value may be:

MQCHT_SENDER
Sender.

MQCHT_SERVER
Server.

MQCHT_RECEIVER
Receiver.

MQCHT_REQUESTER
Requester.

MQCHT_SVRCONN
Server-connection (for use by clients).

MQCHT_CLNTCONN
Client connection.

TransportType (MQCFIN)
Transmission protocol type (parameter identifier: MQIACH_XMIT_PROTOCOL_TYPE).

The value may be:

MQXPT_LU62
LU 6.2.

MQXPT_TCP
TCP/IP.
MQXPT_NETBIOS
NetBIOS.
MQXPT_SPX
SPX.
MQXPT_DECNET
DECnet.

ModeName (MQCFST)
Mode name (parameter identifier: MQCACH_MODE_NAME).

The maximum length of the string is MQ_MODE_NAME_LENGTH.

TpName (MQCFST)
Transaction program name (parameter identifier: MQCACH_TP_NAME).

The maximum length of the string is MQ_TP_NAME_LENGTH.

QMgrName (MQCFST)
Queue manager name (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

XmitQName (MQCFST)
Transmission queue name (parameter identifier: MQCACH_XMIT_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

ConnectionName (MQCFST)
Connection name (parameter identifier: MQCACH_CONNECTION_NAME).

The maximum length of the string is MQ_CONN_NAME_LENGTH.

MCAName (MQCFST)
Message channel agent name (parameter identifier: MQCACH_MCA_NAME).

The maximum length of the string is MQ_MCA_NAME_LENGTH.

ChannelDesc (MQCFST)
Channel description (parameter identifier: MQCACH_DESC).

The maximum length of the string is MQ_CHANNEL_DESC_LENGTH.

BatchSize (MQCFIN)
Batch size (parameter identifier: MQIACH_BATCH_SIZE).

DiscInterval (MQCFIN)
Disconnection interval (parameter identifier: MQIACH_DISC_INTERVAL).

ShortRetryCount (MQCFIN)
Short retry count (parameter identifier: MQIACH_SHORT_RETRY).

ShortRetryInterval (MQCFIN)
Short timer (parameter identifier: MQIACH_SHORT_TIMER).

LongRetryCount (MQCFIN)
Long retry count (parameter identifier: MQIACH_LONG_RETRY).

LongRetryInterval (MQCFIN)
Long timer (parameter identifier: MQIACH_LONG_TIMER).

DataConversion (MQCFIN)
Whether sender should convert application data (parameter identifier: MQIACH_DATA_CONVERSION).

The value may be:

MQCDC_NO_SENDER_CONVERSION
No conversion by sender.

MQCDC_SENDER_CONVERSION
Conversion by sender.

SecurityExit (MQCFST)
Security exit name (parameter identifier: MQCACH_SEC_EXIT_NAME).

The maximum length of the string is MQ_EXIT_NAME_LENGTH.

MsgExit (MQCFST)
Message exit name (parameter identifier: MQCACH_MSG_EXIT_NAME).

The maximum length of the string is MQ_EXIT_NAME_LENGTH.

In the following environments, if more than one message exit has been defined for the channel, the list of names is returned in an MQCFSL structure instead of an MQCFST structure: AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

SendExit (MQCFST)
Send exit name (parameter identifier: MQCACH_SEND_EXIT_NAME).

The maximum length of the string is MQ_EXIT_NAME_LENGTH.

In the following environments, if more than one send exit has been defined for the channel, the list of names is returned in an MQCFSL structure instead of an MQCFST structure: AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

ReceiveExit (MQCFST)
Receive exit name (parameter identifier: MQCACH_RCV_EXIT_NAME).

The maximum length of the string is MQ_EXIT_NAME_LENGTH.

In the following environments, if more than one receive exit has been defined for the channel, the list of names is returned in an MQCFSL structure instead of an MQCFST structure: AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

PutAuthority (MQCFIN)
Put authority (parameter identifier: MQIACH_PUT_AUTHORITY).

The value may be:

MQPA_DEFAULT
Default user identifier is used.

MQPA_CONTEXT
Context user identifier is used.

SeqNumberWrap (MQCFIN)
Sequence wrap number (parameter identifier: MQIACH_SEQUENCE_NUMBER_WRAP).

MaxMsgLength (MQCFIN)
Maximum message length (parameter identifier: MQIACH_MAX_MSG_LENGTH).

SecurityUserData (MQCFST)
Security exit user data (parameter identifier: MQCACH_SEC_EXIT_USER_DATA).

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

MsgUserData (MQCFST)
Message exit user data (parameter identifier: MQCACH_MSG_EXIT_USER_DATA).

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

In the following environments, if more than one message exit user data string has been defined for the channel, the list of strings is returned in an MQCFSL structure instead of an MQCFST structure: AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

SendUserData (MQCFST)
Send exit user data (parameter identifier: MQCACH_SEND_EXIT_USER_DATA).

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

In the following environments, if more than one send exit user data string has been defined for the channel, the list of strings is returned in an MQCFSL structure instead of an MQCFST structure: AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

ReceiveUserData (MQCFST)
Receive exit user data (parameter identifier: MQCACH_RCV_EXIT_USER_DATA).

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

In the following environments, if more than one receive exit user data string has been defined for the channel, the list of strings is returned in an MQCFSL structure instead of an MQCFST structure: AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

MCAType (MQCFIN)
Message channel agent type (parameter identifier: MQIACH_MCA_TYPE).

The value may be:

MQMCAT_PROCESS
Process.

MQMCAT_THREAD
Thread (OS/2 and Windows NT only).

MCAUserIdentifier (MQCFST)
Message channel agent user identifier (parameter identifier: MQCACH_MCA_USER_ID).

The maximum length of the string is MQ_USER_ID_LENGTH.

UserIdentifier (MQCFST)
Task user identifier (parameter identifier: MQCACH_USER_ID).

The maximum length of the string is MQ_USER_ID_LENGTH. However, only the first 10 characters are used.

Password (MQCFST)
Password (parameter identifier: MQCACH_PASSWORD).

If a nonblank password is defined, it is returned as asterisks. Otherwise, it is returned as blanks.

The maximum length of the string is MQ_PASSWORD_LENGTH. However, only the first 10 characters are used.

MsgRetryExit (MQCFST)
Message retry exit name (parameter identifier: MQCACH_MR_EXIT_NAME).

The maximum length of the string is MQ_EXIT_NAME_LENGTH.

MsgRetryUserData (MQCFST)
Message retry exit user data (parameter identifier: MQCACH_MR_EXIT_USER_DATA).

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

MsgRetryCount (MQCFIN)
Message retry count (parameter identifier: MQIACH_MR_COUNT).

MsgRetryInterval (MQCFIN)
Message retry interval (parameter identifier: MQIACH_MR_INTERVAL).

BatchInterval (MQCFIN)
Batch interval (parameter identifier: MQIACH_BATCH_INTERVAL).

HeartbeatInterval (MQCFIN)
Heartbeat interval (parameter identifier: MQIACH_HB_INTERVAL).

NonPersistentMsgSpeed (MQCFIN)
Speed at which non-persistent messages are to be sent (parameter identifier: MQIACH_NPM_SPEED).

The value may be:

MQNPMS_NORMAL
Normal speed.

MQNPMS_FAST
Fast speed.

Inquire Channel Names

The Inquire Channel Names (MQCMD_INQUIRE_CHANNEL_NAMES) command inquires a list of MQSeries channel names that match the generic channel name, and the optional channel type specified.

Required parameters:

ChannelName

Optional parameters:

ChannelType

Required parameters

ChannelName (MQCFST)

Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

Generic channel names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all objects having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Optional parameters

ChannelType (MQCFIN)

Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

If present, this parameter limits the channel names returned to channels of the specified type.

The value may be:

MQCHT_SENDER
: Sender.

MQCHT_SERVER
: Server.

MQCHT_RECEIVER
: Receiver.

MQCHT_REQUESTER
: Requester.

MQCHT_SVRCONN
: Server-connection (for use by clients).

  This value is not supported in the following environment: 32-bit Windows.

MQCHT_CLNTCONN
: Client connection.

  This value is not supported in the following environments: OS/400, 32-bit Windows.

MQCHT_ALL
: All types.

The default value if this parameter is not specified is MQCHT_ALL, which means that channels of all types except MQCHT_CLNTCONN are eligible.

Error codes

In addition to the values for any command shown on page 136, for this command the following may be returned in the response format header:

Reason (MQLONG)
The value may be:

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_CHANNEL_NAME_ERROR
Channel name error.

MQRCCF_CHANNEL_TYPE_ERROR
Channel type not valid.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

Inquire Channel Names (Response)

The response to the Inquire Channel Names (MQCMD_INQUIRE_CHANNEL_NAMES) command consists of the response header followed by a single parameter structure giving zero or more names that match the specified channel name.

Always returned:

ChannelNames

Returned if requested:

None

Response data

ChannelNames (MQCFSL)

Channel names (parameter identifier: MQCACH_CHANNEL_NAMES).

Inquire Channel Status

The Inquire Channel Status (MQCMD_INQUIRE_CHANNEL_STATUS) command inquires about the status of one or more MQSeries channel instances.

This command cannot be used for client-connection channels.

You must specify the name of the channel for which you want to inquire status information. This can be a specific channel name or a generic channel name. By using a generic channel name, you can inquire either:

- Status information for all channels, or
- Status information for one or more channels that match the specified name.

You must also specify whether you want:

- The current status data (of current channels only), or
- The saved status data of all channels.

Before explaining the syntax and options for this command, it is necessary to describe the format of the status data that is available for channels and the states that channels may have.

There are two classes of data available for channel status. These are **saved** and **current**. The status fields available for saved data are a subset of the fields available for current data and are called **common** status fields. Note that although the common data *fields* are the same, the data *values* may be different for saved and current status. The rest of the fields available for current data are called **current-only** status fields.

- **Saved** data consists of the common status fields noted in the syntax diagram. This data is reset at the following times:
  - For all channels:
    - When the channel enters or leaves STOPPED or RETRY state
  - For a sending channel:
    - Before requesting confirmation that a batch of messages has been received
    - When confirmation has been received
  - For a receiving channel:
    - Just before confirming that a batch of messages has been received
  - For a server connection channel:
    - No data is saved

  Therefore, a channel which has never been current will not have any saved status.
- **Current** data consists of the common status fields and current-only status fields as noted in the syntax diagram. The data fields are continually updated as messages are sent or received.

This method of operation has the following consequences:

- An inactive channel may not have any saved status –if it has never been current or has not yet reached a point where saved status is reset.
- The “common” data fields may have different values for saved and current status.

- A current channel always has current status and may have saved status.

Channels may be current or inactive:

Current channels
These are channels that have been started, or on which a client has connected, and that have not finished or disconnected normally. They may not yet have reached the point of transferring messages, or data, or even of establishing contact with the partner. Current channels have **current** status and may also have **saved** status.

The term **Active** is used to describe the set of current channels which are not stopped.

Inactive channels
These are channels that have either not been started or on which a client has not connected, or that have finished or disconnected normally. (Note that if a channel is stopped, it is not yet considered to have finished normally – and is, therefore, still current.) Inactive channels have either **saved** status or no status at all.

There can be more than one instance of a receiver, requester or server-connection channel current at the same time (the requester is acting as a receiver). This occurs if several senders, at different queue managers, each initiate a session with this receiver, using the same channel name. For channels of other types, there can only be one instance current at any time.

For all channel types, however, there can be more than one set of saved status information available for a given channel name. At most one of these sets relates to a current instance of the channel, the rest relate to previously current instances. Multiple instances arise if different transmission queue names or connection names have been used in connection with the same channel. This can happen in the following cases:

- At a sender or server:
  - If the same channel has been connected to by different requesters (servers only),
  - If the transmission queue name has been changed in the definition, or
  - If the connection name has been changed in the definition.
- At a receiver or requester:
  - If the same channel has been connected to by different senders or servers, or
  - If the connection name has been changed in the definition (for requester channels initiating connection).

The number of sets returned for a given channel can be limited by using the *XmitQName*, *ConnectionName* and *ChannelInstanceType* parameters.

Required parameters: *ChannelName*

Optional parameters: *XmitQName*, *ConnectionName ChannelInstanceType*, *ChannelInstanceAttrs*

Required parameters

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

Generic channel names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all objects having names that start with the selected character string. An asterisk on its own matches all possible names.

The channel name is always returned, regardless of the instance attributes requested.

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Optional parameters

XmitQName (MQCFST)
Transmission queue name (parameter identifier: MQCACH_XMIT_Q_NAME).

If this parameter is present, eligible channel instances are limited to those using this transmission queue. If it is not specified, eligible channel instances are not limited in this way.

The transmission queue name is always returned, regardless of the instance attributes requested.

The maximum length of the string is MQ_Q_NAME_LENGTH.

ConnectionName (MQCFST)
Connection name (parameter identifier: MQCACH_CONNECTION_NAME).

If this parameter is present, eligible channel instances are limited to those using this connection name. If it is not specified, eligible channel instances are not limited in this way.

The connection name is always returned, regardless of the instance attributes requested.

If the *TransportType* has a value of MQXPT_TCP, the saved channel status omits any part number from the connection name. A connection name specified when requesting saved channel status should therefore never include a part number. It should only specify the TCP/IP address.

The maximum length of the string is MQ_CONN_NAME_LENGTH.

ChannelInstanceType (MQCFIN)
Channel instance type (parameter identifier: MQIACH_CHANNEL_INSTANCE_TYPE).

It is always returned regardless of the channel instance attributes requested.

The value may be:

MQOT_CURRENT_CHANNEL
Current channel status.

This is the default, and indicates that only current status information for active channels is to be returned.

Both common status information and active-only status information can be requested for current channels.

MQOT_SAVED_CHANNEL
Saved channel status.

Specify this to cause saved status information for both active and inactive channels to be returned.

Only common status information can be returned. Active-only status information is not returned for active channels if this keyword is specified.

The default value if this parameter is not specified is MQOT_CURRENT_CHANNEL.

ChannelInstanceAttrs (MQCFIL)
Channel instance attributes (parameter identifier: MQIACH_CHANNEL_INSTANCE_ATTRS).

If status information is requested which is not relevant for the particular channel type, this is not an error. Similarly, it is not an error to request status information that is applicable only to active channels for saved channel instances. In both of these cases, no structure is returned in the response for the information concerned.

For a saved channel instance, the MQCACH_CURRENT_LUWID, MQIACH_CURRENT_MSGS, and MQIACH_CURRENT_SEQ_NUMBER attributes have meaningful information only if the channel instance is in doubt. However, the attribute values are still returned when requested, even if the channel instance is not in-doubt.

The attribute list may specify the following on its own (this is the default value used if the parameter is not specified):

MQIACF_ALL
All attributes.

or a combination of the following:

Common status

The following information applies to all sets of channel status, whether or not the set is current.

MQCACH_CHANNEL_NAME
Channel name.

MQCACH_XMIT_Q_NAME
Transmission queue name.

MQCACH_CONNECTION_NAME
Connection name.

MQIACH_CHANNEL_INSTANCE_TYPE
Channel instance type.

MQCACH_CURRENT_LUWID
Logical unit of work identifier for current batch.

MQCACH_LAST_LUWID
Logical unit of work identifier for last committed batch.

MQIACH_CURRENT_MSGS
Number of messages sent or received in current batch.

MQIACH_CURRENT_SEQ_NUMBER
Sequence number of last message sent or received.

MQIACH_INDOUBT_STATUS
Whether the channel is currently in-doubt.

MQIACH_LAST_SEQ_NUMBER
Sequence number of last message in last committed batch.

MQCACH_CURRENT_LUWID, MQCACH_LAST_LUWID, MQIACH_CURRENT_MSGS, MQIACH_CURRENT_SEQ_NUMBER, MQIACH_INDOUBT_STATUS and MQIACH_LAST_SEQ_NUMBER do not apply to server-connection channels, and no values are returned. If specified on the command they are ignored.

Current-only status

The following information applies only to current channel instances. The information applies to all channel types, except where stated.

MQCACH_CHANNEL_START_DATE
Date channel was started.

MQCACH_CHANNEL_START_TIME
Time channel was started.

MQCACH_LAST_MSG_DATE
Date last message was sent, or MQI call was handled.

MQCACH_LAST_MSG_TIME
Time last message was sent, or MQI call was handled.

MQCACH_MCA_JOB_NAME
Name of MCA job.

MQIACH_BATCHES
Number of completed batches.

MQIACH_BUFFERS_SENT
Number of buffers sent.

MQIACH_BUFFERS_RCVD
Number of buffers received.

MQIACH_BYTES_SENT
Number of bytes sent.

MQIACH_BYTES_RCVD
Number of bytes received.

MQIACH_LONG_RETRIES_LEFT
Number of long retry attempts remaining.

MQIACH_MCA_STATUS
MCA status.

MQIACH_MSGS
Number of messages sent or received, or number of MQI calls handled.

MQIACH_SHORT_RETRIES_LEFT
Number of short retry attempts remaining.

MQIACH_STOP_REQUESTED
Whether user stop request has been received.

The following are supported on OpenVMS, OS/2, Tandem NSK, UNIX systems, and Windows NT:

MQIACH_BATCH_SIZE
Batch size.

MQIACH_HB_INTERVAL
Heartbeat interval (seconds).

The following is supported on OpenVMS, OS/2, Tandem NSK, UNIX systems, 32-bit Windows, and Windows NT:

MQIACH_NPM_SPEED
Speed of nonpersistent messages.

MQIACH_BATCHES, MQIACH_LONG_RETRIES_LEFT, MQIACH_SHORT_RETRIES_LEFT, MQIACH_BATCH_SIZE, MQIACH_HB_INTERVAL and MQIACH_NPM_SPEED do not apply to server-connection channels, and no values are returned. If specified on the command they are ignored.

Error codes

In addition to the values for any command shown on page 136, for this command the following may be returned in the response format header:

Reason (MQLONG)
The value may be:

MQRC_SELECTOR_ERROR
(2067, X'813') Attribute selector not valid.

MQRCCF_CFIL_COUNT_ERROR
Count of parameter values not valid.

MQRCCF_CFIL_DUPLICATE_VALUE
Duplicate parameter.

MQRCCF_CFIL_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIL_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFIN_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFIN_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIN_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_CHANNEL_NAME_ERROR
Channel name error.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_CHL_INST_TYPE_ERROR
Channel instance type not valid.

MQRCCF_CHL_STATUS_NOT_FOUND
Channel status not found.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

MQRCCF_XMIT_Q_NAME_ERROR
Transmission queue name error.

Inquire Channel Status (Response)

The response to the Inquire Channel Status (MQCMD_INQUIRE_CHANNEL_STATUS) command consists of the response header followed by

- The *ChannelName* structure,
- The *XmitQName* structure,
- The *ConnectionName* structure,
- The *ChannelInstanceType* structure,
- The *ChannelType* structure, and
- The *ChannelStatus* structure

which are followed by the requested combination of status attribute parameter structures. One such message is generated for each channel instance found which matches the criteria specified on the command.

Always returned:

ChannelName, *XmitQName*, *ConnectionName*, *ChannelInstanceType*, *ChannelType*, *ChannelStatus*

Returned if requested:

InDoubtStatus, *LastSequenceNumber*, *LastLUWID*, *CurrentMsgs*, *CurrentSequenceNumber*, *CurrentLUWID*, *LastMsgTime*, *LastMsgDate*, *Msgs*, *BytesSent*, *BytesReceived*, *Batches*, *ChannelStartTime*, *ChannelStartDate*, *BuffersSent*, *BuffersReceived*, *LongRetriesLeft*, *ShortRetriesLeft*, *MCAJobName*, *MCAStatus*, *StopRequested*, *BatchSize*, *HeartbeatInterval*, *NonPersistentMsgSpeed*

Response data

ChannelName (MQCFST)

Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

XmitQName (MQCFST)

Transmission queue name (parameter identifier: MQCACH_XMIT_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

ConnectionName (MQCFST)

Connection name (parameter identifier: MQCACH_CONNECTION_NAME).

The maximum length of the string is MQ_CONN_NAME_LENGTH.

ChannelInstanceType (MQCFIN)

Channel instance type (parameter identifier: MQIACH_CHANNEL_INSTANCE_TYPE).

The value may be:

MQOT_CURRENT_CHANNEL
: Current channel status.

MQOT_SAVED_CHANNEL
: Saved channel status.

ChannelType (MQCFIN)
Channel type (parameter identifier: MQIACH_CHANNEL_TYPE).

The value may be:

MQCHT_SENDER
Sender.

MQCHT_SERVER
Server.

MQCHT_RECEIVER
Receiver.

MQCHT_REQUESTER
Requester.

MQCHT_SVRCONN
Server-connection (for use by clients).

This value is not supported in the following environment: 32-bit Windows.

MQCHT_CLNTCONN
Client connection.

This value is not supported in the following environments: OS/400, 32-bit Windows.

ChannelStatus (MQCFIN)
Channel status (parameter identifier: MQIACH_CHANNEL_STATUS).

The value may be:

MQCHS_BINDING
Channel is negotiating with the partner.

MQCHS_STARTING
Channel is waiting to become active.

MQCHS_RUNNING
Channel is transferring or waiting for messages.

MQCHS_PAUSED
Channel is paused.

MQCHS_STOPPING
Channel is in process of stopping.

MQCHS_RETRYING
Channel is reattempting to establish connection.

MQCHS_STOPPED
Channel is stopped.

MQCHS_REQUESTING
Requester channel is requesting connection.

MQCHS_INITIALIZING
Channel is initializing.

InDoubtStatus (MQCFIN)
Whether the channel is currently in doubt (parameter identifier: MQIACH_INDOUBT_STATUS).

A sending channel is only in doubt while the sending Message Channel Agent is waiting for an acknowledgment that a batch of messages, which it has sent, has been successfully received. It is not in doubt at all other times, including the period during which messages are being sent, but before an acknowledgment has been requested.

A receiving channel is never in doubt.

The value may be:

MQCHIDS_NOT_INDOUBT
Channel is not in-doubt.

MQCHIDS_INDOUBT
Channel is in-doubt.

LastSequenceNumber (MQCFIN)
Sequence number of last message in last committed batch (parameter identifier: MQIACH_LAST_SEQ_NUMBER).

LastLUWID (MQCFST)
Logical unit of work identifier for last committed batch (parameter identifier: MQCACH_LAST_LUWID).

The maximum length is MQ_LUWID_LENGTH.

CurrentMsgs (MQCFIN)
Number of messages in-doubt (parameter identifier: MQIACH_CURRENT_MSGS).

For a sending channel, this is the number of messages that have been sent in the current batch. It is incremented as each message is sent, and when the channel becomes in-doubt it is the number of messages that are in-doubt.

For a receiving channel, it is the number of messages that have been received in the current batch. It is incremented as each message is received.

The value is reset to zero, for both sending and receiving channels, when the batch is committed.

CurrentSequenceNumber (MQCFIN)
Sequence number of last message in in-doubt batch (parameter identifier: MQIACH_CURRENT_SEQ_NUMBER).

For a sending channel, this is the message sequence number of the last message sent. It is updated as each message is sent, and when the channel becomes in-doubt it is the message sequence number of the last message in the in-doubt batch.

For a receiving channel, it is the message sequence number of the last message that was received. It is updated as each message is received.

CurrentLUWID (MQCFST)
Logical unit of work identifier for in-doubt batch (parameter identifier: MQCACH_CURRENT_LUWID).

The logical unit of work identifier associated with the current batch, for a sending or a receiving channel.

For a sending channel, when the channel is in-doubt it is the LUWID of the in-doubt batch.

It is updated with the LUWID of the next batch when this is known.

The maximum length is MQ_LUWID_LENGTH.

LastMsgTime (MQCFST)
Time last message was sent, or MQI call was handled (parameter identifier: MQCACH_LAST_MSG_TIME).

The maximum length of the string is MQ_CHANNEL_TIME_LENGTH.

LastMsgDate (MQCFST)
Date last message was sent, or MQI call was handled (parameter identifier: MQCACH_LAST_MSG_DATE).

The maximum length of the string is MQ_CHANNEL_DATE_LENGTH.

Msgs (MQCFIN)
Number of messages sent or received, or number of MQI calls handled (parameter identifier: MQIACH_MSGS).

BytesSent (MQCFIN)
Number of bytes sent (parameter identifier: MQIACH_BYTES_SENT).

BytesReceived (MQCFIN)
Number of bytes received (parameter identifier: MQIACH_BYTES_RCVD).

Batches (MQCFIN)
Number of completed batches (parameter identifier: MQIACH_BATCHES).

ChannelStartTime (MQCFST)
Time channel started (parameter identifier: MQCACH_CHANNEL_START_TIME).

The maximum length of the string is MQ_CHANNEL_TIME_LENGTH.

ChannelStartDate (MQCFST)
Date channel started (parameter identifier: MQCACH_CHANNEL_START_DATE).

The maximum length of the string is MQ_CHANNEL_DATE_LENGTH.

BuffersSent (MQCFIN)
Number of buffers sent (parameter identifier: MQIACH_BUFFERS_SENT).

BuffersReceived (MQCFIN)
Number of buffers received (parameter identifier: MQIACH_BUFFERS_RCVD).

LongRetriesLeft (MQCFIN)
Number of long retry attempts remaining (parameter identifier: MQIACH_LONG_RETRIES_LEFT).

ShortRetriesLeft (MQCFIN)
Number of short retry attempts remaining (parameter identifier: MQIACH_SHORT_RETRIES_LEFT).

MCAJobName (MQCFST)
Name of MCA job (parameter identifier: MQCACH_MCA_JOB_NAME).

The maximum length of the string is MQ_MCA_JOB_NAME_LENGTH.

MCAStatus (MQCFIN)
MCA status (parameter identifier: MQIACH_MCA_STATUS).

The value may be:

MQMCAS_STOPPED
Message channel agent stopped.

MQMCAS_RUNNING
Message channel agent running.

StopRequested (MQCFIN)
Whether user stop request is outstanding (parameter identifier: MQIACH_STOP_REQUESTED).

The value may be:

MQCHSR_STOP_NOT_REQUESTED
User stop request has not been received.

MQCHSR_STOP_REQUESTED
User stop request has been received.

BatchSize (MQCFIN)
Negotiated batch size (parameter identifier: MQIACH_BATCH_SIZE).

HeartbeatInterval (MQCFIN)
Heartbeat interval (parameter identifier: MQIACH_HB_INTERVAL).

NonPersistentMsgSpeed (MQCFIN)
Speed at which nonpersistent messages are to be sent (parameter identifier: MQIACH_NPM_SPEED).

The value may be:

MQNPMS_NORMAL
Normal speed.

MQNPMS_FAST
Fast speed.

Inquire Process

The Inquire Process (MQCMD_INQUIRE_PROCESS) command inquires about the attributes of existing MQSeries processes.

| *This PCF is not supported if you are using MQSeries for Windows Version 2.1.*

Required parameters: *ProcessName*

Optional parameters: *ProcessAttrs*

Required parameters

ProcessName (MQCFST)

Process name (parameter identifier: MQCA_PROCESS_NAME).

Generic process names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all processes having names that start with the selected character string. An asterisk on its own matches all possible names.

The process name is always returned regardless of the attributes requested.

The maximum length of the string is MQ_PROCESS_NAME_LENGTH.

Optional parameters

ProcessAttrs (MQCFIL)

Process attributes (parameter identifier: MQIACF_PROCESS_ATTRS).

The attribute list may specify the following on its own (this is the default value used if the parameter is not specified):

MQIACF_ALL
: All attributes.

or a combination of the following:

MQCA_PROCESS_NAME
: Name of process definition.

MQCA_PROCESS_DESC
: Description of process definition.

MQIA_APPL_TYPE
: Application type.

MQCA_APPL_ID
: Application identifier.

MQCA_ENV_DATA
: Environment data.

MQCA_USER_DATA
: User data.

Error codes

In addition to the values for any command shown on page 136, for this command the following may be returned in the response format header:

Reason (MQLONG)
The value may be:

MQRC_SELECTOR_ERROR
(2067, X'813') Attribute selector not valid.

MQRC_UNKNOWN_OBJECT_NAME
(2085, X'825') Unknown object name.

MQRCCF_CFIL_COUNT_ERROR
Count of parameter values not valid.

MQRCCF_CFIL_DUPLICATE_VALUE
Duplicate parameter.

MQRCCF_CFIL_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIL_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

Inquire Process (Response)

The response to the Inquire Process (MQCMD_INQUIRE_PROCESS) command consists of the response header followed by the *ProcessName* structure and the requested combination of attribute parameter structures. If a generic process name was specified, one such message is generated for each process found.

This response is not supported on 32-bit Windows.

Always returned:
ProcessName

Returned if requested:
ProcessDesc, *ApplType*, *ApplId*, *EnvData*, *UserData*

Response data

ProcessName (MQCFST)
The name of the process definition (parameter identifier: MQCA_PROCESS_NAME).

The maximum length of the string is MQ_PROCESS_NAME_LENGTH.

ProcessDesc (MQCFST)
Description of process definition (parameter identifier: MQCA_PROCESS_DESC).

The maximum length of the string is MQ_PROCESS_DESC_LENGTH.

ApplType (MQCFIN)
Application type (parameter identifier: MQIA_APPL_TYPE).

The value may be:

MQAT_OS400
OS/400 application.

MQAT_OS2
OS/2 or Presentation Manager application.

MQAT_DOS
DOS client application.

MQAT_WINDOWS
Windows client or 16-bit Windows application.

MQAT_WINDOWS_NT
Windows NT or 32-bit Windows application.

MQAT_UNIX
UNIX application.

MQAT_AIX
AIX application (same value as MQAT_UNIX).

MQAT_CICS
CICS transaction.

user-value: User-defined application type in the range 65 536 through 999 999 999.

ApplId (MQCFST)
Application identifier (parameter identifier: MQCA_APPL_ID).

The maximum length of the string is MQ_PROCESS_APPL_ID_LENGTH.

EnvData (MQCFST)
Environment data (parameter identifier: MQCA_ENV_DATA).

The maximum length of the string is MQ_PROCESS_ENV_DATA_LENGTH.

UserData (MQCFST)
User data (parameter identifier: MQCA_USER_DATA).

The maximum length of the string is MQ_PROCESS_USER_DATA_LENGTH.

Inquire Process Names

The Inquire Process Names (MQCMD_INQUIRE_PROCESS_NAMES) command inquires for a list of process names that match the generic process name specified.

| *This PCF is not supported if you are using MQSeries for Windows Version 2.1.*

Required parameters:

ProcessName

Optional parameters:

None

Required parameters

ProcessName (MQCFST)

Name of process-definition for queue (parameter identifier: MQCA_PROCESS_NAME).

Generic process names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all objects having names that start with the selected character string. An asterisk on its own matches all possible names.

Error codes

In addition to the values for any command shown on page 136, for this command the following may be returned in the response format header:

Reason (MQLONG)

The value may be:

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

Inquire Process Names (Response)

The response to the Inquire Process Names (MQCMD_INQUIRE_PROCESS_NAMES) command consists of the response header followed by a single parameter structure giving zero or more names that match the specified process name.

This response is not supported on 32-bit Windows.

Always returned:

ProcessNames

Returned if requested:

None

Response data

ProcessNames (MQCFSL)

Process Names (parameter identifier: MQCACF_PROCESS_NAMES).

Inquire Queue

The Inquire Queue (MQCMD_INQUIRE_Q) command inquires about the attributes of MQSeries queues.

Required parameters:

QName

Optional parameters:

QType, *QAttrs*

Required parameters

QName (MQCFST)

Queue name (parameter identifier: MQCA_Q_NAME).

Generic queue names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all queues having names that start with the selected character string. An asterisk on its own matches all possible names.

The queue name is always returned, regardless of the attributes requested.

The maximum length of the string is MQ_Q_NAME_LENGTH.

Optional parameters

QType (MQCFIN)

Queue type (parameter identifier: MQIA_Q_TYPE).

If this parameter is present, eligible queues are limited to those of the specified type. Any attribute selector specified in the *QAttrs* list which is only valid for queues of a different type or types is ignored; no error is raised.

If this parameter is not present (or if MQQT_ALL is specified), queues of all types are eligible. Each attribute specified must be a valid queue attribute selector (that is, it must be one of those in the following list), but it may not be applicable to all (or any) of the queues actually returned. Queue attribute selectors that are valid but not applicable to the queue are ignored, no error messages occur and no attribute is returned. The value may be:

MQQT_ALL
: All queue types.

MQQT_LOCAL
: Local queue.

MQQT_ALIAS
: Alias queue definition.

MQQT_REMOTE
: Local definition of a remote queue.

The following is supported on all platforms, but on OS/400 for receipt by MQSeries for AS/400 V4R2:

MQQT_MODEL
: Model queue definition.

The default value if this parameter is not specified is MQQT_ALL.

Note: If this parameter is present, it must occur immediately after the *QName* parameter.

QAttrs (MQCFIL)
Queue attributes (parameter identifier: MQIACF_Q_ATTRS).

The attribute list may specify the following on its own (this is the default value used if the parameter is not specified):

MQIACF_ALL
All attributes.

or a combination of the following:

Relevant for any QType:

MQCA_Q_NAME
Queue name.

MQIA_Q_TYPE
Queue type.

MQCA_Q_DESC
Queue description.

MQIA_INHIBIT_PUT
Whether put operations are allowed.

MQIA_DEF_PRIORITY
Default message priority.

MQIA_DEF_PERSISTENCE
Default message persistence.

Relevant for alias QType:

MQIA_INHIBIT_GET
Whether get operations are allowed.

MQCA_BASE_Q_NAME
Name of queue that alias resolves to.

MQIA_SCOPE
Queue definition scope.

Relevant for local QType:

MQIA_INHIBIT_GET
Whether get operations are allowed.

MQCA_PROCESS_NAME
Name of process definition.

MQIA_MAX_Q_DEPTH
Maximum number of messages allowed on queue.

MQIA_MAX_MSG_LENGTH
Maximum message length.

MQIA_BACKOUT_THRESHOLD
Backout threshold.

MQCA_BACKOUT_REQ_Q_NAME
Excessive backout requeue name.

MQIA_SHAREABILITY
Whether queue can be shared.

MQIA_DEF_INPUT_OPEN_OPTION
Default open-for-input option.

MQIA_HARDEN_GET_BACKOUT
Whether to harden backout count.

MQIA_MSG_DELIVERY_SEQUENCE
Whether message priority is relevant.

MQIA_RETENTION_INTERVAL
Queue retention interval.

MQIA_DEFINITION_TYPE
Queue definition type.

MQIA_USAGE
Usage.

MQIA_OPEN_INPUT_COUNT
Number of MQOPEN calls that have the queue open for input.

MQIA_OPEN_OUTPUT_COUNT
Number of MQOPEN calls that have the queue open for output.

MQIA_CURRENT_Q_DEPTH
Number of messages on queue.

MQCA_CREATION_DATE
Queue creation date.

MQCA_CREATION_TIME
Queue creation time.

MQCA_INITIATION_Q_NAME
Initiation queue name.

MQIA_TRIGGER_CONTROL
Trigger control.

MQIA_TRIGGER_TYPE
Trigger type.

MQIA_TRIGGER_MSG_PRIORITY
Threshold message priority for triggers.

MQIA_TRIGGER_DEPTH
Trigger depth.

MQCA_TRIGGER_DATA
Trigger data.

MQIA_SCOPE
Queue definition scope.

MQIA_Q_DEPTH_HIGH_LIMIT
High limit for queue depth.

MQIA_Q_DEPTH_LOW_LIMIT
Low limit for queue depth.

MQIA_Q_DEPTH_MAX_EVENT
Control attribute for queue depth max events.

MQIA_Q_DEPTH_HIGH_EVENT
Control attribute for queue depth high events.

MQIA_Q_DEPTH_LOW_EVENT
Control attribute for queue depth low events.

MQIA_Q_SERVICE_INTERVAL
Limit for queue service interval.

MQIA_Q_SERVICE_INTERVAL_EVENT
Control attribute for queue service interval events.

| The following is supported on AIX, HP-UX, OS/2, OS/400, Sun Solaris, and Windows NT:

MQIA_DIST_LISTS
Distribution list support.

Relevant for remote QType:

MQCA_REMOTE_Q_NAME
Name of remote queue as known locally on the remote queue manager.

MQCA_REMOTE_Q_MGR_NAME
Name of remote queue manager.

MQCA_XMIT_Q_NAME
Transmission queue name.

MQIA_SCOPE
Queue definition scope.

Relevant for model QType:

MQIA_INHIBIT_GET
Whether get operations are allowed.

MQCA_PROCESS_NAME
Name of process definition.

MQIA_MAX_Q_DEPTH
Maximum number of messages allowed on queue.

MQIA_MAX_MSG_LENGTH
Maximum message length.

MQIA_BACKOUT_THRESHOLD
Backout threshold.

MQCA_BACKOUT_REQ_Q_NAME
Excessive backout requeue name.

MQIA_SHAREABILITY
Whether queue can be shared.

MQIA_DEF_INPUT_OPEN_OPTION
Default open-for-input option.

MQIA_HARDEN_GET_BACKOUT
Whether to harden backout count.

MQIA_MSG_DELIVERY_SEQUENCE
Whether message priority is relevant.

MQIA_RETENTION_INTERVAL
Queue retention interval.

MQIA_DEFINITION_TYPE
Queue definition type.

MQIA_USAGE
Usage.

MQCA_CREATION_DATE
Queue creation date.

MQCA_CREATION_TIME
Queue creation time.

MQCA_INITIATION_Q_NAME
Initiation queue name.

MQIA_TRIGGER_CONTROL
Trigger control.

MQIA_TRIGGER_TYPE
Trigger type.

MQIA_TRIGGER_MSG_PRIORITY
Threshold message priority for triggers.

MQIA_TRIGGER_DEPTH
Trigger depth.

MQCA_TRIGGER_DATA
Trigger data.

MQIA_Q_DEPTH_HIGH_LIMIT
High limit for queue depth.

MQIA_Q_DEPTH_LOW_LIMIT
Low limit for queue depth.

MQIA_Q_DEPTH_MAX_EVENT
Control attribute for queue depth max events.

MQIA_Q_DEPTH_HIGH_EVENT
Control attribute for queue depth high events.

MQIA_Q_DEPTH_LOW_EVENT
Control attribute for queue depth low events.

MQIA_Q_SERVICE_INTERVAL
Limit for queue service interval.

MQIA_Q_SERVICE_INTERVAL_EVENT
Control attribute for queue service interval events.

| The following is supported on AIX, HP-UX, OS/2, OS/400, Sun Solaris, and Windows NT:

MQIA_DIST_LISTS
Distribution list support.

Error codes

In addition to the values for any command shown on page 136, for this command the following may be returned in the response format header:

Reason (MQLONG)
The value may be:

MQRC_SELECTOR_ERROR
(2067, X'813') Attribute selector not valid.

MQRC_UNKNOWN_OBJECT_NAME
(2085, X'825') Unknown object name.

MQRCCF_CFIL_COUNT_ERROR
Count of parameter values not valid.

MQRCCF_CFIL_DUPLICATE_VALUE
Duplicate parameter.

MQRCCF_CFIL_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIL_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFIN_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFIN_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIN_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_Q_TYPE_ERROR
Queue type not valid.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

Inquire Queue (Response)

The response to the Inquire Queue (MQCMD_INQUIRE_Q) command consists of the response header followed by the *QName* structure and the requested combination of attribute parameter structures. If a generic queue name was specified, one such message is generated for each queue found.

Always returned:

QName

Returned if requested:

QType, *QDesc*, *InhibitGet*, *InhibitPut*, *DefPriority*, *DefPersistence*, *ProcessName*, *MaxQDepth*, *MaxMsgLength*, *BackoutThreshold*, *BackoutRequeueName*, *Shareability*, *DefInputOpenOption*, *HardenGetBackout*, *MsgDeliverySequence*, *RetentionInterval*, *DefinitionType*, *DistLists*, *Usage*, *OpenInputCount*, *OpenOutputCount*, *CurrentQDepth*, *CreationDate*, *CreationTime*, *InitiationQName*, *TriggerControl*, *TriggerType*, *TriggerMsgPriority*, *TriggerDepth*, *TriggerData*, *BaseQName*, *RemoteQName*, *RemoteQMgrName*, *XmitQName*, *Scope*, *QDepthHighLimit*, *QDepthLowLimit*, *QDepthMaxEvent*, *QDepthHighEvent*, *QDepthLowEvent*, *QServiceInterval*, *QServiceIntervalEvent*

Response data

QName (MQCFST)

Queue name (parameter identifier: MQCA_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

QType (MQCFIN)

Queue type (parameter identifier: MQIA_Q_TYPE).

The value may be:

MQQT_ALIAS
: Alias queue definition.

MQQT_LOCAL
: Local queue.

MQQT_REMOTE
: Local definition of a remote queue.

MQQT_MODEL
: Model queue definition.

QDesc (MQCFST)

Queue description (parameter identifier: MQCA_Q_DESC).

The maximum length of the string is MQ_Q_DESC_LENGTH.

InhibitGet (MQCFIN)

Whether get operations are allowed (parameter identifier: MQIA_INHIBIT_GET).

The value may be:

MQQA_GET_ALLOWED
: Get operations are allowed.

MQQA_GET_INHIBITED
Get operations are inhibited.

InhibitPut (MQCFIN)
Whether put operations are allowed (parameter identifier: MQIA_INHIBIT_PUT).

The value may be:

MQQA_PUT_ALLOWED
Put operations are allowed.

MQQA_PUT_INHIBITED
Put operations are inhibited.

DefPriority (MQCFIN)
Default priority (parameter identifier: MQIA_DEF_PRIORITY).

DefPersistence (MQCFIN)
Default persistence (parameter identifier: MQIA_DEF_PERSISTENCE).

The value may be:

MQPER_PERSISTENT
Message is persistent.

MQPER_NOT_PERSISTENT
Message is not persistent.

ProcessName (MQCFST)
Name of process definition for queue (parameter identifier: MQCA_PROCESS_NAME).

The maximum length of the string is MQ_PROCESS_NAME_LENGTH.

MaxQDepth (MQCFIN)
Maximum queue depth (parameter identifier: MQIA_MAX_Q_DEPTH).

MaxMsgLength (MQCFIN)
Maximum message length (parameter identifier: MQIA_MAX_MSG_LENGTH).

BackoutThreshold (MQCFIN)
Backout threshold (parameter identifier: MQIA_BACKOUT_THRESHOLD).

BackoutRequeueName (MQCFST)
Excessive backout requeue name (parameter identifier: MQCA_BACKOUT_REQ_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

Shareability (MQCFIN)
Whether queue can be shared (parameter identifier: MQIA_SHAREABILITY).

The value may be:

MQQA_SHAREABLE
Queue is shareable.

MQQA_NOT_SHAREABLE
Queue is not shareable.

DefInputOpenOption (MQCFIN)
Default input open option for defining whether queues can be shared (parameter identifier: MQIA_DEF_INPUT_OPEN_OPTION).

The value may be:

MQOO_INPUT_EXCLUSIVE
Open queue to get messages with exclusive access.

MQOO_INPUT_SHARED
Open queue to get messages with shared access.

HardenGetBackout (MQCFIN)
Whether to harden backout (parameter identifier: MQIA_HARDEN_GET_BACKOUT).

The value may be:

MQQA_BACKOUT_HARDENED
Backout count remembered.

MQQA_BACKOUT_NOT_HARDENED
Backout count may not be remembered.

MsgDeliverySequence (MQCFIN)
Whether priority is relevant (parameter identifier: MQIA_MSG_DELIVERY_SEQUENCE).

The value may be:

MQMDS_PRIORITY
Messages are returned in priority order.

MQMDS_FIFO
Messages are returned in FIFO order (first in, first out).

RetentionInterval (MQCFIN)
Retention interval (parameter identifier: MQIA_RETENTION_INTERVAL).

DefinitionType (MQCFIN)
Queue definition type (parameter identifier: MQIA_DEFINITION_TYPE).

The value may be:

MQQDT_PREDEFINED
Predefined permanent queue.

MQQDT_PERMANENT_DYNAMIC
Dynamically defined permanent queue.

MQQDT_TEMPORARY_DYNAMIC
Dynamically defined temporary queue.

DistLists (MQCFIN)
Distribution list support (parameter identifier: MQIA_DIST_LISTS).

This parameter is supported in the following environments: AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

The value may be:

MQDL_SUPPORTED
Distribution lists supported.

MQDL_NOT_SUPPORTED
Distribution lists not supported.

Usage (MQCFIN)
Usage (parameter identifier: MQIA_USAGE).

The value may be:

MQUS_NORMAL
Normal usage.

MQUS_TRANSMISSION
Transmission queue.

OpenInputCount (MQCFIN)
Number of MQOPEN calls that have the queue open for input (parameter identifier: MQIA_OPEN_INPUT_COUNT).

OpenOutputCount (MQCFIN)
Number of MQOPEN calls that have the queue open for output (parameter identifier: MQIA_OPEN_OUTPUT_COUNT).

CurrentQDepth (MQCFIN)
Current queue depth (parameter identifier: MQIA_CURRENT_Q_DEPTH).

CreationDate (MQCFST)
Queue creation date (parameter identifier: MQCA_CREATION_DATE).

The maximum length of the string is MQ_CREATION_DATE_LENGTH.

CreationTime (MQCFST)
Creation time (parameter identifier: MQCA_CREATION_TIME).

The maximum length of the string is MQ_CREATION_TIME_LENGTH.

InitiationQName (MQCFST)
Initiation queue name (parameter identifier: MQCA_INITIATION_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

TriggerControl (MQCFIN)
Trigger control (parameter identifier: MQIA_TRIGGER_CONTROL).

The value may be:

MQTC_OFF
Trigger messages not required.

MQTC_ON
Trigger messages required.

TriggerType (MQCFIN)
Trigger type (parameter identifier: MQIA_TRIGGER_TYPE).

The value may be:

MQTT_NONE
No trigger messages.

MQTT_FIRST
Trigger message when queue depth goes from 0 to 1.

MQTT_EVERY
Trigger message for every message.

MQTT_DEPTH
Trigger message when depth threshold exceeded.

TriggerMsgPriority (MQCFIN)
Threshold message priority for triggers (parameter identifier: MQIA_TRIGGER_MSG_PRIORITY).

TriggerDepth (MQCFIN)
Trigger depth (parameter identifier: MQIA_TRIGGER_DEPTH).

TriggerData (MQCFST)
Trigger data (parameter identifier: MQCA_TRIGGER_DATA).

The maximum length of the string is MQ_TRIGGER_DATA_LENGTH.

BaseQName (MQCFST)
Queue name to which the alias resolves (parameter identifier: MQCA_BASE_Q_NAME).

This is the name of a queue that is defined to the local queue manager.

The maximum length of the string is MQ_Q_NAME_LENGTH.

RemoteQName (MQCFST)
Name of remote queue as known locally on the remote queue manager (parameter identifier: MQCA_REMOTE_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

RemoteQMgrName (MQCFST)
Name of remote queue manager (parameter identifier: MQCA_REMOTE_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

XmitQName (MQCFST)
Transmission queue name (parameter identifier: MQCA_XMIT_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

Scope (MQCFIN)
Scope of the queue definition (parameter identifier: MQIA_SCOPE).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

The value may be:

MQSCO_Q_MGR
Queue-manager scope.

MQSCO_CELL
Cell scope.

QDepthHighLimit (MQCFIN)
High limit for queue depth (parameter identifier: MQIA_Q_DEPTH_HIGH_LIMIT).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

The threshold against which the queue depth is compared to generate a Queue Depth High event.

QDepthLowLimit (MQCFIN)
Low limit for queue depth (parameter identifier: MQIA_Q_DEPTH_LOW_LIMIT).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

The threshold against which the queue depth is compared to generate a Queue Depth Low event.

QDepthMaxEvent (MQCFIN)
Controls whether Queue Full events are generated (parameter identifier: MQIA_Q_DEPTH_MAX_EVENT).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

The value may be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

QDepthHighEvent (MQCFIN)
Controls whether Queue Depth High events are generated (parameter identifier: MQIA_Q_DEPTH_HIGH_EVENT).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

The value may be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

QDepthLowEvent (MQCFIN)
Controls whether Queue Depth Low events are generated (parameter identifier: MQIA_Q_DEPTH_LOW_EVENT).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

The value may be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

QServiceInterval (MQCFIN)
Target for queue service interval (parameter identifier: MQIA_Q_SERVICE_INTERVAL).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

The service interval used for comparison to generate Queue Service Interval High and Queue Service Interval OK events.

QServiceIntervalEvent (MQCFIN)
Controls whether Service Interval High or Service Interval OK events are generated (parameter identifier: MQIA_Q_SERVICE_INTERVAL_EVENT).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

The value may be:

MQQSIE_HIGH
Queue Service Interval High events enabled.

MQQSIE_OK
Queue Service Interval OK events enabled.

MQQSIE_NONE
No queue service interval events enabled.

Inquire Queue Manager

The Inquire Queue Manager (MQCMD_INQUIRE_Q_MGR) command inquires about the attributes of a queue manager.

Required parameters:

None

Optional parameters:

QMgrAttrs

Optional parameters

QMgrAttrs (MQCFIL)

Queue manager attributes (parameter identifier: MQIACF_Q_MGR_ATTRS).

The attribute list may specify the following on its own (this is the default value used if the parameter is not specified):

MQIACF_ALL
: All attributes.

or a combination of the following:

MQCA_Q_MGR_NAME
: Name of local queue manager.

MQCA_Q_MGR_DESC
: Queue manager description.

MQIA_PLATFORM
: Platform on which the queue manager resides.

MQIA_COMMAND_LEVEL
: Command level supported by queue manager.

MQIA_TRIGGER_INTERVAL
: Trigger interval.

MQCA_DEAD_LETTER_Q_NAME
: Name of dead-letter queue.

MQIA_MAX_PRIORITY
: Maximum priority.

MQCA_COMMAND_INPUT_Q_NAME
: System command input queue name.

MQCA_DEF_XMIT_Q_NAME
: Default transmission queue name.

MQIA_CODED_CHAR_SET_ID
: Coded character set identifier.

MQIA_MAX_HANDLES
: Maximum number of handles.

MQIA_MAX_UNCOMMITTED_MSGS
: Maximum number of uncommitted messages within a unit of work.

MQIA_MAX_MSG_LENGTH
: Maximum message length.

MQIA_SYNCPOINT
Syncpoint availability.

MQIA_AUTHORITY_EVENT
Control attribute for authority events.

MQIA_INHIBIT_EVENT
Control attribute for inhibit events.

MQIA_LOCAL_EVENT
Control attribute for local events.

MQIA_REMOTE_EVENT
Control attribute for remote events.

MQIA_START_STOP_EVENT
Control attribute for start stop events.

MQIA_PERFORMANCE_EVENT
Control attribute for performance events.

| The following attributes are supported on AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT:

MQIA_DIST_LISTS
Distribution list support.

MQIA_CHANNEL_AUTO_DEF
Control attribute for automatic channel definition.

MQIA_CHANNEL_AUTO_DEF_EVENT
Control attribute for automatic channel definition events.

MQCA_CHANNEL_AUTO_DEF_EXIT
Automatic channel definition exit name.

Error codes

In addition to the values for any command shown on page 136, for this command the following may be returned in the response format header:

Reason (MQLONG)
The value may be:

MQRC_SELECTOR_ERROR
(2067, X'813') Attribute selector not valid.

MQRCCF_CFIL_COUNT_ERROR
Count of parameter values not valid.

MQRCCF_CFIL_DUPLICATE_VALUE
Duplicate parameter.

MQRCCF_CFIL_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIL_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

Inquire Queue Manager (Response)

The response to the Inquire Queue Manager (MQCMD_INQUIRE_Q_MGR) command consists of the response header followed by the *QMgrName* structure and the requested combination of attribute parameter structures.

Always returned:

QMgrName

Returned if requested:

QmgrDesc, *Platform*, *CommandLevel*, *TriggerInterval*, *DeadLetterQName*, *MaxPriority*, *CommandInputQName*, *DefXmitQName*, *CodedCharSetId*, *MaxHandles*, *MaxUncommittedMsgs*, *MaxMsgLength*, *DistLists*, *SyncPoint*, *AuthorityEvent*, *InhibitEvent*, *LocalEvent*, *RemoteEvent*, *StartStopEvent*, *PerformanceEvent*, *ChannelAutoDef*, *ChannelAutoDefEvent*, *ChannelAutoDefExit*

Response data

QMgrName (MQCFST)
Name of local queue manager (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

QmgrDesc (MQCFST)
Queue manager description (parameter identifier: MQCA_Q_MGR_DESC).

The maximum length of the string is MQ_Q_MGR_DESC_LENGTH.

Platform (MQCFIN)
Platform on which the queue manager resides (parameter identifier: MQIA_PLATFORM).

The value may be:

MQPL_OS400
OS/400.

MQPL_OS2
OS/2.

MQPL_UNIX
UNIX systems.

MQPL_AIX
AIX (same value as MQPL_UNIX).

MQPL_WINDOWS_NT
Windows NT or 32-bit Windows.

| MQPL_NSK
| Tandem NSK.

| MQPL_VMS
| OpenVMS.

CommandLevel (MQCFIN)
Command level supported by queue manager (parameter identifier: MQIA_COMMAND_LEVEL).

The value may be:

MQCMDL_LEVEL_1
Level 1 of system control commands.

This value is returned by the following:

- MQSeries for AIX version 2 release 2
- MQSeries for MVS/ESA:
  - version 1 release 1.1
  - version 1 release 1.2
  - version 1 release 1.3
- MQSeries for OS/2 version 2 release 0
- MQSeries for OS/400:
  - version 2 release 3
  - version 3 release 1
  - version 3 release 6
- MQSeries for Windows version 2 release 0.

MQCMDL_LEVEL_101
MQSeries for Windows version 2 release 0.1.

MQCMDL_LEVEL_110
MQSeries for Windows version 2 release 1.

MQCMDL_LEVEL_114
MQSeries for MVS/ESA version 1 release 1.4.

MQCMDL_LEVEL_120
MQSeries for MVS/ESA version 1 release 2.0.

MQCMDL_LEVEL_200
MQSeries for Windows NT version 2 release 0.

MQCMDL_LEVEL_201
MQSeries for OS/2 version 2 release 0.1.

MQCMDL_LEVEL_220
Level 220 of system control commands.

This value is returned by the following:

- MQSeries for AT&T GIS UNIX version 2 release 2.
- MQSeries for SINIX and DC/OSx version 2 release 2.
- MQSeries for Sun OS version 2 release 2.
- MQSeries for Tandem NonStop Kernel version 2 release 2.

MQCMDL_LEVEL_221
Level 221 of system control commands.

This value is returned by the following:

- MQSeries for AIX version 2 release 2.1.
- MQSeries for Digital OpenVMS version 2 release 2.

MQCMDL_LEVEL_320
MQSeries for OS/400 version 3 release 2, and version 3 release 7.

MQCMDL_LEVEL_420
MQSeries for AS/400 version 4 release 2.

MQCMDL_LEVEL_500
Level 500 of system control commands.

This value is returned by the following:

- MQSeries for AIX version 5 release 0
- MQSeries for HP-UX version 5 release 0
- MQSeries for OS/2 version 5 release 0
- MQSeries for Solaris version 5 release 0
- MQSeries for Windows NT version 5 release 0

The set of system control commands that corresponds to a particular value of the *CommandLevel* attribute varies according to the value of the *Platform* attribute; both must be used to decide which system control commands are supported.

TriggerInterval (MQCFIN)
Trigger interval (parameter identifier: MQIA_TRIGGER_INTERVAL).

Specifies the trigger time interval, expressed in milliseconds, for use only with queues where *TriggerType* has a value of MQTT_FIRST.

In this case trigger messages are normally only generated when a suitable message arrives on the queue, and the queue was previously empty. Under certain circumstances, however, an additional trigger message can be generated with MQTT_FIRST triggering, even if the queue was not empty. These additional trigger messages are not generated more often than every *TriggerInterval* milliseconds.

The value may be in the range 0 through 999 999 999.

DeadLetterQName (MQCFST)
Dead letter (undelivered message) queue name (parameter identifier: MQCA_DEAD_LETTER_Q_NAME).

Specifies the name of the local queue that is to be used for undelivered messages. Messages are put on this queue if they cannot be routed to their correct destination.

The maximum length of the string is MQ_Q_NAME_LENGTH.

MaxPriority (MQCFIN)
Maximum priority (parameter identifier: MQIA_MAX_PRIORITY).

The value may be in the range 0-9.

CommandInputQName (MQCFST)
Command input queue name (parameter identifier: MQCA_COMMAND_INPUT_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

DefXmitQName (MQCFST)
Default transmission queue name (parameter identifier: MQCA_DEF_XMIT_Q_NAME).

This is the name of the default transmission queue that is used for the transmission of messages to remote queue managers, if there is no other indication of which transmission queue to use.

The maximum length of the string is MQ_Q_NAME_LENGTH.

CodedCharSetId (MQCFIN)
Coded character set identifier (parameter identifier: MQIA_CODED_CHAR_SET_ID).

MaxHandles (MQCFIN)
Maximum number of handles (parameter identifier: MQIA_MAX_HANDLES).

Specifies the maximum number of handles that any one job can have open at the same time.

The value may be in the range 1 through 999 999 999.

MaxUncommittedMsgs (MQCFIN)
Maximum number of uncommitted messages within a unit of work (parameter identifier: MQIA_MAX_UNCOMMITTED_MSGS).

That is:

- The number of messages that can be retrieved, plus
- The number of messages that can be put on a queue, plus
- Any trigger messages generated within this unit of work

under any one syncpoint. This limit does not apply to messages that are retrieved or put outside syncpoint.

The value may be in the range 1 through 10 000.

MaxMsgLength (MQCFIN)
Maximum message length (parameter identifier: MQIA_MAX_MSG_LENGTH).

DistLists (MQCFIN)
Distribution list support (parameter identifier: MQIA_DIST_LISTS).

This parameter is supported in the following environments: AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

The value may be:

MQDL_SUPPORTED
Distribution lists supported.

MQDL_NOT_SUPPORTED
Distribution lists not supported.

SyncPoint (MQCFIN)
Syncpoint availability (parameter identifier: MQIA_SYNCPOINT).

The value may be:

MQSP_AVAILABLE
Units of work and syncpointing available.

MQSP_NOT_AVAILABLE
Units of work and syncpointing not available.

AuthorityEvent (MQCFIN)
Controls whether authorization (Not Authorized) events are generated (parameter identifier: MQIA_AUTHORITY_EVENT).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

The value may be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

InhibitEvent (MQCFIN)
Controls whether inhibit (Inhibit Get and Inhibit Put) events are generated (parameter identifier: MQIA_INHIBIT_EVENT).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

The value may be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

LocalEvent (MQCFIN)
Controls whether local error events are generated (parameter identifier: MQIA_LOCAL_EVENT).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

The value may be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

RemoteEvent (MQCFIN)
Controls whether remote error events are generated (parameter identifier: MQIA_REMOTE_EVENT).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

The value may be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

StartStopEvent (MQCFIN)
Controls whether start and stop events are generated (parameter identifier: MQIA_START_STOP_EVENT).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

The value may be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

PerformanceEvent (MQCFIN)
Controls whether performance-related events are generated (parameter identifier: MQIA_PERFORMANCE_EVENT).

On OS/400, this is valid for receipt by MQSeries for AS/400 V4R2

The value may be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

ChannelAutoDef (MQCFIN)
Controls whether receiver and server-connection channels can be auto-defined (parameter identifier: MQIA_CHANNEL_AUTO_DEF).

This parameter is supported in the following environments: AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

The value may be:

MQCHAD_DISABLED
Channel auto-definition disabled.

MQCHAD_ENABLED
Channel auto-definition enabled.

ChannelAutoDefEvent (MQCFIN)
Controls whether channel auto-definition events are generated (parameter identifier: MQIA_CHANNEL_AUTO_DEF_EVENT).

Only relevant if channel auto-definition is enabled (see *ChannelAutoDef*).

This parameter is supported in the following environments: AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

The value may be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

ChannelAutoDefExit (MQCFST)
Channel auto-definition exit name (parameter identifier: MQCA_CHANNEL_AUTO_DEF_EXIT).

If a nonblank name is defined, and channel auto-definiion is enabled (see *ChannelAutoDef*), this exit is invoked when an inbound request for an undefined channel is received.

The format of the name is the same as for the *SecurityExit* parameter described in “Change Channel” on page 139.

The maximum length of the string is MQ_EXIT_NAME_LENGTH.

This parameter is supported in the following environments: AIX, HP-UX, OS/2, OS/400, Sun Solaris, Windows NT.

Inquire Queue Names

The Inquire Queue Names (MQCMD_INQUIRE_Q_NAMES) command inquires a list of queue names that match the generic queue name, and the optional queue type specified.

Required parameters:

QName

Optional parameters:

QType

Required parameters

QName (MQCFST)

Queue name (parameter identifier: MQCA_Q_NAME).

Generic queue names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all objects having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length of the string is MQ_Q_NAME_LENGTH.

Optional parameters

QType (MQCFIN)

Queue type (parameter identifier: MQIA_Q_TYPE).

If present, this parameter limits the queue names returned to queues of the specified type. If this parameter is not present, queues of all types are eligible. The value may be:

MQQT_ALL
: All queue types.

MQQT_LOCAL
: Local queue.

MQQT_ALIAS
: Alias queue definition.

MQQT_REMOTE
: Local definition of a remote queue.

The following is supported on all platforms, but on OS/400 for receipt by MQSeries for AS/400 V4R2:

MQQT_MODEL
: Model queue definition.

The default value if this parameter is not specified is MQQT_ALL.

Error codes

In addition to the values for any command shown on page 136, for this command the following may be returned in the response format header:

Reason (MQLONG)

The value may be:

MQRCCF_CFIN_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFIN_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIN_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_Q_TYPE_ERROR
Queue type not valid.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

Inquire Queue Names (Response)

The response to the Inquire Queue Names (MQCMD_INQUIRE_Q_NAMES) command consists of the response header followed by a single parameter structure giving zero or more names that match the specified queue name.

Always returned:

QNames

Returned if requested:

None

Response data

QNames (MQCFSL)

Queue names (parameter identifier: MQCACF_Q_NAMES).

Ping Channel

The Ping Channel (MQCMD_PING_CHANNEL) command tests a channel by sending data as a special message to the remote message queue manager and checking that the data is returned. The data is generated by the local queue manager.

This command can only be used for channels with a *ChannelType* value of MQCHT_SENDER or MQCHT_SERVER. It is not valid if the channel is running; however it is valid if the channel is stopped or in retry mode.

| *This PCF is not supported if you are using MQSeries for Windows Version 2.1.*

Required parameters:
ChannelName

Optional parameters:
DataCount

Required parameters

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The name of the channel to be tested. The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Optional parameters

DataCount (MQCFIN)
Data count (parameter identifier: MQIACH_DATA_COUNT).

Specifies the length of the data.

Specify a value in the range 16 through 32 768. The default value is 64 bytes.

Error codes

In addition to the values for any command shown on page 136, for this command the following may be returned in the response format header:

Reason (MQLONG)
The value may be:

MQRCCF_ALLOCATE_FAILED
Allocation failed.

MQRCCF_BIND_FAILED
Bind failed.

MQRCCF_CCSID_ERROR
Coded character-set identifier error.

MQRCCF_CFIN_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFIN_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIN_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_CHANNEL_IN_USE
Channel in use.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_CHANNEL_TYPE_ERROR
Channel type not valid.

MQRCCF_CONFIGURATION_ERROR
Configuration error.

MQRCCF_CONNECTION_CLOSED
Connection closed.

MQRCCF_CONNECTION_REFUSED
Connection refused.

MQRCCF_DATA_TOO_LARGE
Data too large.

MQRCCF_ENTRY_ERROR
Invalid connection name.

MQRCCF_HOST_NOT_AVAILABLE
Remote system not available.

MQRCCF_NO_COMMS_MANAGER
Communications manager not available.

MQRCCF_NO_STORAGE
Not enough storage available.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_PING_DATA_COMPARE_ERROR
Ping Channel command failed.

MQRCCF_PING_DATA_COUNT_ERROR
Data count not valid.

MQRCCF_PING_ERROR
Ping error.

MQRCCF_RECEIVE_FAILED
Receive failed.

MQRCCF_RECEIVED_DATA_ERROR
Received data error.

MQRCCF_REMOTE_QM_TERMINATING
Remote queue manager terminating.

MQRCCF_REMOTE_QM_UNAVAILABLE
Remote queue manager not available.

MQRCCF_SEND_FAILED
Send failed.

MQRCCF_NO_STORAGE
Not enough storage available.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

MQRCCF_TERMINATED_BY_SEC_EXIT
Channel terminated by security exit.

MQRCCF_UNKNOWN_REMOTE_CHANNEL
Remote channel not known.

MQRCCF_USER_EXIT_NOT_AVAILABLE
User exit not available.

Ping Queue Manager

The Ping Queue Manager (MQCMD_PING_Q_MGR) command tests whether the queue manager and its command server is responsive to commands. If the queue manager is responding a positive reply is returned.

Required parameters:
None

Optional parameters:
None

Error codes

In addition to the values for any command shown on page 136, for this command the following may be returned in the response format header:

Reason (MQLONG)
The value may be:

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

Reset Channel

The Reset Channel (MQCMD_RESET_CHANNEL) command resets the message sequence number for an MQSeries channel with, optionally, a specifed sequence number to be used the next time that the channel is started.

This command can be issued to a channel of any type except (MQCHT_SVRCONN and MQCHT_CLNTCONN). However, if it is issued to a sender (MQCHT_SENDER) or server (MQCHT_SERVER) channel, then in addition to resetting the value at the end at which the command is issued, the value at the other (receiver or requester) end will also be reset to the same value, when this channel is next initiated (and resynchronized if necessary).

If the command is issued to a receiver (MQCHT_RECEIVER) or requester (MQCHT_REQUESTER) channel, the value at the other end is *not* reset as well; this must be done separately if necessary.

Required parameters:

ChannelName

Optional parameters:

MsgSeqNumber

Required parameters

ChannelName (MQCFST)

Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The name of the channel to be reset. The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Optional parameters

MsgSeqNumber (MQCFIN)

Message sequence number (parameter identifier: MQIACH_MSG_SEQUENCE_NUMBER).

Specifies the new message sequence number.

The value may be in the range 1-999 999 999. The default value is one.

Error codes

In addition to the values for any command shown on page 136, for this command the following may be returned in the response format header:

Reason (MQLONG)

The value may be:

MQRCCF_CFIN_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFIN_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIN_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_MSG_SEQ_NUMBER_ERROR
Message sequence number not valid.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

Reset Queue Statistics

The Reset Queue Statistics (MQCMD_RESET_Q_STATS) command reports the performance data for a queue and then resets the performance data.

| *This PCF is not supported if you are using MQSeries for Tandem NSK version 2.2.*

Performance data is maintained for each local queue (including transmission queues). It is reset at the following times:

- When a Reset Queue Statistics command is issued
- When the queue manager is restarted

Required parameters:

QName

Optional parameters:

None

Required parameters

QName (MQCFST)

Queue name (parameter identifier: MQCA_Q_NAME).

The name of the local queue to be tested and reset.

Generic queue names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all objects having names that start with the selected character string. An asterisk on its own matches all possible names.

The maximum length of the string is MQ_Q_NAME_LENGTH.

Error codes

In addition to the values for any command shown on page 136, for this command the following may be returned in the response format header:

Reason (MQLONG)

The value may be:

MQRC_UNKNOWN_OBJECT_NAME
(2085, X'825') Unknown object name.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_Q_WRONG_TYPE
Action not valid for the queue of specified type.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

Reset Queue Statistics (Response)

The response to the Reset Queue Statistics (MQCMD_RESET_Q_STATS) command consists of the response header followed by the *QName* structure and the attribute parameter structures shown below. If a generic queue name was specified, one such message is generated for each queue found.

Always returned:

QName, *TimeSinceReset*, *HighQDepth*, *MsgEnqCount*, *MsgDeqCount*

Response data

QName (MQCFST)

Queue name (parameter identifier: MQCA_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

TimeSinceReset (MQCFIN)

Time since statistics reset in seconds (parameter identifier: MQIA_TIME_SINCE_RESET).

HighQDepth (MQCFIN)

Maximum number of messages on a queue (parameter identifier: MQIA_HIGH_Q_DEPTH).

This count is the peak value of the *CurrentQDepth* local queue attribute since the last reset. The *CurrentQDepth* is incremented during an MQPUT call, and during backout of an MQGET call, and is decremented during a (nonbrowse) MQGET call, and during backout of an MQPUT call.

MsgEnqCount (MQCFIN)

Number of messages enqueued (parameter identifier: MQIA_MSG_ENQ_COUNT).

This count includes messages that have been put to the queue, but have not yet been committed. The count is not decremented if the put is subsequently backed out.

MsgDeqCount (MQCFIN)

Number of messages dequeued (parameter identifier: MQIA_MSG_DEQ_COUNT).

This count includes messages that have been successfully retrieved (with a nonbrowse MQGET) from the queue, even though the MQGET has not yet been committed. The count is not decremented if the MQGET is subsequently backed out.

Resolve Channel

The Resolve Channel (MQCMD_RESOLVE_CHANNEL) command requests a channel to commit or back out in-doubt messages.

This command is used when the other end of a link fails during the confirmation stage, and for some reason it is not possible to reestablish the connection. In this situation the sending end remains in an in-doubt state, as to whether or not the messages were received. Any outstanding units of work must be resolved using Resolve Channel with either backout or commit.

Care must be exercised in the use of this command. If the resolution specified is not the same as the resolution at the receiving end, messages can be lost or duplicated.

This command can only be used for channels with a *ChannelType* value of MQCHT_SENDER or MQCHT_SERVER.

Required parameters:
ChannelName, *InDoubt*

Optional parameters:
None

Required parameters

ChannelName (MQCFST)
Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The name of the channel to be resolved. The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Optional parameters

InDoubt (MQCFIN)
Indoubt resolution (parameter identifier: MQIACH_IN_DOUBT).

Specifies whether to commit or back out the in-doubt messages.

The value may be:

MQIDO_COMMIT
Commit.

MQIDO_BACKOUT
Backout.

Error codes

In addition to the values for any command shown on page 136, for this command the following may be returned in the response format header:

Reason (MQLONG)
The value may be:

MQRCCF_CFIN_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFIN_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIN_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_INDOUBT_VALUE_ERROR
In-doubt value not valid.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

MQRCCF_PARM_SEQUENCE_ERROR
Parameter sequence not valid.

Start Channel

The Start Channel (MQCMD_START_CHANNEL) command starts an MQSeries channel.

This command can be issued to a channel of any type (except MQCHT_CLNTCONN). If, however, it is issued to a channel with a *ChannelType* value of MQCHT_RECEIVER or MQCHT_SVRCONN, the only action is to enable the channel, not start it.

Required parameters:

ChannelName

Optional parameters:

None

Required parameters

ChannelName (MQCFST)

Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The name of the channel to be started. The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Error codes

In addition to the values for any command shown on page 136, for this command the following may be returned in the response format header:

Reason (MQLONG)

The value may be:

MQRCCF_CFST_DUPLICATE_PARM
: Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
: Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
: Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
: String length not valid.

MQRCCF_CHANNEL_INDOUBT
: Channel in-doubt.

MQRCCF_CHANNEL_IN_USE
: Channel in use.

MQRCCF_CHANNEL_NOT_FOUND
: Channel not found.

MQRCCF_CHANNEL_TYPE_ERROR
: Channel type not valid.

MQRCCF_MQCONN_FAILED
: MQCONN call failed.

MQRCCF_MQINQ_FAILED
: MQINQ call failed.

MQRCCF_MQOPEN_FAILED
MQOPEN call failed.

MQRCCF_NOT_XMIT_Q
Queue is not a transmission queue.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

Start Channel Initiator

The Start Channel Initiator (MQCMD_START_CHANNEL_INIT) command starts an MQSeries channel initiator.

| *This PCF is not supported if you are using MQSeries for Windows Version 2.1.*

Required parameters:
InitiationQName

Optional parameters:
None

Required parameters

InitiationQName (MQCFST)
Initiation queue name (parameter identifier: MQCA_INITIATION_Q_NAME).

The name of the initiation queue for the channel initiation process. That is, the initiation queue that is specified in the definition of the transmission queue.

The maximum length of the string is MQ_Q_NAME_LENGTH.

Error codes

In addition to the values for any command shown on page 136, for this command the following may be returned in the response format header:

Reason (MQLONG)
The value may be:

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_MQCONN_FAILED
MQCONN call failed.

MQRCCF_MQGET_FAILED
MQGET call failed.

MQRCCF_MQOPEN_FAILED
MQOPEN call failed.

MQRCCF_OBJECT_NAME_ERROR
Object name not valid.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

Start Channel Listener

The Start Channel Listener (MQCMD_START_CHANNEL_LISTENER) command starts an MQSeries TCP/IP listener.

| *This PCF is supported only if you are using MQSeries for AS/400 V4R2,*
| *MQSeries for OS/2 Warp V5, or MQSeries for Windows NT V5,*

This command is valid only for TCP/IP transmission protocols.

Required parameters:
None

Optional parameters:
None

Error codes

In addition to the values for any command shown on page 136, for this command the following may be returned in the response format header:

Reason (MQLONG)
The value may be:

MQRCCF_LISTENER_NOT_STARTED
Listener not started.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

Stop Channel

The Stop Channel (MQCMD_STOP_CHANNEL) command stops an MQSeries channel.

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

Required parameters:

ChannelName

Optional parameters:

Quiesce

Required parameters

ChannelName (MQCFST)

Channel name (parameter identifier: MQCACH_CHANNEL_NAME).

The name of the channel to be stopped. The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

Optional parameters

Quiesce (MQCFIN)

Quiesce channel (parameter identifier: MQIACF_QUIESCE).

Specifies whether the channel should be quiesced or stopped immediately. If this parameter is not present the channel **is** quiesced. The value may be:

MQQO_YES
: Quiesce the channel.

MQQO_NO
: Do not quiesce the channel.

Error codes

In addition to the values for any command shown on page 136, for this command the following may be returned in the response format header:

Reason (MQLONG)

The value may be:

MQRCCF_CFIN_DUPLICATE_PARM
: Duplicate parameter.

MQRCCF_CFIN_LENGTH_ERROR
: Structure length not valid.

MQRCCF_CFIN_PARM_ID_ERROR
: Parameter identifier is not valid.

MQRCCF_CFST_DUPLICATE_PARM
: Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
: Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_CHANNEL_DISABLED
Channel disabled.

MQRCCF_CHANNEL_NOT_ACTIVE
Channel not active.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

MQRCCF_MQCONN_FAILED
MQCONN call failed.

MQRCCF_MQOPEN_FAILED
MQOPEN call failed.

MQRCCF_MQSET_FAILED
MQSET call failed.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

MQRCCF_QUIESCE_VALUE_ERROR
Quiesce value not valid.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

Chapter 9. Structures used for commands and responses

Commands, responses, and events are of the form:

- PCF header (MQCFH) structure, (described on page 334) followed by
- Zero or more parameter structures. Each of these is one of the following:
  - PCF integer parameter (MQCFIN, page 339)
  - PCF string parameter (MQCFST, page 341)
  - PCF integer list parameter (MQCFIL, page 345)
  - PCF string list parameter (MQCFSL, page 347)

This chapter defines these parameter structures.

How the structures are shown

The structures are described in a language-independent form. The declarations are shown in the following programming languages:

- C
- COBOL
- PL/I
- S/390 assembler

Data types

For each field of the structure the data type is given in brackets after the field name. These are the elementary data types described in the *MQSeries Application Programming Reference*.

Initial values and default structures

The *initial value* of each field is shown under its description. This is the value of the field in the *default structure*.

The default structures are supplied in the following header files:

C	CMQCFC	**PL/I**	CMQCFP
COBOL	CMQCFV	**Assembler**	CMQCFA
	CMQCFHL		CMQCFINA
	CMQCFHV		CMQCFILA
	CMQCFINL		CMQCFSTA
	CMQCFINV		CMQCFSLA
	CMQCFSLL		CMQCFHA
	CMQCFSLV		
	CMQCFSTL		
	CMQCFSTV		
	CMQCFILL		
	CMQCFILV		

Usage notes

If all of the strings in a PCF message have the same coded character-set identifier, the *CodedCharSetId* field in the message descriptor MQMD should be set to that identifier when the message is put, and the *CodedCharSetId* fields in the MQCFST and MQCFSL structures within the message should be set to MQCCSI_DEFAULT.

If some of the strings in the message have different character-set identifiers, the *CodedCharSetId* field in MQMD should be set to MQCCSI_EMBEDDED when the message is put, and the *CodedCharSetId* fields in the MQCFST and MQCFSL structures within the message should be set to the identifiers that apply.

Do not specify MQCCSI_EMBEDDED in MQMD when the message is put, with MQCCSI_DEFAULT in the MQCFST or MQCFSL structures within the message, as this will prevent conversion of the message.

Note: Only single-byte character sets (SBCS) should be used for the strings in the message. If a double-byte character set (DBCS) is specified, it will prevent conversion of the message.

MQCFH – PCF header

The MQCFH structure describes the information that is present at the start of the message data of a command message, or a response to a command message. In either case, the message descriptor *Format* field is MQFMT_ADMIN.

The PCF structures are also used for event messages. In this case the message descriptor *Format* field is MQFMT_EVENT.

The PCF structures can also be used for user-defined message data. In this case the message descriptor *Format* field is MQFMT_PCF (see “Message descriptor for a PCF command” on page 127). Also in this case, not all of the fields in the structure are meaningful. The supplied initial values can be used for most fields, but the application must set the *StrucLength* and *ParameterCount* fields to the values appropriate to the data.

Type (MQLONG)
Structure type.

This indicates the content of the message. The following are valid:

MQCFT_COMMAND
Message is a command.

MQCFT_RESPONSE
Message is a response to a command.

MQCFT_EVENT
Message is reporting an event.

The initial value of this field is MQCFT_COMMAND.

StrucLength (MQLONG)
Structure length.

This is the length in bytes of the MQCFH structure. The value must be:

MQCFH_STRUC_LENGTH
Length of command format header structure.

The initial value of this field is MQCFH_STRUC_LENGTH.

Version (MQLONG)
Structure version number.

The value must be:

MQCFH_VERSION_1
Version number for command format header structure.

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

MQCFH_CURRENT_VERSION
Current version of command format header structure.

The initial value of this field is MQCFH_VERSION_1.

Command (MQLONG)
Command identifier.

For a command message, this identifies the function to be performed. For a response message, it identifies the command to which this is the reply. The following are valid:

MQCMD_CHANGE_Q_MGR
Change queue manager.

MQCMD_INQUIRE_Q_MGR
Inquire queue manager.

MQCMD_CHANGE_PROCESS
Change process.

MQCMD_COPY_PROCESS
Copy process.

MQCMD_CREATE_PROCESS
Create process.

MQCMD_DELETE_PROCESS
Delete process.

MQCMD_INQUIRE_PROCESS
Inquire process.

MQCMD_CHANGE_Q
Change queue.

MQCMD_CLEAR_Q
Clear queue.

MQCMD_COPY_Q
Copy queue.

MQCMD_CREATE_Q
Create queue.

MQCMD_DELETE_Q
Delete queue.

MQCMD_INQUIRE_Q
Inquire queue.

MQCMD_RESET_Q_STATS
Reset queue statistics.

MQCMD_INQUIRE_Q_NAMES
Inquire queue names.

MQCMD_INQUIRE_PROCESS_NAMES
Inquire process-definition names.

MQCMD_INQUIRE_CHANNEL_NAMES
Inquire channel names.

MQCMD_CHANGE_CHANNEL
Change channel.

MQCMD_COPY_CHANNEL
Copy channel.

MQCMD_CREATE_CHANNEL
Create channel.

MQCMD_DELETE_CHANNEL
Delete channel.

MQCMD_INQUIRE_CHANNEL
Inquire channel.

MQCMD_PING_CHANNEL
Ping channel.

MQCMD_RESET_CHANNEL
Reset channel.

MQCMD_START_CHANNEL
Start channel.

MQCMD_STOP_CHANNEL
Stop channel.

MQCMD_START_CHANNEL_INIT
Start channel initiator.

MQCMD_START_CHANNEL_LISTENER
Start channel listener.

MQCMD_ESCAPE
Escape.

MQCMD_RESOLVE_CHANNEL
Resolve channel.

MQCMD_PING_Q_MGR
Ping queue manager.

MQCMD_INQUIRE_CHANNEL_STATUS
Inquire channel status.

MQCMD_Q_MGR_EVENT
Queue manager event.

MQCMD_PERFM_EVENT
Performance event.

MQCMD_CHANNEL_EVENT
Channel event.

The initial value of this field is 0.

MsgSeqNumber (MQLONG)
Message sequence number.

This is the sequence number of the message within a group of related messages. For a command, this field must have the value one (because a command is always contained within a single message). For a response, the field has the value one for the first (or only) response to a command, and increases by one for each successive response to that command.

The last (or only) message in a group has the MQCFC_LAST flag set in the *Control* field.

The initial value of this field is 1.

Control (MQLONG)
Control options.

The following are valid:

MQCFC_LAST
Last message in the group.

For a command, this value must always be set.

MQCFC_NOT_LAST
Not the last message in the group.

The initial value of this field is MQCFC_LAST.

CompCode (MQLONG)
Completion code.

This field is meaningful only for a response; its value is not significant for a command. The following are possible:

MQCC_OK
Command completed successfully.

MQCC_WARNING
Command completed with warning.

MQCC_FAILED
Command failed.

MQCC_UNKNOWN
Whether command succeeded is not known.

The initial value of this field is MQCC_OK.

Reason (MQLONG)
Reason code qualifying completion code.

This field is meaningful only for a response; its value is not significant for a command.

The possible reason codes that could be returned in response to a command are listed at the end of each command format definition in

Chapter 8, “Definitions of the Programmable Command Formats” on page 135. The reason codes are listed in alphabetic order, with complete descriptions in Appendix A, “Error codes” on page 455.

The initial value of this field is MQRC_NONE.

ParameterCount (MQLONG)
Count of parameter structures.

This is the number of parameter structures (MQCFIL, MQCFIN, MQCFSL, and MQCFST) that follow the MQCFH structure. The value of this field is zero or greater.

The initial value of this field is 0.

Table 18. Initial values of fields in MQCFH

Field name	Name of constant	Value of constant
Type	MQCFT_COMMAND	1
StrucLength	MQCFH_STRUC_LENGTH	36
Version	MQCFH_VERSION_1	1
Command	None	0
MsgSeqNumber	None	1
Control	MQCFC_LAST	1
CompCode	MQCC_OK	0
Reason	MQRC_NONE	0
ParameterCount	None	0

Notes:

1. In the C programming language, the macro variable MQCFH_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:

   ```
   MQCFH MyCFH = {MQCFH_DEFAULT};
   ```

C language declaration

```
typedef struct tagMQCFH {
  MQLONG  Type;            /* Structure type */
  MQLONG  StrucLength;     /* Structure length */
  MQLONG  Version;         /* Structure version number */
  MQLONG  Command;         /* Command identifier */
  MQLONG  MsgSeqNumber;    /* Message sequence number */
  MQLONG  Control;         /* Control options */
  MQLONG  CompCode;        /* Completion code */
  MQLONG  Reason;          /* Reason code qualifying completion code */
  MQLONG  ParameterCount;  /* Count of parameter structures */
 } MQCFH;
```

COBOL language declaration

```
**   MQCFH structure
  10 MQCFH.
**    Structure type
   15 MQCFH-TYPE           PIC S9(9) BINARY.
**    Structure length
```

```
   15 MQCFH-STRUCLENGTH    PIC S9(9) BINARY.
**    Structure version number
   15 MQCFH-VERSION        PIC S9(9) BINARY.
**    Command identifier
   15 MQCFH-COMMAND        PIC S9(9) BINARY.
**    Message sequence number
   15 MQCFH-MSGSEQNUMBER   PIC S9(9) BINARY.
**    Control options
   15 MQCFH-CONTROL        PIC S9(9) BINARY.
**    Completion code
   15 MQCFH-COMPCODE       PIC S9(9) BINARY.
**    Reason code qualifying completion code
   15 MQCFH-REASON         PIC S9(9) BINARY.
**    Count of parameter structures
   15 MQCFH-PARAMETERCOUNT PIC S9(9) BINARY.
```

PL/I language declaration (AIX, MVS/ESA, OS/2, and Windows NT)

```
dcl
 1 MQCFH based,
  3 Type           fixed bin(31), /* Structure type */
  3 StrucLength    fixed bin(31), /* Structure length */
  3 Version        fixed bin(31), /* Structure version number */
  3 Command        fixed bin(31), /* Command identifier */
  3 MsgSeqNumber   fixed bin(31), /* Message sequence number */
  3 Control        fixed bin(31), /* Control options */
  3 CompCode       fixed bin(31), /* Completion code */
  3 Reason         fixed bin(31), /* Reason code qualifying completion
                                     code */
  3 ParameterCount fixed bin(31); /* Count of parameter structures */
```

System/390 assembler-language declaration (MVS/ESA only)

```
MQCFH                       DSECT
MQCFH_TYPE                  DS   F        Structure type
MQCFH_STRUCLENGTH           DS   F        Structure length
MQCFH_VERSION               DS   F        Structure version number
MQCFH_COMMAND               DS   F        Command identifier
MQCFH_MSGSEQNUMBER          DS   F        Message sequence number
MQCFH_CONTROL               DS   F        Control options
MQCFH_COMPCODE              DS   F        Completion code
MQCFH_REASON                DS   F        Reason code qualifying
*                                         completion code
MQCFH_PARAMETERCOUNT        DS   F        Count of parameter
*                                         structures
MQCFH_LENGTH                EQU  *-MQCFH  Length of structure
                            ORG  MQCFH
MQCFH_AREA                  DS   CL(MQCFH_LENGTH)
```

MQCFIN – PCF integer parameter

The MQCFIN structure describes an integer parameter in a message that is a command or a response to a command. In either case, the format name in the message descriptor is MQFMT_ADMIN.

The MQCFIN structure is also used for event messages. In this case the message descriptor *Format* field is MQFMT_EVENT.

The MQCFIN structure can also be used for user-defined message data. In this case the message descriptor *Format* field is MQFMT_PCF (see "Message descriptor for a PCF command" on page 127). Also in this case, not all of the fields in the structure are meaningful. The supplied initial values can be used for most fields, but the application must set the *Value* field to the value appropriate to the data.

Type (MQLONG)
Structure type.

This indicates that the structure is a MQCFIN structure describing an integer parameter. The value must be:

MQCFT_INTEGER
Structure defining an integer.

The initial value of this field is MQCFT_INTEGER.

StrucLength (MQLONG)
Structure length.

This is the length in bytes of the MQCFIN structure. The value must be:

MQCFIN_STRUC_LENGTH
Length of command format integer-parameter structure.

The initial value of this field is MQCFIN_STRUC_LENGTH.

Parameter (MQLONG)
Parameter identifier.

This identifies the parameter whose value is contained in the structure. The values that can occur in this field depend on the value of the *Command* field in the MQCFH structure; see page 334 for details.

The initial value of this field is 0.

Value (MQLONG)
Parameter value.

This is the value of the parameter identified by the *Parameter* field.

The initial value of this field is 0.

Table 19. Initial values of fields in MQCFIN

Field name	Name of constant	Value of constant
Type	MQCFT_INTEGER	3
StrucLength	MQCFIN_STRUC_LENGTH	16
Parameter	None	0
Value	None	0

Notes:

1. In the C programming language, the macro variable MQCFIN_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:

```
MQCFIN MyCFIN = {MQCFIN_DEFAULT};
```

C language declaration

```
typedef struct tagMQCFIN {
  MQLONG  Type;         /* Structure type */
  MQLONG  StrucLength;  /* Structure length */
  MQLONG  Parameter;    /* Parameter identifier */
  MQLONG  Value;        /* Parameter value */
 } MQCFIN;
```

COBOL language declaration

```
**   MQCFIN structure
  10 MQCFIN.
**    Structure type
   15 MQCFIN-TYPE        PIC S9(9) BINARY.
**    Structure length
   15 MQCFIN-STRUCLENGTH PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFIN-PARAMETER   PIC S9(9) BINARY.
**    Parameter value
   15 MQCFIN-VALUE       PIC S9(9) BINARY.
```

PL/I language declaration (AIX, MVS/ESA, OS/2, and Windows NT)

```
dcl
 1 MQCFIN based,
  3 Type        fixed bin(31), /* Structure type */
  3 StrucLength fixed bin(31), /* Structure length */
  3 Parameter   fixed bin(31), /* Parameter identifier */
  3 Value       fixed bin(31); /* Parameter value */
```

System/390 assembler-language declaration (MVS/ESA only)

```
MQCFIN                    DSECT
MQCFIN_TYPE               DS   F        Structure type
MQCFIN_STRUCLENGTH        DS   F        Structure length
MQCFIN_PARAMETER          DS   F        Parameter identifier
MQCFIN_VALUE              DS   F        Parameter value
MQCFIN_LENGTH             EQU  *-MQCFIN Length of structure
                          ORG  MQCFIN
MQCFIN_AREA               DS   CL(MQCFIN_LENGTH)
```

MQCFST – PCF string parameter

The MQCFST structure describes a string parameter in a message that is a command or a response to a command. In either case, the format name in the message descriptor is MQFMT_ADMIN.

The MQCFST structure is also used for event messages. In this case the message descriptor *Format* field is MQFMT_EVENT.

The MQCFST structure can also be used for user-defined message data. In this case the message descriptor *Format* field is MQFMT_PCF (see “Message descriptor for a PCF command” on page 127). Also in this case, not all of the fields in the structure are meaningful. The supplied initial values can be used for most fields, but the application must set the *StrucLength*, *StringLength*, and *String* fields to the values appropriate to the data.

The structure ends with a variable-length character string; see the *String* field below for further details.

See “Usage notes” on page 334 for further information on how the structure should be used.

Type (MQLONG)
Structure type.

This indicates that the structure is an MQCFST structure describing a string parameter. The value must be:

MQCFT_STRING
Structure defining a string.

The initial value of this field is MQCFT_STRING.

StrucLength (MQLONG)
Structure length.

This is the length in bytes of the MQCFST structure, including the variable-length string at the end of the structure (the *String* field). The length must be a multiple of four, and must be sufficient to contain the string; any bytes between the end of the string and the length defined by the *StrucLength* field are not significant.

The following constant gives the length of the *fixed* part of the structure, that is the length excluding the *String* field:

MQCFST_STRUC_LENGTH_FIXED
Length of fixed part of command format string-parameter structure.

The initial value of this field is MQCFST_STRUC_LENGTH_FIXED.

Parameter (MQLONG)
Parameter identifier.

This identifies the parameter whose value is contained in the structure. The values that can occur in this field depend on the value of the *Command* field in the MQCFH structure; see page 334 for details.

The initial value of this field is `0`.

CodedCharSetId (MQLONG)
Coded character set identifier.

This specifies the coded character set identifier of the data in the *String* field. The following special value can be used:

MQCCSI_DEFAULT
Default coded character set identifier.

Character data is in the character set defined by the *CodedCharSetId* field in the message descriptor MQMD.

The initial value of this field is MQCCSI_DEFAULT.

StringLength (MQLONG)
Length of string.

This is the length in bytes of the data in the *String* field; it must be zero or greater. This length need not be a multiple of four.

The initial value of this field is 0.

String (MQCHAR×*StringLength*)
String value.

This is the value of the parameter identified by the *Parameter* field:

- In MQFMT_ADMIN command messages, if the specified string is shorter than the standard length of the parameter, the omitted characters are assumed to be blanks. If the specified string is longer than the standard length, those characters in excess of the standard length must be blanks.
- In MQFMT_ADMIN response messages, string parameters are returned padded with blanks to the standard length of the parameter.
- In MQFMT_EVENT messages, trailing blanks are omitted from string parameters (that is, the string may be shorter than the defined length of the parameter).

In all cases, *StringLength* gives the length of the string actually present in the message.

The string can contain any characters that are in the character set defined by *CodedCharSetId*, and that are valid for the parameter identified by *Parameter*.

Note: In the MQCFST structure, a null character in the string is treated as normal data, and does not act as a delimiter for the string. This means that when a receiving application reads a MQFMT_PCF, MQFMT_EVENT, or MQFMT_ADMIN message, the receiving application receives all of the data specified by the sending application. The data may, of course, have been converted between character sets (for example, by the receiving application specifying the MQGMO_CONVERT option on the MQGET call).

In contrast, when the queue manager reads an MQFMT_ADMIN message from the command input queue, the queue manager processes the data as though it had been specified on an MQI call. This means that within the string, the first null and the characters following it (up to the end of the string) are treated as blanks.

The way that this field is declared depends on the programming language:

- For the C programming language, the field is declared as an array with one element. Storage for the structure should be allocated dynamically, and pointers used to address the fields within it.
- For the COBOL, PL/I, and System/390 assembler programming languages, the field is omitted from the structure declaration. When an instance of the structure is declared, the user should include MQCFST in a larger structure, and declare additional field(s) following MQCFST, to represent the *String* field as required.

In C, the initial value of this field is the null string.

Table 20. Initial values of fields in MQCFST

Field name	Name of constant	Value of constant
Type	MQCFT_STRING	4
StrucLength	MQCFST_STRUC_LENGTH_FIXED	20
Parameter	None	0
CodedCharSetId	MQCCSI_DEFAULT	0
StringLength	None	0
String (present only in C)	None	Null string

Notes:

1. In the C programming language, the macro variable MQCFST_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:

```
struct {
  MQCFST Hdr;
  MQCHAR Data[99];
} MyCFST = {MQCFST_DEFAULT};
```

C language declaration

```
typedef struct tagMQCFST {
  MQLONG  Type;            /* Structure type */
  MQLONG  StrucLength;     /* Structure length */
  MQLONG  Parameter;       /* Parameter identifier */
  MQLONG  CodedCharSetId;  /* Coded character set identifier */
  MQLONG  StringLength;    /* Length of string */
  MQCHAR  String[1];       /* String value - first
                              character */
 } MQCFST;
```

COBOL language declaration

```
**   MQCFST structure
  10 MQCFST.
**    Structure type
   15 MQCFST-TYPE           PIC S9(9) BINARY.
**    Structure length
   15 MQCFST-STRUCLENGTH    PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFST-PARAMETER      PIC S9(9) BINARY.
**    Coded character set identifier
   15 MQCFST-CODEDCHARSETID PIC S9(9) BINARY.
**    Length of string
   15 MQCFST-STRINGLENGTH   PIC S9(9) BINARY.
```

PL/I language declaration (AIX, MVS/ESA, OS/2, and Windows NT)

```
dcl
 1 MQCFST based,
  3 Type           fixed bin(31), /* Structure type */
  3 StrucLength    fixed bin(31), /* Structure length */
  3 Parameter      fixed bin(31), /* Parameter identifier */
  3 CodedCharSetId fixed bin(31), /* Coded character set identifier */
  3 StringLength   fixed bin(31); /* Length of string */
```

System/390 assembler-language declaration (MVS/ESA only)

```
MQCFST                  DSECT
MQCFST_TYPE             DS   F        Structure type
MQCFST_STRUCLENGTH      DS   F        Structure length
MQCFST_PARAMETER        DS   F        Parameter identifier
MQCFST_CODEDCHARSETID   DS   F        Coded character set
*                                     identifier
MQCFST_STRINGLENGTH     DS   F        Length of string
MQCFST_LENGTH           EQU  *-MQCFST Length of structure
                        ORG  MQCFST
MQCFST_AREA             DS   CL(MQCFST_LENGTH)
```

MQCFIL – PCF integer list parameter

The MQCFIL structure describes an integer-list parameter in a message that is a command or a response to a command. In either case, the format name in the message descriptor is MQFMT_ADMIN.

The MQCFIL structure is also used for event messages. In this case the message descriptor *Format* field is MQFMT_EVENT.

The MQCFIL structure can also be used for user-defined message data. In this case the message descriptor *Format* field is MQFMT_PCF (see “Message descriptor for a PCF command” on page 127). Also in this case, not all of the fields in the structure are meaningful. The supplied initial values can be used for most fields, but the application must set the *StrucLength*, *Count*, and *Values* fields to the values appropriate to the data.

The structure ends with a variable-length array of integers; see the *Values* field below for further details.

Type (MQLONG)
Structure type.

This indicates that the structure is an MQCFIL structure describing an integer-list parameter. The value must be:

MQCFT_INTEGER_LIST
Structure defining an integer list.

The initial value of this field is MQCFT_INTEGER_LIST.

StrucLength (MQLONG)
Structure length.

This is the length in bytes of the MQCFIL structure, including the variable-size array of integers at the end of the structure (the *Values* field). The length must be a multiple of four, and must be sufficient to contain the array; any bytes between the end of the array and the length defined by the *StrucLength* field are not significant.

The following constant gives the length of the *fixed* part of the structure, that is the length excluding the *Values* field:

MQCFIL_STRUC_LENGTH_FIXED
Length of fixed part of command format integer-list parameter structure.

The initial value of this field is MQCFIL_STRUC_LENGTH_FIXED.

Parameter (MQLONG)
Parameter identifier.

This identifies the parameter whose values are contained in the structure. The values that can occur in this field depend on the value of the *Command* field in the MQCFH structure; see page 334 for details.

The initial value of this field is 0.

Count (MQLONG)
Count of parameter values.

This is the number of elements in the *Values* array; it must be zero or greater.

The initial value of this field is 0.

Values (MQLONG×*Count*)
Parameter values.

This is an array of values for the parameter identified by the *Parameter* field. For example, for MQIACF_Q_ATTRS, this is a list of attribute selectors (MQCA_* and MQIA_* values).

The way that this field is declared depends on the programming language:

- For the C programming language, the field is declared as an array with one element. Storage for the structure should be allocated dynamically, and pointers used to address the fields within it.
- For the COBOL, PL/I, and System/390 assembler programming languages, the field is omitted from the structure declaration. When an instance of the structure is declared, the user should include MQCFIN in a larger structure, and declare additional field(s) following MQCFIN, to represent the *Values* field as required.

In C, the initial value of this field is a single 0.

Table 21. Initial values of fields in MQCFIL

Field name	Name of constant	Value of constant
Type	MQCFT_INTEGER_LIST	5
StrucLength	MQCFIL_STRUC_LENGTH_FIXED	16
Parameter	None	0
Count	None	0
Values (present only in C)	None	0

Notes:

1. In the C programming language, the macro variable MQCFIL_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:

```
struct {
  MQCFIL Hdr;
  MQLONG Data[99];
} MyCFIL = {MQCFIL_DEFAULT};
```

C language declaration

```
typedef struct tagMQCFIL {
  MQLONG  Type;         /* Structure type */
  MQLONG  StrucLength;  /* Structure length */
  MQLONG  Parameter;    /* Parameter identifier */
  MQLONG  Count;        /* Count of parameter values */
  MQLONG  Values[1];    /* Parameter values - first element */
 } MQCFIL;
```

COBOL language declaration

```
**   MQCFIL structure
  10 MQCFIL.
**    Structure type
   15 MQCFIL-TYPE        PIC S9(9) BINARY.
**    Structure length
   15 MQCFIL-STRUCLENGTH PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFIL-PARAMETER   PIC S9(9) BINARY.
**    Count of parameter values
   15 MQCFIL-COUNT       PIC S9(9) BINARY.
```

PL/I language declaration (AIX, MVS/ESA, OS/2, and Windows NT)

```
dcl
 1 MQCFIL based,
  3 Type        fixed bin(31), /* Structure type */
  3 StrucLength fixed bin(31), /* Structure length */
  3 Parameter   fixed bin(31), /* Parameter identifier */
  3 Count       fixed bin(31); /* Count of parameter values */
```

System/390 assembler-language declaration (MVS/ESA only)

```
MQCFIL                  DSECT
MQCFIL_TYPE             DS   F        Structure type
MQCFIL_STRUCLENGTH      DS   F        Structure length
MQCFIL_PARAMETER        DS   F        Parameter identifier
MQCFIL_COUNT            DS   F        Count of parameter values
MQCFIL_LENGTH           EQU  *-MQCFIL Length of structure
                        ORG  MQCFIL
MQCFIL_AREA             DS   CL(MQCFIL_LENGTH)
```

MQCFSL – PCF string list parameter

The MQCFSL structure describes a string-list parameter in a message which is a command or a response to a command. In either case, the format name in the message descriptor is MQFMT_ADMIN.

The MQCFSL structure is also used for event messages. In this case the message descriptor *Format* field is MQFMT_EVENT.

The MQCFSL structure can also be used for user-defined message data. In this case the message descriptor *Format* field is MQFMT_PCF (see “Message descriptor for a PCF command” on page 127). Also in this case, not all of the fields in the structure are meaningful. The supplied initial values can be used for most fields, but the application must set the *StrucLength*, *Count*, *StringLength*, and *Strings* fields to the values appropriate to the data.

The structure ends with a variable-length array of character strings; see the *Strings* field below for further details.

See "Usage notes" on page 334 for further information on how the structure should be used.

Type (MQLONG)
Structure type.

This indicates that the structure is an MQCFSL structure describing a string-list parameter. The value must be:

MQCFT_STRING_LIST
Structure defining a string list.

The initial value of this field is MQCFT_STRING_LIST.

StrucLength (MQLONG)
Structure length.

This is the length in bytes of the MQCFSL structure, including the variable-length data at the end of the structure (the *Strings* field). The length must be a multiple of four, and must be sufficient to contain all of the strings; any bytes between the end of the strings and the length defined by the *StrucLength* field are not significant.

The following constant gives the length of the *fixed* part of the structure, that is the length excluding the *Strings* field:

MQCFSL_STRUC_LENGTH_FIXED
Length of fixed part of command format string-list parameter structure.

The initial value of this field is MQCFSL_STRUC_LENGTH_FIXED.

Parameter (MQLONG)
Parameter identifier.

This identifies the parameter whose values are contained in the structure. The values that can occur in this field depend on the value of the *Command* field in the MQCFH structure; see page 334 for details.

The initial value of this field is `0`.

CodedCharSetId (MQLONG)
Coded character set identifier.

This specifies the coded character set identifier of the data in the *Strings* field. The following special value can be used:

MQCCSI_DEFAULT
Default coded character set identifier.

Character data is in the character set defined by the *CodedCharSetId* field in the message descriptor MQMD.

The initial value of this field is MQCCSI_DEFAULT.

Count (MQLONG)
Count of parameter values.

This is the number of strings present in the *Strings* field; it must be zero or greater.

The initial value of this field is 0.

StringLength (MQLONG)
Length of one string.

This is the length in bytes of one parameter value, that is the length of one string in the *Strings* field; all of the strings are this length. The length must be zero or greater, and need not be a multiple of four.

The initial value of this field is 0.

Strings (MQCHAR×*StringLength*×*Count*)
String values.

This is a set of string values for the parameter identified by the *Parameter* field. The number of strings is given by the *Count* field, and the length of each string is given by the *StringLength* field. The strings are concatenated together, with no bytes skipped between adjacent strings. The total length of the strings is the length of one string multiplied by the number of strings present (that is, *StringLength*×*Count*).

- In MQFMT_ADMIN command messages, if the specified string is shorter than the standard length of the parameter, the omitted characters are assumed to be blanks. If the specified string is longer than the standard length, those characters in excess of the standard length must be blanks.
- In MQFMT_ADMIN response messages, string parameters are returned padded with blanks to the standard length of the parameter.
- In MQFMT_EVENT messages, trailing blanks are omitted from string parameters (that is, the string may be shorter than the defined length of the parameter).

In all cases, *StringLength* gives the length of the string actually present in the message.

The strings can contain any characters that are in the character set defined by *CodedCharSetId*, and that are valid for the parameter identified by *Parameter*.

Note: In the MQCFSL structure, a null character in a string is treated as normal data, and does not act as a delimiter for the string. This means that when a receiving application reads a MQFMT_PCF, MQFMT_EVENT, or MQFMT_ADMIN message, the receiving application receives all of the data specified by the sending application. The data may, of course, have been converted between character sets (for example, by the receiving application specifying the MQGMO_CONVERT option on the MQGET call).

In contrast, when the queue manager reads an MQFMT_ADMIN message from the command input queue, the queue manager processes the data as though it had been specified on an MQI call. This means that within each string, the first null and the characters following it (up to the end of the string) are treated as blanks.

The way that this field is declared depends on the programming language:

- For the C programming language, the field is declared as an array with one element. Storage for the structure should be allocated dynamically, and pointers used to address the fields within it.

- For the COBOL, PL/I, and System/390 assembler programming languages, the field is omitted from the structure declaration. When an instance of the structure is declared, the user should include MQCFSL in a larger structure, and declare additional field(s) following MQCFSL, to represent the *Strings* field as required.

In C, the initial value of this field is the null string.

Table 22. Initial values of fields in MQCFSL

Field name	Name of constant	Value of constant
Type	MQCFT_STRING_LIST	6
StrucLength	MQCFSL_STRUC_LENGTH_FIXED	24
Parameter	None	0
CodedCharSetId	MQCCSI_DEFAULT	0
Count	None	0
StringLength	None	0
Strings (present only in C)	None	Null string

Notes:

1. In the C programming language, the macro variable MQCFSL_DEFAULT contains the values listed above. It can be used in the following way to provide initial values for the fields in the structure:

```
struct {
  MQCFSL Hdr;
  MQCHAR Data[999];
} MyCFSL = {MQCFSL_DEFAULT};
```

C language declaration

```
typedef struct tagMQCFSL {
  MQLONG  Type;            /* Structure type */
  MQLONG  StrucLength;     /* Structure length */
  MQLONG  Parameter;       /* Parameter identifier */
  MQLONG  CodedCharSetId;  /* Coded character set identifier */
  MQLONG  Count;           /* Count of parameter values */
  MQLONG  StringLength;    /* Length of one string */
  MQCHAR  Strings[1];      /* String values - first
                              character */
 } MQCFSL;
```

COBOL language declaration

```
**   MQCFSL structure
  10 MQCFSL.
**    Structure type
   15 MQCFSL-TYPE           PIC S9(9) BINARY.
**    Structure length
   15 MQCFSL-STRUCLENGTH    PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFSL-PARAMETER      PIC S9(9) BINARY.
**    Coded character set identifier
   15 MQCFSL-CODEDCHARSETID PIC S9(9) BINARY.
**    Count of parameter values
```

```
   15 MQCFSL-COUNT          PIC S9(9) BINARY.
**    Length of one string
   15 MQCFSL-STRINGLENGTH   PIC S9(9) BINARY.
```

PL/I language declaration (AIX, MVS/ESA, OS/2, and Windows NT)

```
dcl
 1 MQCFSL based,
  3 Type           fixed bin(31), /* Structure type */
  3 StrucLength    fixed bin(31), /* Structure length */
  3 Parameter      fixed bin(31), /* Parameter identifier */
  3 CodedCharSetId fixed bin(31), /* Coded character set identifier */
  3 Count          fixed bin(31), /* Count of parameter values */
  3 StringLength   fixed bin(31); /* Length of one string */
```

System/390 assembler-language declaration (MVS/ESA only)

```
MQCFSL                      DSECT
MQCFSL_TYPE                 DS   F        Structure type
MQCFSL_STRUCLENGTH          DS   F        Structure length
MQCFSL_PARAMETER            DS   F        Parameter identifier
MQCFSL_CODEDCHARSETID       DS   F        Coded character set
*                                         identifier
MQCFSL_COUNT                DS   F        Count of parameter values
MQCFSL_STRINGLENGTH         DS   F        Length of one string
MQCFSL_LENGTH               EQU  *-MQCFSL Length of structure
                            ORG  MQCFSL
MQCFSL_AREA                 DS   CL(MQCFSL_LENGTH)
```

Chapter 10. Example of using PCFs

This is an example of how Programmable Command Formats could be used in a program for administration of MQSeries queues.

Enquire local queue attributes

A C language program is listed here that uses MQSeries for OS/2 V2.0. It is given as an example of using PCFs and has been limited to a simple case. This program will be of most use as an example if you are considering the use of PCFs to manage your MQSeries environment.

The program, once compiled, will inquire of the default queue manager about a subset of the attributes for all local queues defined to it. It then produces an output file, SAVEQMGR.TST, in the directory from which it was run. This file is of a format suitable for use with RUNMQSC.

Program listing

```
/*==========================================================================*/
/*                                                                          */
/*              (C) Copyright IBM Corporation 1995                          */
/*==========================================================================*/
/* v1.0  12-05-95 NDC  Created                                              */
/*==========================================================================*/
/* Module Name: MS02.C                                                      */
/*==========================================================================*/
/*                                                                          */
/* This is a program to inquire of the default queue manager about the      */
/* local queues defined to it.                                              */
/*                                                                          */
/* The program takes this information and appends it to a file              */
/* SAVEQMGR.TST which is of a format suitable for RUNMQSC. It could,        */
/* therefore, be used to recreate or clone a queue manager.                 */
/*                                                                          */
/* It is offered as an example of using Programmable Command Formats (PCFs) */
/* as a method for administering a queue manager.                           */
/*                                                                          */
/*==========================================================================*/



/* Include standard libraries */
#include <memory.h>
#include <stdio.h>

/* Include MQSeries headers */
#include <cmqc.h>
#include <cmqcfc.h>
#include <cmqxc.h>


typedef struct LocalQParms {
   MQCHAR48    QName;
   MQLONG      QType;
   MQCHAR64    QDesc;
   MQLONG      InhibitPut;
   MQLONG      DefPriority;
   MQLONG      DefPersistence;
   MQLONG      InhibitGet;
   MQCHAR48    ProcessName;
   MQLONG      MaxQDepth;
   MQLONG      MaxMsgLength;
```

```
   MQLONG      BackoutThreshold;
   MQCHAR48    BackoutReqQName;
   MQLONG      Shareability;
   MQLONG      DefInputOpenOption;
   MQLONG      HardenGetBackout;
   MQLONG      MsgDeliverySequence;
   MQLONG      RetentionInterval;
   MQLONG      DefinitionType;
   MQLONG      Usage;
   MQLONG      OpenInputCount;
   MQLONG      OpenOutputCount;
   MQLONG      CurrentQDepth;
   MQCHAR12    CreationDate;
   MQCHAR8     CreationTime;
   MQCHAR48    InitiationQName;
   MQLONG      TriggerControl;
   MQLONG      TriggerType;
   MQLONG      TriggerMsgPriority;
   MQLONG      TriggerDepth;
   MQCHAR64    TriggerData;
   MQLONG      Scope;
   MQLONG      QDepthHighLimit;
   MQLONG      QDepthLowLimit;
   MQLONG      QDepthMaxEvent;
   MQLONG      QDepthHighEvent;
   MQLONG      QDepthLowEvent;
   MQLONG      QServiceInterval;
   MQLONG      QServiceIntervalEvent;
} LocalQParms;


void ProcessStringParm( MQCFST *pPCFString, LocalQParms *DefnLQ );

void ProcessIntegerParm( MQCFIN *pPCFInteger, LocalQParms *DefnLQ );

int AddToFileQLOCAL( LocalQParms DefnLQ );

void MQParmCpy( char *target, char *source, int length );

void PutMsg( MQHCONN   hConn       /* Connection to queue manager            */
           , MQCHAR8   MsgFormat   /* Format of user data to be put in msg   */
           , MQHOBJ    hQName      /* handle of queue to put the message to  */
           , MQCHAR48  QName       /* name of queue to put the message to    */
           , MQBYTE   *UserMsg     /* The user data to be put in the message */
           , MQLONG    UserMsgLen  /*                                        */
           );

void GetMsg( MQHCONN   hConn          /* handle of queue manager           */
           , MQLONG    MQParm         /* Options to specify nature of get  */
           , MQHOBJ    hQName         /* handle of queue to read from      */
           , MQCHAR48  QName          /* name of queue to read from        */
           , MQBYTE   *UserMsg        /* Input/Output buffer containing msg */
           , MQLONG    ReadBufferLen  /* Length of supplied buffer         */
           );
MQHOBJ OpenQ( MQHCONN     hConn
            , MQCHAR48    QName
            , MQLONG      OpenOpts
            );




int main( int argc, char *argv[] )
{
  MQCHAR48             QMgrName;         /* Name of connected queue mgr     */
  MQHCONN              hConn;            /* handle to connected queue mgr   */
  MQOD                 ObjDesc;          /*                                 */
  MQLONG               OpenOpts;         /*                                 */
  MQLONG               CompCode;         /* MQ API completion code          */
```

```
MQLONG              Reason;          /* Reason qualifying above        */
                                     /*                                */
MQHOBJ              hAdminQ;         /* handle to output queue         */
MQHOBJ              hReplyQ;         /* handle to input queue          */
                                     /*                                */
MQLONG              AdminMsgLen;     /* Length of user message buffer  */
MQBYTE             *pAdminMsg;       /* Ptr to outbound data buffer    */
MQCFH              *pPCFHeader;      /* Ptr to PCF header structure    */
MQCFST             *pPCFString;      /* Ptr to PCF string parm block   */
MQCFIN             *pPCFInteger;     /* Ptr to PCF integer parm block  */
MQLONG             *pPCFType;        /* Type field of PCF message parm */
LocalQParms         DefnLQ;          /*                                */
                                     /*                                */
char                ErrorReport[40]; /*                        */
MQCHAR8             MsgFormat;       /* Format of inbound message      */
short               Index;           /* Loop counter                   */



/* Connect to default queue manager */
memset( QMgrName, '\0', sizeof( QMgrName ) );
MQCONN(  QMgrName                    /* I  : use default queue manager */
      , &hConn                       /*  O : queue manager handle  */
      , &CompCode                    /*  O : Completion code           */
      , &Reason                      /*  O : Reason qualifying CompCode */
      );

if ( CompCode != MQCC_OK ) {
   printf( "MQCONN failed for %s, CC=%d RC=%d\n"
         , QMgrName
         , CompCode
         , Reason
         );
   exit( -1 );
} /* endif */


/* Open all the required queues */
hAdminQ = OpenQ( hConn, "SYSTEM.ADMIN.COMMAND.QUEUE\0", MQOO_OUTPUT );

hReplyQ = OpenQ( hConn, "SAVEQMGR.REPLY.QUEUE\0", MQOO_INPUT_EXCLUSIVE );


/* ****************************************************************** */
/* Put a message to the SYSTEM.ADMIN.COMMAND.QUEUE to inquire all     */
/* the local queues defined on the queue manager.                     */
/*                                                                    */
/* The request consists of a Request Header and a parameter block     */
/* used to specify the generic search. The header and the parameter   */
/* block follow each other in a contiguous buffer which is pointed    */
/* to by the variable pAdminMsg. This entire buffer is then put to    */
/* the queue.                                                         */
/*                                                                    */
/* The command server, (use STRMQCSV to start it), processes the      */
/* SYSTEM.ADMIN.COMMAND.QUEUE and puts a reply on the application     */
/* ReplyToQ for each defined queue.                                   */
/* ****************************************************************** */

/* Set the length for the message buffer */
AdminMsgLen = MQCFH_STRUC_LENGTH
            + MQCFST_STRUC_LENGTH_FIXED + MQ_Q_NAME_LENGTH
            + MQCFIN_STRUC_LENGTH
            ;

/* ------------------------------------------------------------------ */
/* Set pointers to message data buffers                              */
/*                                                                    */
/* pAdminMsg points to the start of the message buffer               */
/*                                                                    */
/* pPCFHeader also points to the start of the message buffer. It is  */
/* used to indicate the type of command we wish to execute and the   */
```

```
/* number of parameter blocks following in the message buffer.       */
/*                                                                   */
/* pPCFString points into the message buffer immediatley after the   */
/* header and is used to map the following bytes onto a PCF string   */
/* parameter block. In this case the string is used to indicate the  */
/* nameof the queue we want details about, * indicating al queues.   */
/*                                                                   */
/* pPCFInteger points into the message buffer immediately after the  */
/* string block described above. It is used to map the following     */
/* bytes onto a PCF integer parameter block. This block indicates    */
/* the type of queue we wish to receive details about, thereby       */
/* qualifying the generic search set up by passing the previous      */
/* string parameter.                                                 */
/*                                                                   */
/* Note that this example is a generic search for all attributes of  */
/* all local queues known to the queue manager. By using different,  */
/* or more, parameter blocks in the request header it is possible    */
/* to narrow the search.                                             */
/* ----------------------------------------------------------------- */

pAdminMsg   = (MQBYTE *)malloc( AdminMsgLen );

pPCFHeader  = (MQCFH *)pAdminMsg;

pPCFString  = (MQCFST *)(pAdminMsg
                       + MQCFH_STRUC_LENGTH
                       );

pPCFInteger = (MQCFIN *)( pAdminMsg
                       + MQCFH_STRUC_LENGTH
                       + MQCFST_STRUC_LENGTH_FIXED + MQ_Q_NAME_LENGTH
                       );


/* Setup request header */
pPCFHeader->Type           = MQCFT_COMMAND;
pPCFHeader->StrucLength    = MQCFH_STRUC_LENGTH;
pPCFHeader->Version        = MQCFH_VERSION_1;
pPCFHeader->Command        = MQCMD_INQUIRE_Q;
pPCFHeader->MsgSeqNumber   = MQCFC_LAST;
pPCFHeader->Control        = MQCFC_LAST;
pPCFHeader->ParameterCount = 2;

/* Setup parameter block */
pPCFString->Type           = MQCFT_STRING;
pPCFString->StrucLength    = MQCFST_STRUC_LENGTH_FIXED + MQ_Q_NAME_LENGTH;
pPCFString->Parameter      = MQCA_Q_NAME;
pPCFString->CodedCharSetId = MQCCSI_DEFAULT;
pPCFString->StringLength   = MQ_Q_NAME_LENGTH;
memset( pPCFString->String, ' ', MQ_Q_NAME_LENGTH );
memcpy( pPCFString->String, "*", 1 );

/* Setup parameter block */
pPCFInteger->Type        = MQCFT_INTEGER;
pPCFInteger->StrucLength = MQCFIN_STRUC_LENGTH;
pPCFInteger->Parameter   = MQIA_Q_TYPE;
pPCFInteger->Value       = MQQT_LOCAL;

PutMsg(  hConn               /* Queue manager handle            */
      ,  MQFMT_ADMIN         /* Format of message               */
      ,  hAdminQ             /* Handle of command queue         */
      , "SYSTEM.ADMIN.COMMAND.QUEUE\0"
      ,  (MQBYTE *)pAdminMsg  /* Data part of message to put     */
      ,  AdminMsgLen
      );

free( pAdminMsg );


/* ***************************************************************** */
/* Get and process the replies received from the command server onto  */
/* the applications ReplyToQ.                                        */
```

```
/*                                                                  */
/* There will be one message per defined local queue.               */
/*                                                                  */
/* The last message will have the Control field of the PCF header   */
/* set to MQCFC_LAST. All others will be MQCFC_NOT_LAST.            */
/*                                                                  */
/* An individual Reply message consists of a header followed by a   */
/* number a parameters, the exact number, type and order will depend */
/* upon the type of request.                                        */
/*                                                                  */
/* ---------------------------------------------------------------- */
/*                                                                  */
/* The message is retrieved into a buffer pointed to by pAdminMsg.  */
/* This buffer as been allocated to be large enough to hold all the */
/* parameters for a local queue definition.                         */
/*                                                                  */
/* pPCFHeader is then allocated to point also to the beginning of   */
/* the buffer and is used to access the PCF header structure. The   */
/* header contains several fields. The one we are specifically      */
/* interested in is the ParameterCount. This tells us how many      */
/* parameters follow the header in the message buffer. There is     */
/* one parameter for each local queue attribute known by the        */
/* queue manager.                                                   */
/*                                                                  */
/* At this point we do not know the order or type of each parameter */
/* block in the buffer, the first MQLONG of each block defines its  */
/* type; they may be parameter blocks containing either strings or  */
/* integers.                                                        */
/*                                                                  */
/* pPCFType is used initially to point to the first byte beyond the */
/* known parameter block. Initially then, it points to the first byte */
/* after the PCF header. Subsequently it is incremented by the length */
/* of the identified parameter block and therefore points at the    */
/* next. Looking at the value of the data pointed to by pPCFType we */
/* can decide how to process the next group of bytes, either as a   */
/* string, or an integer.                                           */
/*                                                                  */
/* In this way we parse the message buffer extracting the values of */
/* each of the parameters we are interested in.                     */
/*                                                                  */
/* **************************************************************** */

/* AdminMsgLen is to be set to the length of the expected reply     */
/* message. This structure is specific to Local Queues.             */
AdminMsgLen = MQCFH_STRUC_LENGTH
            + (MQCFST_STRUC_LENGTH_FIXED * 12)
            + (MQCFIN_STRUC_LENGTH * 30)
            + MQ_Q_NAME_LENGTH
            + MQ_Q_DESC_LENGTH
            + MQ_PROCESS_NAME_LENGTH
            + MQ_Q_NAME_LENGTH
            + MQ_CREATION_DATE_LENGTH
            + MQ_CREATION_TIME_LENGTH
            + MQ_Q_NAME_LENGTH
            + MQ_TRIGGER_DATA_LENGTH
            + MQ_Q_NAME_LENGTH
            + MQ_Q_NAME_LENGTH
            + MQ_Q_MGR_NAME_LENGTH
            + MQ_Q_NAME_LENGTH
            ;

/* Set pointers to message data buffers */
pAdminMsg = (MQBYTE *)malloc( AdminMsgLen );

do {

   GetMsg(  hConn                     /* Queue manager handle          */
         ,  MQGMO_WAIT
                                      /* Parameters on Get             */
         ,  hReplyQ                   /* Get queue handle              */
         ,  "SAVEQMGR.REPLY.QUEUE\0"
         ,  (MQBYTE *)pAdminMsg       /* pointer to message area       */
```

```
            ,  AdminMsgLen               /* length of get buffer           */
            );

      /* Examine Header */
      pPCFHeader = (MQCFH *)pAdminMsg;

      /* Examine first parameter */
      pPCFType = (MQLONG *)(pAdminMsg + MQCFH_STRUC_LENGTH);

      Index = 1;

      while ( Index <= pPCFHeader->ParameterCount ) {

         /* Establish the type of each parameter and allocate  */
         /* a pointer of the correct type to reference it.     */
         switch ( *pPCFType ) {
         case MQCFT_INTEGER:
            pPCFInteger = (MQCFIN *)pPCFType;
            ProcessIntegerParm( pPCFInteger, &DefnLQ );
            Index++;
            /* Increment the pointer to the next parameter by the */
            /* length of the current parm.                        */
            pPCFType = (MQLONG *)( (MQBYTE *)pPCFType
                                 + pPCFInteger->StrucLength
                                 );
            break;
         case MQCFT_STRING:
            pPCFString = (MQCFST *)pPCFType;
            ProcessStringParm( pPCFString, &DefnLQ );
            Index++;
            /* Increment the pointer to the next parameter by the */
            /* length of the current parm.                        */
            pPCFType = (MQLONG *)( (MQBYTE *)pPCFType
                                 + pPCFString->StrucLength
                                 );
            break;
         } /* endswitch */

      } /* endwhile */

      /* ------------------------------------------------------- */
      /* Message parsed, append to output file                   */
      /* ------------------------------------------------------- */
      AddToFileQLOCAL( DefnLQ );


      /* ------------------------------------------------------- */
      /* Finished processing the current message, do the next one. */
      /* ------------------------------------------------------- */

   } while ( pPCFHeader->Control == MQCFC_NOT_LAST ); /* enddo */

   free( pAdminMsg );

   /* ------------------------------------ */
   /* Processing of the local queues complete */
   /* ------------------------------------ */

}


void ProcessStringParm( MQCFST *pPCFString, LocalQParms *DefnLQ )
{
   switch ( pPCFString->Parameter ) {
   case MQCA_Q_NAME:
      MQParmCpy( DefnLQ->QName, pPCFString->String, 48 );
      break;
   case MQCA_Q_DESC:
      MQParmCpy( DefnLQ->QDesc, pPCFString->String, 64 );
      break;
   case MQCA_PROCESS_NAME:
      MQParmCpy( DefnLQ->ProcessName, pPCFString->String, 48 );
```

```
      break;
   case MQCA_BACKOUT_REQ_Q_NAME:
      MQParmCpy( DefnLQ->BackoutReqQName, pPCFString->String, 48 );
      break;
   case MQCA_CREATION_DATE:
      MQParmCpy( DefnLQ->CreationDate, pPCFString->String, 12 );
      break;
   case MQCA_CREATION_TIME:
      MQParmCpy( DefnLQ->CreationTime, pPCFString->String, 8 );
      break;
   case MQCA_INITIATION_Q_NAME:
      MQParmCpy( DefnLQ->InitiationQName, pPCFString->String, 48 );
      break;
   case MQCA_TRIGGER_DATA:
      MQParmCpy( DefnLQ->TriggerData, pPCFString->String, 64 );
      break;
   } /* endswitch */
}


void ProcessIntegerParm( MQCFIN *pPCFInteger, LocalQParms *DefnLQ )
{
   switch ( pPCFInteger->Parameter ) {
   case MQIA_Q_TYPE:
      DefnLQ->QType = pPCFInteger->Value;
      break;
   case MQIA_INHIBIT_PUT:
      DefnLQ->InhibitPut = pPCFInteger->Value;
      break;
   case MQIA_DEF_PRIORITY:
      DefnLQ->DefPriority = pPCFInteger->Value;
      break;
   case MQIA_DEF_PERSISTENCE:
      DefnLQ->DefPersistence = pPCFInteger->Value;
      break;
   case MQIA_INHIBIT_GET:
      DefnLQ->InhibitGet = pPCFInteger->Value;
      break;
   case MQIA_SCOPE:
      DefnLQ->Scope = pPCFInteger->Value;
      break;
   case MQIA_MAX_Q_DEPTH:
      DefnLQ->MaxQDepth = pPCFInteger->Value;
      break;
   case MQIA_MAX_MSG_LENGTH:
      DefnLQ->MaxMsgLength = pPCFInteger->Value;
      break;
   case MQIA_BACKOUT_THRESHOLD:
      DefnLQ->BackoutThreshold = pPCFInteger->Value;
      break;
   case MQIA_SHAREABILITY:
      DefnLQ->Shareability = pPCFInteger->Value;
      break;
   case MQIA_DEF_INPUT_OPEN_OPTION:
      DefnLQ->DefInputOpenOption = pPCFInteger->Value;
      break;
   case MQIA_HARDEN_GET_BACKOUT:
      DefnLQ->HardenGetBackout = pPCFInteger->Value;
      break;
   case MQIA_MSG_DELIVERY_SEQUENCE:
      DefnLQ->HardenGetBackout = pPCFInteger->Value;
      break;
   case MQIA_RETENTION_INTERVAL:
      DefnLQ->RetentionInterval = pPCFInteger->Value;
      break;
   case MQIA_DEFINITION_TYPE:
      DefnLQ->DefinitionType = pPCFInteger->Value;
      break;
   case MQIA_USAGE:
      DefnLQ->Usage = pPCFInteger->Value;
      break;
   case MQIA_OPEN_INPUT_COUNT:
```

```
      DefnLQ->OpenInputCount = pPCFInteger->Value;
      break;
   case MQIA_OPEN_OUTPUT_COUNT:
      DefnLQ->OpenOutputCount = pPCFInteger->Value;
      break;
   case MQIA_CURRENT_Q_DEPTH:
      DefnLQ->CurrentQDepth = pPCFInteger->Value;
      break;
   case MQIA_TRIGGER_CONTROL:
      DefnLQ->TriggerControl = pPCFInteger->Value;
      break;
   case MQIA_TRIGGER_TYPE:
      DefnLQ->TriggerType = pPCFInteger->Value;
      break;
   case MQIA_TRIGGER_MSG_PRIORITY:
      DefnLQ->TriggerMsgPriority = pPCFInteger->Value;
      break;
   case MQIA_TRIGGER_DEPTH:
      DefnLQ->TriggerDepth = pPCFInteger->Value;
      break;
   case MQIA_Q_DEPTH_HIGH_LIMIT:
      DefnLQ->QDepthHighLimit = pPCFInteger->Value;
      break;
   case MQIA_Q_DEPTH_LOW_LIMIT:
      DefnLQ->QDepthLowLimit = pPCFInteger->Value;
      break;
   case MQIA_Q_DEPTH_MAX_EVENT:
      DefnLQ->QDepthMaxEvent = pPCFInteger->Value;
      break;
   case MQIA_Q_DEPTH_HIGH_EVENT:
      DefnLQ->QDepthHighEvent = pPCFInteger->Value;
      break;
   case MQIA_Q_DEPTH_LOW_EVENT:
      DefnLQ->QDepthLowEvent = pPCFInteger->Value;
      break;
   case MQIA_Q_SERVICE_INTERVAL:
      DefnLQ->QServiceInterval = pPCFInteger->Value;
      break;
   case MQIA_Q_SERVICE_INTERVAL_EVENT:
      DefnLQ->QServiceIntervalEvent = pPCFInteger->Value;
      break;
   } /* endswitch */
}


/* ------------------------------------------------------------------------ */
/*                                                                          */
/* This process takes the attributes of a single local queue and adds them  */
/* to the end of a file, SAVEQMGR.TST, which can be found in the current    */
/* directory.                                                               */
/*                                                                          */
/* The file is of a format suitable for subsequent input to RUNMQSC.        */
/*                                                                          */
/* ------------------------------------------------------------------------ */
int AddToFileQLOCAL( LocalQParms DefnLQ )
{
   char    ParmBufferff120“;   /* Temporary buffer to hold for output to file */
   FILE   *fp;                 /* Pointer to a file                            */

   /* Append these details to the end of the current SAVEQMGR.TST file */
   fp = fopen( "SAVEQMGR.TST", "a" );

   sprintf( ParmBuffer, "DEFINE QLOCAL ('%s') REPLACE +\n", DefnLQ.QName );
   fputs( ParmBuffer, fp );

   sprintf( ParmBuffer, "       DESCR('%s') +\n" , DefnLQ.QDesc );
   fputs( ParmBuffer, fp );

   if ( DefnLQ.InhibitPut == MQQA_PUT_ALLOWED ) {
      sprintf( ParmBuffer, "       PUT(ENABLED) +\n" );
      fputs( ParmBuffer, fp );
   } else {
```

```
   sprintf( ParmBuffer, "       PUT(DISABLED) +\n" );
   fputs( ParmBuffer, fp );
} /* endif */

sprintf( ParmBuffer, "       DEFPRTY(%d) +\n", DefnLQ.DefPriority );
fputs( ParmBuffer, fp );

if ( DefnLQ.DefPersistence == MQPER_PERSISTENT ) {
   sprintf( ParmBuffer, "       DEFPSIST(YES) +\n" );
   fputs( ParmBuffer, fp );
} else {
   sprintf( ParmBuffer, "       DEFPSIST(NO) +\n" );
   fputs( ParmBuffer, fp );
} /* endif */

if ( DefnLQ.InhibitGet == MQQA_GET_ALLOWED ) {
   sprintf( ParmBuffer, "       GET(ENABLED) +\n" );
   fputs( ParmBuffer, fp );
} else {
   sprintf( ParmBuffer, "       GET(DISABLED) +\n" );
   fputs( ParmBuffer, fp );
} /* endif */

sprintf( ParmBuffer, "       MAXDEPTH(%d) +\n", DefnLQ.MaxQDepth );
fputs( ParmBuffer, fp );

sprintf( ParmBuffer, "       MAXMSGL(%d) +\n", DefnLQ.MaxMsgLength );
fputs( ParmBuffer, fp );

if ( DefnLQ.Shareability == MQQA_SHAREABLE ) {
   sprintf( ParmBuffer, "       SHARE +\n" );
   fputs( ParmBuffer, fp );
} else {
   sprintf( ParmBuffer, "       NOSHARE +\n" );
   fputs( ParmBuffer, fp );
} /* endif */

if ( DefnLQ.DefInputOpenOption == MQOO_INPUT_SHARED ) {
   sprintf( ParmBuffer, "       DEFSOPT(SHARED) +\n" );
   fputs( ParmBuffer, fp );
} else {
   sprintf( ParmBuffer, "       DEFSOPT(EXCL) +\n" );
   fputs( ParmBuffer, fp );
} /* endif */

if ( DefnLQ.MsgDeliverySequence == MQMDS_PRIORITY ) {
   sprintf( ParmBuffer, "       MSGDLVSQ(PRIORITY) +\n" );
   fputs( ParmBuffer, fp );
} else {
   sprintf( ParmBuffer, "       MSGDLVSQ(FIFO) +\n" );
   fputs( ParmBuffer, fp );
} /* endif */

if ( DefnLQ.HardenGetBackout == MQQA_BACKOUT_HARDENED ) {
   sprintf( ParmBuffer, "       HARDENBO +\n" );
   fputs( ParmBuffer, fp );
} else {
   sprintf( ParmBuffer, "       NOHARDENBO +\n" );
   fputs( ParmBuffer, fp );
} /* endif */

if ( DefnLQ.Usage == MQUS_NORMAL ) {
   sprintf( ParmBuffer, "       USAGE(NORMAL) +\n" );
   fputs( ParmBuffer, fp );
} else {
   sprintf( ParmBuffer, "       USAGE(XMIT) +\n" );
   fputs( ParmBuffer, fp );
} /* endif */

if ( DefnLQ.TriggerControl == MQTC_OFF ) {
   sprintf( ParmBuffer, "       NOTRIGGER +\n" );
   fputs( ParmBuffer, fp );
```

```
} else {
   sprintf( ParmBuffer, "       TRIGGER +\n" );
   fputs( ParmBuffer, fp );
} /* endif */

switch ( DefnLQ.TriggerType ) {
case MQTT_NONE:
   sprintf( ParmBuffer, "       TRIGTYPE(NONE) +\n" );
   fputs( ParmBuffer, fp );
   break;
case MQTT_FIRST:
   sprintf( ParmBuffer, "       TRIGTYPE(FIRST) +\n" );
   fputs( ParmBuffer, fp );
   break;
case MQTT_EVERY:
   sprintf( ParmBuffer, "       TRIGTYPE(EVERY) +\n" );
   fputs( ParmBuffer, fp );
   break;
case MQTT_DEPTH:
   sprintf( ParmBuffer, "       TRIGTYPE(DEPTH) +\n" );
   fputs( ParmBuffer, fp );
   break;
} /* endswitch */

sprintf( ParmBuffer, "       TRIGDPTH(%d) +\n", DefnLQ.TriggerDepth );
fputs( ParmBuffer, fp );

sprintf( ParmBuffer, "       TRIGMPRI(%d) +\n", DefnLQ.TriggerMsgPriority);
fputs( ParmBuffer, fp );

sprintf( ParmBuffer, "       TRIGDATA('%s') +\n", DefnLQ.TriggerData );
fputs( ParmBuffer, fp );

sprintf( ParmBuffer, "       PROCESS('%s') +\n", DefnLQ.ProcessName );
fputs( ParmBuffer, fp );

sprintf( ParmBuffer, "       INITQ('%s') +\n", DefnLQ.InitiationQName );
fputs( ParmBuffer, fp );


sprintf( ParmBuffer, "       RETINTVL(%d) +\n", DefnLQ.RetentionInterval );
fputs( ParmBuffer, fp );

sprintf( ParmBuffer, "       BOTHRESH(%d) +\n", DefnLQ.BackoutThreshold );
fputs( ParmBuffer, fp );

sprintf( ParmBuffer, "       BOQNAME('%s') +\n", DefnLQ.BackoutReqQName );
fputs( ParmBuffer, fp );

if ( DefnLQ.Scope == MQSCO_Q_MGR ) {
   sprintf( ParmBuffer, "       SCOPE(QMGR) +\n" );
   fputs( ParmBuffer, fp );
} else {
   sprintf( ParmBuffer, "       SCOPE(CELL) +\n" );
   fputs( ParmBuffer, fp );
} /* endif */

sprintf( ParmBuffer, "       QDEPTHHI(%d) +\n", DefnLQ.QDepthHighLimit );
fputs( ParmBuffer, fp );

sprintf( ParmBuffer, "       QDEPTHLO(%d) +\n", DefnLQ.QDepthLowLimit );
fputs( ParmBuffer, fp );

if ( DefnLQ.QDepthMaxEvent == MQEVR_ENABLED ) {
   sprintf( ParmBuffer, "       QDPMAXEV(ENABLED) +\n" );
   fputs( ParmBuffer, fp );
} else {
   sprintf( ParmBuffer, "       QDPMAXEV(DISABLED) +\n" );
   fputs( ParmBuffer, fp );
} /* endif */

if ( DefnLQ.QDepthHighEvent == MQEVR_ENABLED ) {
```

```
      sprintf( ParmBuffer, "       QDPHIEV(ENABLED) +\n" );
      fputs( ParmBuffer, fp );
   } else {
      sprintf( ParmBuffer, "       QDPHIEV(DISABLED) +\n" );
      fputs( ParmBuffer, fp );
   } /* endif */

   if ( DefnLQ.QDepthLowEvent == MQEVR_ENABLED ) {
      sprintf( ParmBuffer, "       QDPLOEV(ENABLED) +\n" );
      fputs( ParmBuffer, fp );
   } else {
      sprintf( ParmBuffer, "       QDPLOEV(DISABLED) +\n" );
      fputs( ParmBuffer, fp );
   } /* endif */

   sprintf( ParmBuffer, "       QSVCINT(%d) +\n", DefnLQ.QServiceInterval );
   fputs( ParmBuffer, fp );

   switch ( DefnLQ.QServiceIntervalEvent ) {
   case MQQSIE_OK:
      sprintf( ParmBuffer, "       QSVCIEV(OK)\n" );
      fputs( ParmBuffer, fp );
      break;
   case MQQSIE_NONE:
      sprintf( ParmBuffer, "       QSVCIEV(NONE)\n" );
      fputs( ParmBuffer, fp );
      break;
   case MQQSIE_HIGH:
      sprintf( ParmBuffer, "       QSVCIEV(HIGH)\n" );
      fputs( ParmBuffer, fp );
      break;
   } /* endswitch */

   sprintf( ParmBuffer, "\n" );
   fputs( ParmBuffer, fp );

   fclose(fp);

}

/* ------------------------------------------------------------------------ */
/*                                                                          */
/* The queue manager returns strings of the maximum length for each         */
/* specific parameter, padded with blanks.                                  */
/*                                                                          */
/* We are interested in only the non-blank characters so will extract them  */
/* from the message buffer, and terminate the string with a null, \0.       */
/*                                                                          */
/* ------------------------------------------------------------------------ */
void MQParmCpy( char *target, char *source, int length )
{
   int   counter=0;

   while ( counter < length && source[counter] != ' ' ) {
      target[counter] = source[counter];
      counter++;
   } /* endwhile */

   if ( counter < length) {
      target‡counter“ = '\0';
   } /* endif */
}
```

Part 3. Installable services

Chapter 11. Installable services and components

This chapter introduces the installable services and the functions and components associated with them. The interface to these functions is documented so that you can supply components, or so that components can be provided by software vendors. This interface is described in Chapter 15, “Installable services interface” on page 395. Installable services and components are supported by:

- MQSeries for AIX
- MQSeries for AT&T GIS UNIX
- MQSeries for Digital OpenVMS
- MQSeries for HP-UX
- MQSeries for OS/2 Warp
- MQSeries for SINIX and DC/OSx
- MQSeries for SunOS
- MQSeries for Sun Solaris
- MQSeries for Tandem NonStop Kernel
- MQSeries for Windows NT

Why installable services?

The major reasons for providing MQSeries installable services are:

- To provide you with the flexibility of choosing whether to use components provided by MQSeries products, or replace, or augment, them with others.
- To allow vendors to participate, by providing components that may be using new technologies, without making internal changes to MQSeries products.
- To allow MQSeries to exploit new technologies faster and cheaper, and so provide products earlier and at lower prices.

Installable services and *service components* are part of the MQSeries product structure. At the center of this structure is the part of the queue manager that implements the function and rules associated with the Message Queue Interface (MQI). This central part requires a number of service functions, called *installable services*, in order to perform its work. The following installable services are defined:

- Authorization service
- Name service
- User identifier service (MQSeries for OS/2 Warp only)

Each installable service is a related set of functions that are implemented using one or more *service components*. Each component is invoked using a properly architected, publicly available interface. This enables independent software vendors and other third parties to provide installable components to augment or replace those provided by the MQSeries products. Table 23 on page 368 summarizes the services and components that can be used on the various platforms.

Table 23. Installable services and components summary			
Platform	**Supplied component**	**Function**	**Requirements**
Authorization Service			
Digital OpenVMS, Tandem NSK, UNIX systems, and Windows NT	Object Authority Manager (OAM)	Provides authorization checking on commands and MQI calls. Users can write their component to augment or replace the OAM.	(Appropriate platform authorization facilities are assumed)
OS/2	None	User defined	A third-party or user-written authority manager
Name Service			
AIX, Digital OpenVMS, HP-UX, OS/2, Sun Solaris, and Windows NT	DCE name service component	Allows queue managers to share queues or: User defined. Shared queues must have their *Scope* attribute set to CELL.	DCE is required for the supplied component. or: A third-party or user-written name manager.
AT&T GIS UNIX, SINIX and DC/OSx, and SunOS	None	User defined. Shared queues must have their *Scope* attribute set to CELL.	A third-party or user-written name manager.
User Identifier Service			
UNIX systems, Digital OpenVMS, TandemNSK, and Windows NT	None	Because these systems provide the required user IDs, this service is not required.	–
OS/2	DLL and (modified) sample source	Provides a user ID that is automatically inserted into an MQSeries message	None—uses OS/2 environment variables

Functions and components

Each service consists of a set of related functions. For example, the name service contains function for:

- Looking up a queue name, returning the name of the queue manager at which the queue is defined
- Inserting a queue name into the service's directory
- Deleting a queue name from the service's directory

It also contains an initialization function and a termination function.

An installable service is provided by one or more service components. Each component is capable of performing some or all of the functions that are defined for that service. For example, in MQSeries for AIX, the supplied authorization service component, the OAM, performs all seven of the available functions. See "Authorization service interface" on page 380 for more information. The component is also responsible for managing any underlying resources or software (for example, DCE name services) that it needs to implement the service.

Configuration files provide a standard method for loading the component, and determining the addresses of the functional routines that it provides.

Figure 10 shows how services and components are related:

- A service is defined to a queue manager by stanzas in a configuration file.
- Each service is supported by supplied code in the queue manager. Users cannot change this code and therefore cannot create their own services.
- Each service is implemented by one or more components; these may be supplied with the product or user-written. Multiple components for a service can be invoked, each supporting different facilities within the service.
- Entry points 'connect' the service components to the supporting code in the queue manager.

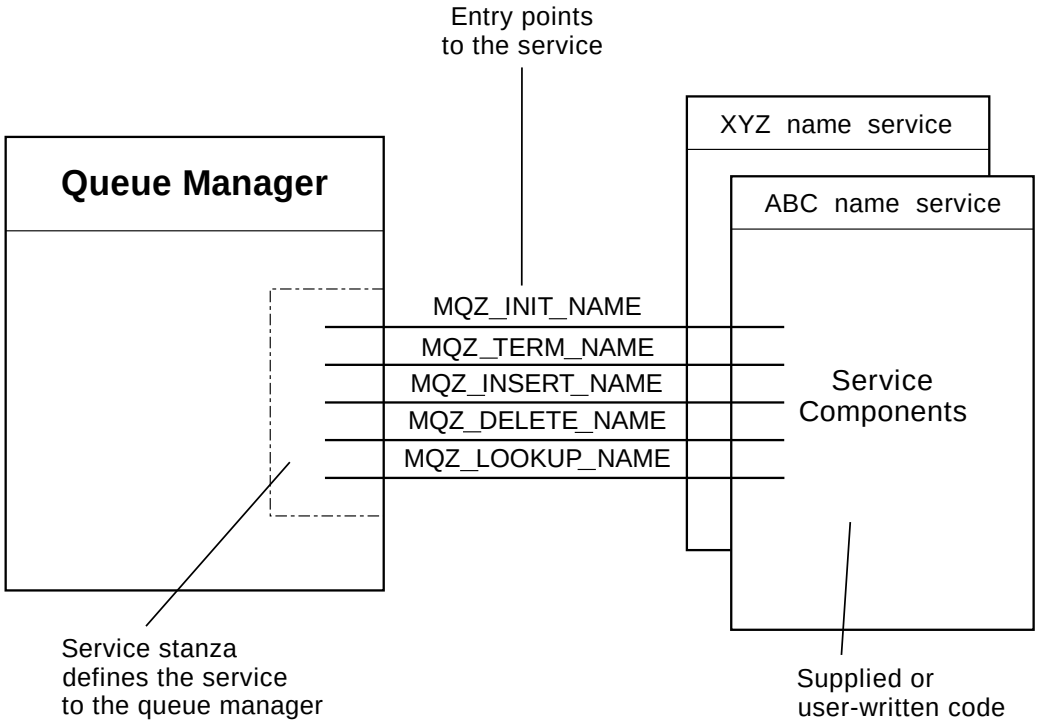


Figure 10. Understanding services, components, and entry points

Entry-points

Each service component is represented by a list of the entry-point addresses of the routines that support a particular installable service. The installable service defines the function to be performed by each routine.

The ordering of the service components when they are configured defines the order in which entry-points are called in an attempt to satisfy a request for the service.

In the supplied header file `cmqzc.h`, the supplied entry points to each service have an MQZID_ prefix.

Return codes

Service components provide return codes to the queue manager to report on a variety of conditions. They report the success or failure of the operation, and indicate whether or not the queue manager is to proceed to the next service component, or whether the queue manager itself should make that decision. A separate *Continuation* parameter carries this indication.

Component data

A single service component may require data to be shared between its various functions. Installable services provide an optional data area to be passed on each invocation of a given service component. This data area is for the exclusive use of the service component. It is shared by all the invocations of a given function, even if they are made from different address spaces or processes. It is guaranteed to be addressable from the service component whenever it is called. You must declare the size of this area in the *ServiceComponent* stanza.

Initialization

When the component initialization routine is invoked, it must call the queue manager **MQZEP** function for each entry-point supported by the component. **MQZEP** defines an entry-point to the service. All the undefined exit points are assumed to be NULL.

Primary initialization

A component is always invoked with this option once, before it is invoked in any other way.

Secondary initialization

A component may be invoked with this option, on certain platforms. For example, it may be invoked once for each operating system process, thread or task by which the service is accessed.

If secondary initialization is used:

- The component may be invoked more than once for secondary initialization. For each such call, a matching call for secondary termination is issued when the service is no longer needed.

  For naming services this is the MQZ_TERM_NAME call.

  For authorization services this is the MQZ_TERM_AUTHORITY call.
- The entry points must be respecified (by calling MQZEP) each time the component is called for primary and secondary initialization.
- Only one copy of component data is used for the component; there is not a different copy for each secondary initialization.
- The component is not invoked for any other calls to the service (from the operating system process, thread or task, as appropriate) before secondary initialization has been carried out.
- The *Version* parameter must be set by the component to the same value for primary and secondary initialization.

Secondary termination

A component is invoked with this option, if it has been invoked for secondary initialization.

Primary termination

A component is always invoked with this option once, when it is no longer required. No further calls are made to this component.

Configuring services and components

Service components are configured using the queue manager configuration files.

The configuration information for a queue manager is included in the queue manager configuration file. For each service to be used, this file must contain a *Service* stanza, which defines the service to the queue manager. For each component within a service, there must be a *ServiceComponent* stanza. This identifies the name and path of the module containing the code for that component.

The authorization service component for MQSeries for AIX, known as the Object Authority Manager (OAM), is supplied with the product. When you create a queue manager, the queue manager configuration file is automatically updated to include the appropriate stanzas for the authorization service and for the default component (the OAM). For the other components, you must configure the queue manager configuration file manually.

The code for each service component is loaded into the queue manager when the queue manager is started, using dynamic binding, where this is supported on the platform.

Service stanza format

The format of the *Service* stanza is:

```
Service:
  Name=<service_name>
  EntryPoints=<entries>
```

where:

`<service_name>` The name of the service. This is defined by the service.

`<entries>` The number of entry-points defined for the service. This includes the initialization and termination entry points.

Service component stanza format

The format of the *Service component* stanza is:

```
ServiceComponent:
  Service=<service_name>
  Name=<component_name>
  Module=<module_name>
  ComponentDataSize=<size>
```

where:

`<service_name>` The name of the service. This must match the `Name` specified in a service stanza.

`<component_name>` A descriptive name of the service component. This must be unique, and contain only the characters that are valid for the names of MQSeries objects (for example, queue names). This name occurs in operator messages generated by the service. It is recommended. therefore, that the name starts with a company trademark or similar distinguishing string.

`<module_name>` The name of the module to contain the code for this component.

Note: Specify a full path name.

`<size>` The size in bytes of the component data area passed to the component on each call. Specify zero if no component data is required.

These two stanzas can appear in any order and the stanza keys under them can also appear in any order. For either of these stanzas, all the stanza keys must be present. If a stanza key is duplicated, the last one is used.

At startup time, the queue manager processes each service component entry in the configuration file in turn. It attempts to load the specified component module. If successful, it invokes the entry-point of the component (which must be the entry-point for initialization of the component), passing it a configuration handle.

Creating your own service component

To create your own service component:

- Write the component using the sample provided (see page 391). You must ensure that the header file `cmqzc.h` is included in your program.
- **For UNIX systems**: Create the shared library by compiling the program, and linking it with the shared libraries `libmqm*` and `libmqmzf*` (the threading suffixes and file extensions vary by platform).

  Note: On Version 2.2.1 of the MQSeries products for UNIX systems, the agent typically runs in an unthreaded environment.

  On Version 5 of the MQSeries products for UNIX systems, the agent runs in a threaded environment and the OAM and Name Service ***must*** be built to run in a threaded environment.

  This includes using the threaded versions of `libmqm` and `libmqmzf`.

  Threaded Sun Solaris installable services

  If the MQ DCE option is ***not*** currently installed, threaded installable services on Sun Solaris ***must*** be threaded with Posix** V10 threading.

  If the MQ DCE option ***is*** currently installed, threaded installable services on Sun Solaris ***must*** be threaded with DCE threading.

- **For OS/2 and Windows NT**: Create a DLL by compiling the program, and linking it with the libraries MQM.LIB and MQMZF.LIB.

See the section on data-conversion exits in the *MQSeries Application Programming Guide* for details of how to compile and link code for shared libraries.

- Add stanzas to the queue manager configuration file to define the service to the queue manager and to specify the location of the module. Refer to the individual chapters for each service, for more information.
- Stop and restart the queue manager to activate the component.

Using multiple service components

This section may be omitted at first reading.

You can install more than one component for a given service. This allows components to provide only partial implementations of the service, and to rely on other components to provide the remaining functions.

For example, suppose you create a new name service component called `XYZ_name_serv`. This component supports looking up a queue name, but does not support inserting a name in or deleting a name from the service directory.

What the component does

This component uses a simple algorithm that returns a fixed queue-manager name for any queue name with which it is invoked. This component does not hold a database of queue names, and therefore does not support the insert and delete functions.

How the component is used

The component `XYZ_name_serv` is then installed on the same queue manager as the MQSeries DCE-based name services component. The *`ServiceComponent`* stanzas are ordered so that the DCE-based component is invoked first. Any calls to insert or delete a queue in a component directory are handled by the DCE-based component—it is the only one that implements these functions. However, a lookup call—which the DCE-based component is unable to resolve—is passed on to the lookup-only component, `XYZ_name_serv`. This component supplies a queue-manager name from its simple algorithm.

Omitting entry points

You can design a service component not to implement some functions. The installable services framework places no restrictions on which functions may be omitted. However, for specific installable services, omission of one or more functions might be logically inconsistent with the purpose of the service.

Example of entry points

Table 24 on page 374 shows an example of the installable name service for which the two components have been installed. Each supports a different set of functions associated with this particular installable service. For insert function, the ABC component entry-point is invoked first. Entry points that have not been defined to the service (using **MQZEP**) are assumed to be NULL. An entry-point for initialization is provided in the table, but this is not required because initialization is carried out by the main entry-point of the component.

When the queue manager has to use an installable service, it uses the entry-points defined for that service (the columns in Table 24 on page 374). Taking each component in turn, the queue manager determines the address of the routine that implements the required function. It then calls the routine, if it exists. If the operation is successful, any results and status information are used by the queue manager.

Table 24. Example of entry-points for an installable service

Function number	**ABC name service component**	**XYZ name service component**
MQZID_INIT_NAME (Initialize)	ABC_initialize()	XYZ_initialize()
MQZID_TERM_NAME (Terminate)	ABC_terminate()	XYZ_terminate()
MQZID_INSERT_NAME (Insert)	ABC_Insert()	NULL
MQZID_DELETE_NAME (Delete)	ABC_Delete()	NULL
MQZID_LOOKUP_NAME (Lookup)	NULL	XYZ_Lookup()

If the routine does not exist, the queue manager may repeat this process for the next component in the list. In addition, if the routine does exist but returns a code indicating that it could not perform the operation, the attempt continues with the next available component. Routines in service components may return a code that indicates that no further attempts to perform the operation should be made.

Chapter 12. Authorization service

The authorization service is an installable service that is available on:

MQSeries for AIX
MQSeries for AT&T GIS UNIX
MQSeries for Digital OpenVMS
MQSeries for HP-UX
MQSeries for OS/2 Warp
MQSeries for SINIX and DC/OSx
MQSeries for SunOS
MQSeries for Sun Solaris
MQSeries for Tandem NonStop Kernel
MQSeries for Windows NT

This service enables queue managers to invoke authorization facilities, for example, checking that a user ID has authority to open a queue.

The authorization service is a component of the MQSeries security enabling interface (SEI), which is part of the MQSeries framework.

Object authority manager (OAM)

The authorization service component supplied with the MQSeries products is called the Object Authority Manager (OAM). By default, the OAM is active and works with the control commands **dspmqaut** (display authority) and **setmqaut** (set/reset authority).

The syntax of these commands and how to use them are described in detail in the following manuals:

- For UNIX systems and Windows NT – *MQSeries System Administration*.
- For Digital OpenVMS – the *MQSeries for Digital OpenVMS System Management Guide*.
- For Tandem NSK – the *MQSeries for Tandem NonStop Kernel System Management Guide*.

The OAM works with the *entity* of a principal or group, except on Tandem NSK which works only at the group level. These entities vary from platform to platform.

When an MQI request is made or a command is issued, the OAM checks the authorization of the entity associated with the operation to see whether it can:

- Perform the requested operation.
- Access the specified queue manager resources.

The authorization service enables you to augment or replace the authority checking provided for queue managers by writing your own authorization service component.

Defining the service to the operating system

The authorization service stanzas in the the queue manager configuration file `qm.ini` define the authorization service to the queue manager. See “Configuring services and components” on page 371 for information about the types of stanza.

Authorization service on UNIX systems

On these platforms:

Principal
Is a UNIX system user ID, or an ID associated with an application program running on behalf of a user.

Group
Is a UNIX system-defined collection of principals.

Authorizations can be granted or revoked at the group level.

Configuring authorization service stanzas: UNIX systems

On MQSeries for UNIX systems, each queue manager has its own queue manager configuration file. For example, on AIX, the default path and file name of the queue manager configuration file for queue manager QMNAME is `/var/mqm/qmgrs/QMNAME/qm.ini`.

The *Service* stanza and the *ServiceComponent* stanza for the default authorization component are added to `qm.ini` automatically, but can be overridden through the use of `mqsnoaut`. Any other *ServiceComponent* stanzas must be added manually.

For example, the following stanzas in the queue manager configuration file define two authorization service components on MQSeries for AIX:

```
Service:
   Name=AuthorizationService
   EntryPoints=7

ServiceComponent:
   Service=AuthorizationService
   Name=MQSeries.UNIX.authorization.service
   Module=/usr/lpp/mqm/lib/amqzfu
   ComponentDataSize=0

ServiceComponent:
   Service=AuthorizationService
   Name=user.defined.authorization.service
   Module=/usr/bin/udas01
   ComponentDataSize=96
```

Figure 11. Authorization service stanzas in qm.ini

The first service component stanza, `MQSeries.UNIX.authorization.service` defines the default authorization service component, the OAM. If you remove this stanza and restart the queue manager, the OAM is disabled and no authorization checks are made.

In the second (user-defined) service component stanza, `/usr/bin/udas01` is the path and file name of the code module for the user-defined component. For the user-defined service, the size of the data area that the component requires is specified as 96 bytes.

You must add the second service component stanza manually to the configuration file **before** you start the queue manager. The configuration file is read when the queue manager is started, therefore, if you change a stanza, the changes can only take effect when the queue manager is restarted.

Authorization service on Windows NT

On this platform:

Principal
Is a Windows NT user ID, or an ID associated with an application program running on behalf of a user.

Group
Is a Windows NT group.

Authorizations can be granted or revoked at the principal or group level.

Configuring authorization service stanzas: Windows NT

On MQSeries for Windows NT each queue manager has its own queue manager configuration file. The default path and file name of the queue manager configuration file for queue manager QMNAME is `C:\MQM\QMGRS\QMNAME\qm.ini`.

The *Service* stanza and the *ServiceComponent* stanza for the default authorization component are added to `qm.ini` automatically, but can be overridden through the use of `mqsnoaut`. Any other *ServiceComponent* stanzas must be added manually.

```
Service:
   Name=AuthorizationService
   EntryPoints=9

ServiceComponent:
   Service=AuthorizationService
   Name=MQSeries.WindowsNT.auth.service
   Module=C:\MQM\BIN\AMQZFU.DLL
   ComponentDataSize=0
```

Figure 12. Authorization service stanzas (Windows NT)

The service component stanza, `MQSeries.WindowsNT.auth.service` defines the default authorization service component, the OAM. If you remove this stanza and restart the queue manager, the OAM is disabled and no authorization checks are made.

Authorization service on MQSeries for OS/2 Warp

On MQSeries for OS/2, no authorization service component is supplied with the product. However, the facilities are there if you want to do this for yourself by writing your own authorization service component.

If you write your own authorization component, you must define what your component does, and implement it using the interface provided.

Configuring authorization service stanzas: OS/2

On MQSeries for OS/2 each queue manager has its own queue manager configuration file. For example, the default path and file name of the queue manager configuration file for queue manager QMNAME is `C:\MQM\QMGRS\QMNAME\qm.ini`.

By default, the *Service* and *ServiceComponent* stanzas for the authorization service are not present in `qm.ini`.

To implement a user-written service, you must add these stanzas manually. For example, the following stanzas in the queue manager configuration file define an authorization service component on MQSeries for OS/2:

```
Service:
   Name=AuthorizationService
   EntryPoints=7

ServiceComponent:
   Service=AuthorizationService
   Name=user.defined.authorization.service
   Module=C:\MQM\DLL\UDAS01.DLL
   ComponentDataSize=128
```

Figure 13. Authorization service stanzas in qm.ini (OS/2)

These stanzas must be defined in `qm.ini` **before** you start the queue manager. The configuration file is read when the queue manager is started, therefore, if you change a stanza, the changes can only take effect when the queue manager is restarted.

Authorization service on Digital OpenVMS

On this platform:

Principal
Is a Digital OpenVMS system user ID, or an ID associated with an application program running on behalf of a user.

Group
Is a Digital OpenVMS system-defined collection of principals.

Authorizations can be granted or revoked at the group level.

Configuring authorization service stanzas: Digital OpenVMS

On MQSeries for Digital OpenVMS each queue manager has its own queue manager configuration file. The default path and file name of the queue manager configuration file for queue manager QMNAME is `MQS_ROOT:[MQM.QMGRS.QMNAME]QM.INI`.

The *Service* stanza and the *ServiceComponent* stanza for the default authorization component are added to `qm.ini` automatically, but can be overridden through the use of `mqsnoaut`. Any other *ServiceComponent* stanzas must be added manually.

```
Service:
   Name=AuthorizationService
   EntryPoints=9

ServiceComponent:
   Service=AuthorizationService
   Name=MQSeries.UNIX.Auth.Service
   Module=amqzfu
   ComponentDataSize=1024
```

Figure 14. Authorization service stanzas (Digital OpenVMS)

The service component stanza, `MQSeries.UNIX.auth.service` defines the default authorization service component, the OAM. If you remove this stanza and restart the queue manager, the OAM is disabled and no authorization checks are made.

Authorization service on Tandem NSK

On this platform:

Group
Is a Tandem NSK system-defined group.

Authorizations can be granted or revoked at the group level.

Configuring authorization service stanzas: Tandem NSK

On MQSeries for Tandem NSK each queue manager has its own queue manager configuration file. The path and file name of the queue manager configuration file depends on the name of the queue manager. You should look in the `mqs.ini` file to find the name of the file. For example, the sub-volume of queue manager `QMQM` is `QMQMD`.

The *Service* stanza and the *ServiceComponent* stanza for the default authorization component are added to `qm.ini` automatically, but can be overridden through the use of `mqsnoaut`. Any other *ServiceComponent* stanzas must be added manually.

```
Service:
   Service=AuthorizationService
   EntryPoints=9

ServiceComponent:
   Service=AuthorizationService
   Name=MQSeries.TANDEM.auth.service
   Module=MQOAM
   ComponentDataSize=0
   ComponentID=0
```

Figure 15. Authorization service stanzas (Tandem NSK)

The service component stanza, `MQSeries.TANDEM.auth.service` defines the default authorization service component, the OAM. If you remove this stanza and restart the queue manager, the OAM is disabled and no authorization checks are made.

Authorization service interface

The authorization service provides the following entry points for use by the queue manager:

MQZ_INIT_AUTHORITY
Initialize authorization service component.

MQZ_TERM_AUTHORITY
Terminate authorization service component.

MQZ_CHECK_AUTHORITY
Checks whether an entity has authority to perform one or more operations on a specified object.

MQZ_SET_AUTHORITY
Sets the authority that an entity has to a specified object.

MQZ_GET_AUTHORITY
Gets the authority that an entity has to access a specified object.

MQZ_GET_EXPLICIT_AUTHORITY
Gets either the authority that a named group has to access a specified object (but without the additional authority of the **nobody** group) or the authority that the primary group of the named principal has to access a specified object.

MQZ_COPY_ALL_AUTHORITY
Copy all the current authorizations that exist for a referenced object to another object.

MQZ_DELETE_AUTHORITY
Deletes all authorizations associated with a specified object.

These names are defined as **typedef**s, in the header file `cmqzc.h`, which can be used to prototype the component functions.

The initialization function (**MQZ_INIT_AUTHORITY**) must be the main entry point for the component. The other functions are invoked through the entry point address that the initialization function has added into the component entry point vector.

See “Creating your own service component” on page 372 for more information. The supplied sample, “Sample program for user identifier service” on page 391, shows how to write a program for an installable service. Use this example as a basis for your own programs, bearing in mind that this sample is written for OS/2.

Chapter 13. Name service

The MQSeries name service provides support to the queue manager for looking up the name of the queue manager that owns a specified queue. No other queue attributes can be retrieved from a name service.

The name service is an installable service, which enables an application to open remote queues for output as if they were local queues. A name service is not invoked for objects other than queues.

Note: The remote queues ***must*** have their *Scope* attribute set to CELL.

The name service is available on:

- MQSeries for AIX
- MQSeries for AT&T GIS UNIX
- MQSeries for Digital OpenVMS
- MQSeries for HP-UX
- MQSeries for OS/2 Warp
- MQSeries for SINIX and DC/OSx
- MQSeries for SunOS
- MQSeries for Sun Solaris
- MQSeries for Windows NT

When an application opens a queue, the name of the queue is first looked for in the queue manager's directory. If it is not found there, it is then looked for in as many name services as have been configured, until one is found that recognizes the queue name. If none recognizes the name, the open fails.

The name service returns the owning queue manager for that queue. The queue manager then continues with the MQOPEN request as if the command had specified the queue and queue manager name in the original request.

The name service interface (NSI) is part of the MQSeries framework.

How the name service works

If a queue definition specifies the *Scope* attribute as queue manager (SCOPE(QMGR) in MQSC) the queue definition (along with all the queue attributes) is stored in the queue manager's directory only; this cannot be replaced by an installable service.

If a queue definition specifies the *Scope* attribute as cell (SCOPE(CELL) in MQSC) the queue definition is also stored in the queue manager's directory, along with all the queue attributes. However, the queue and queue-manager name are also stored in a name service. If no service is available that can store this information, a queue with the *Scope* cell cannot be defined.

The directory in which the information is stored may be managed by the service, or the service may use an underlying service (such as a DCE directory) for this purpose. In either case, however, definitions stored in the directory must persist, even after the component and queue manager have terminated, until they are explicitly deleted.

Notes:

1. You do need to define the channel to send a message to a remote host's local queue definition (with a scope of CELL) on a different queue manager within a naming directory cell.
2. You cannot get messages directly from the remote queue, even when it has a scope of CELL.
3. No remote queue definition is required when sending to a queue with a scope of CELL.

   The point of the naming service is that the destination queue is defined centrally, although you still need a transmission queue to the destination queue manager.

   The transmission queue on the local system must have the same name as the queue manager owning the target queue, with the scope of cell, on the remote system.

   For example, if the remote queue manager has the name QM01, then the transmission queue on the local system must also have the name QM01. See the section on "Queue name resolution" in the *MQSeries Intercommunication* book for further information.

Name service interface

A name service provides the following entry points for use by the queue manager:

MQZ_INIT_NAME	Initialize the name service component.
MQZ_TERM_NAME	Terminate the name service component.
MQZ_LOOKUP_NAME	Look up the queue-manager name for the specified queue.
MQZ_INSERT_NAME	Insert an entry containing the owning queue-manager name for the specified queue into the directory used by the service.
MQZ_DELETE_NAME	Delete the entry for the specified queue from the directory used by the service.

If there is more than one name service configured:

- For lookup, the MQZ_LOOKUP_NAME function is invoked for each service in the list until the queue name is resolved (unless any component indicates that the search should stop).
- For insert, the MQZ_INSERT_NAME function is invoked for the first service in the list that supports this function.
- For delete, the MQZ_DELETE_NAME function is invoked for the first service in the list that supports this function.

It is not therefore useful to have more than one component that supports the insert and delete functions. However, a component that only supports lookup is feasible, and could be used, for example, as the last component in the list to resolve any name, that is not known by any other name service component, to a queue manager at which the name may be defined.

In the C programming language the names are defined as function datatypes using the typedef statement. These can be used to prototype the service functions, to ensure that the parameters are correct.

The header file that contains all the material specific to installable services is `cmqzc.h` for the C language.

Apart from the initialization function (MQZ_INIT_NAME) – which must be the component's main entry point – functions are invoked by the entry point address that the initialization function has added, using the MQZEP call.

The following examples of configuration file stanzas for the name service specify a name service component provided by the (fictitious) ABC company.

```
# Stanza for name service
 Service:
    Name=NameService
    EntryPoints=5

# Stanza for name service component, provided by ABC
 ServiceComponent:
    Service=NameService
    Name=ABC.Name.Service
    Module=disk:[dir.dir]abcname
    ComponentDataSize=1024
```

Figure 16. Name service stanzas in qm.ini (for Digital OpenVMS)

```
# Stanza for name service
 Service:
    Name=NameService
    EntryPoints=5

# Stanza for name service component, provided by ABC
 ServiceComponent:
    Service=NameService
    Name=ABC.Name.Service
    Module=C:\MQM\DLL\ABCNAME.DLL
    ComponentDataSize=1024
```

Figure 17. Name service stanzas in qm.ini (for OS/2)

```
Service:
   Name=NameService
   EntryPoints=5

ServiceComponent:
   Service=NameService
   Name=MQSeries.DCE.name.service
   Module=C:\MQM\BIN\AMQNFA
   ComponentDataSize=1024
```

Figure 18. Name service stanzas in qm.ini (for Windows NT)

```
# Stanza for name service
 Service:
   Name=NameService
   EntryPoints=5

# Stanza for name service component, provided by ABC
 ServiceComponent:
   Service=NameService
   Name=ABC.Name.Service
   Module=/usr/lib/abcname
   ComponentDataSize=1024
```

Figure 19. Name service stanzas in qm.ini (for UNIX systems)

Using DCE to share queues on different queue managers

IBM supplies an implementation of a name service that uses DCE (Distributed Computing Environment), although you are free to write your own component that does not use DCE.

To use the supplied name service component, you must define the name service and its installed component to the queue manager. You do this by inserting the appropriate stanza in the queue manager configuration file (`qm.ini`) file. See the *MQSeries System Administration* book for details. You must also do some DCE configuration.

If your queue managers are located on nodes within a Distributed Computing Environment (DCE) cell, you can configure them to share queues. Applications can then connect to one queue manager and open a queue for output on *another* queue manager on another node.

Normally the queue manager rejects open requests from a local application if the queue is not defined on that queue manager. However, when DCE names is in use the remote queue does not need to be defined on the local queue manager. Also if an appropriate set of transmission queues are defined the queue may be moved between remote queue managers within the DCE cell without any changes being required to the local definitions.

Configuration tasks for shared queues

This section describes how you set up shared queues on queue managers that reside on nodes that are within the DCE cell.

For each queue manager:

1. Use the **endmqm** command to stop the queue manager if it is running.
2. Configure the name service by adding the required name service stanza to the queue manager configuration file. The contents of this stanza are described in the *MQSeries System Administration* book. To invoke the name service, you have to restart the queue manager.
3. Use the **strmqm** command to restart the queue manager.
4. Set up channels for messaging between queue managers; see the *MQSeries Intercommunication* book for further details.

For any queue that you want to be shared, specify the SCOPE attribute as CELL. For example, use these MQSC commands:

```
DEFINE QLOCAL (GREY.PUBLIC.QUEUE) SCOPE(CELL)
or
ALTER QLOCAL (PINK.LOCAL.QUEUE) SCOPE(CELL)
```

The queue created or altered must belong to a queue manager on a node within the DCE cell.

DCE configuration

To use the supplied name service component, you must have the OSF Distributed Computing Environment (DCE) Directory Service configured. This service enables applications that connect to one queue manager to open queues that belong to another queue manager in the same DCE cell.

The name service stanzas in the queue manager configuration file `qm.ini` define the name service to the queue manager.

Scripts are supplied that set up the DCE keytables, principal, and directory entries so that the supplied name service can run:

`dcesetsv`
: Sets up a principal, and directory entries, on the DCE cell servers. It can be run from any host in the cell, and it is run ***once*** for the entire DCE cell

`dcesetkt`
: Sets up a system keytable on the local host. It ***must*** be run on every host in the DCE cell on which the MQSeries DCE Naming Service is run.

These scripts are placed in the following directories:

For OS/2 and Windows NT

`C:\MQM\TOOLS\DCE\SAMPLES`, where `C:\` is the installation drive.

For UNIX systems

`/mqmtop/samp`

where `mqmtop` is:

- `/usr/lpp/mqm` on AIX
- `/opt/mqm` on AT&T GIS UNIX, HP-UX, SINIX and DC/OSx, Sun OS, and Sun Solaris.

For Digital OpenVMS

`mqs_examples;dcesetup.com`

Note to users

You need to install the MQ DCE option if you are using AIX or Sun Solaris.

Chapter 14. User identifier service

This service is available only on MQSeries for OS/2 Warp.

The *user identifier service* enables queue managers running under OS/2 to obtain a user-defined user ID.

By default, the user ID associated with MQI applications running under OS/2 is `OS2`.

This user ID is used by the local queue manager when an application issues an MQCONN request. That is, the queue manager inserts this user ID into the context fields of any messages that are sent by the application.

The user identification service enables a user-defined user ID to be substituted in place of the supplied one. The mechanism for doing this must be defined by the user.

Defining the service to OS/2

The user identifier service stanzas in the queue manager configuration file `qm.ini` defines the User Identifier service to the queue manager. By default, this file is located in `C:\mqm\qmgrs\QMNAME\qm.ini`. You must add these stanzas manually to the configuration file **before** you start the queue manager.

For example, the following queue manager stanza defines the supplied user identifier service component.

```
Service:
   Name=UserIdentifierService
   EntryPoints=3

ServiceComponent:
   Service=UserIdentifierService
   Name=MQSeries.environment.UserID.Service
   Module=C:\MQM\DLL\AMQZFCB2.DLL
   ComponentDataSize=24
```

For the example shown in “Sample program for user identifier service” on page 391, the following assignments must be made:

`Name=UserIdentifierService`
: Specifies the type of service as a user identifier service.

`EntryPoints=3`
: There are three entry points in the service. See “MQZEP – Add component entry point” on page 396 for more information.

`Service=UserIdentifierService`
: Specifies that this component belongs to the user identifier service. The name must match the name in the service stanza.

`Name=MQSeries.environment.UserID.Service`
: The name of the service componenent. This name identifies the component in any messages that are generated from it.

`Module=C:\mqm\dll\AMQZFCB2.DLL`
: The path and name of the DLL containing the component functions.

`ComponentDataSize=24`
: The size of the data to be passed between the module and the queue manager. In this case, 24 bytes, 12 each for the user ID and the (optional) password respectively.

The queue manager configuration file—and therefore the user identifier service stanza—is read when the queue manager is started. Therefore, if you change the stanza, the changes can only take effect when the queue manager is restarted.

User identifier service interface

The user identifier service provides the following entry points for use by the queue manager:

MQZ_INIT_USERID Initialize user identifier service. This routine must be called before any others.

MQZ_FIND_USERID Find user ID.

MQZ_TERM_USERID Terminate user identifier service.

These names are defined as **typedef**s in `cmqzc.h`, which can be used to prototype the component functions.

The initialization function (**MQZ_INIT_USERID**), is the main entry point for the component. Other functions are invoked by the entry point address which the initialization function has added into the component entry point vector.

User identifier service samples

MQSeries for OS/2 provides two user identifier samples, a source sample `AMQSZFC0.C` and a DLL `AMQZFCB2.DLL`.

The supplied sample `AMQSZFC0.C` shows how to use environmental variables to specify a user ID and password. See “Sample program for user identifier service” on page 391, for an explanation and a source file listing.

You can use `AMQZFCB2.DLL` directly, by simply specifying the path and filename in the *`ServiceComponent`* stanza in `qm.ini`. “Defining the service to OS/2” on page 389 shows the appropriate stanzas.

Note: This sample is distinct from the supplied sample user identifier service DLL `AMQZFCB2.DLL`, which has a similar, but not identical function.

Sample program for user identifier service

The sample AMQZFC0.C is supplied with MQSeries for OS/2. Although it is not available on other platforms, it does show you how to create an installable service. You can, therefore, use it as a basis for your own installable services.

To use the user identifier service, you must set the user identifier service stanza in the qm.ini file. See "Defining the service to OS/2" on page 389 for details.

Purpose

AMQSZFC0.C enables a queue manager to insert a user-defined user ID into messages sent by applications connected to this queue manager. The mechanism for this is user-defined (OS/2) environment variables. You could replace this with your own mechanism.

Setting the environment variables

The user-defined environment variables are MQS_USERID and MQS_PASSWORD, which are used for the user ID and password respectively. You must set these, before you start the queue manager. Use the following OS/2 commands:

```
SET MQS_USERID=LEMON
SET MQS_PASSWORD=ABC123
```

You can, for example, add these lines to your CONFIG.SYS file.

The sample copies these variables into shared memory where they can be accessed by MQSeries internal processes.

Note: The password you supply is **not** used by MQSeries for OS/2 Warp.

How the sample works

The sample consists of the following:

- Initialize the service entry point vectors, using the **MQZEP** function to pass the address of a user-written function entry point. **MQZEP** is called for each such function.
- Initialize this instance of the service.
- For the primary initialization, obtain the user ID and password and put them into this component's shared memory.

The **MQZ_FIND_USERID** (Find User ID) function is invoked by the local queue manager whenever an application makes an MQCONN request. In this case, the function retrieves the user ID and password from shared memory.

Note: Changing the environment variables after the queue manager is started has no effect. The original ones are used.

```
/*****************************************************************/
/*                                                               */
/* Program name: AMQSZFC0                                        */
/*                                                               */
/* Description : Sample program for the UserIdentifierService    */
/*                 using the environment variables to get the    */
/*                 password and userID                           */
/*                                                               */
/* Statement:     Licensed Materials - Property of IBM           */
/*                                                               */
/*                33H2205, 5622-908                              */
/*                33H2267, 5765-623                              */
/*                29H0990, 5697-176                              */
/*                (C) Copyright IBM Corp. 1994, 1995             */
/*                                                               */
/*****************************************************************/

/*---------------------------------------------------------------*/
/* Includes                                                      */
/*---------------------------------------------------------------*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include <cmqc.h>         /* MQI API                             */
#include <cmqzc.h>        /* MQI API for installable services    */

/*---------------------------------------------------------------*/
/* Typedefs                                                      */
/*---------------------------------------------------------------*/

typedef struct tag_USERDATA  USERDATA, *PUSERDATA;

struct tag_USERDATA
{
  MQCHAR12  UserID;                   /* UserID                  */
  MQCHAR12  Password;                 /* Password                */
};

/*---------------------------------------------------------------*/
/* Prototypes                                                    */
/*---------------------------------------------------------------*/

MQZ_INIT_USERID mqs_userid_init;
MQZ_FIND_USERID mqs_userid_find;

/*****************************************************************/
/*                                                               */
/* Function Name:  mqs_userid_init                               */
/*                                                               */
/* Description:    Initialise the instance of the Userid Pluggable*/
/*                 Service                                       */
/*                                                               */
/*****************************************************************/
void MQENTRY mqs_userid_init (MQHCONFIG hconfig,
                              MQLONG    options,
                              MQCHAR48  QMgrName,
                              MQLONG    ComponentDataLength,
                              PMQBYTE   data,
                              PMQLONG   Version,
                              PMQLONG   CompCode_ptr,
                              PMQLONG   Reason_ptr)
{
  MQLONG cc = MQCC_OK;
  MQLONG rc = MQRC_NONE;

  /*****************************************************************/
  /* Initialise the Entry point vector                             */
  /*****************************************************************/

  if (cc == MQCC_OK) MQZEP(hconfig, MQZID_INIT_USERID, (PMQFUNC) mqs_userid_init, &cc, &rc);
```

```
  if (cc == MQCC_OK) MQZEP(hconfig, MQZID_TERM_USERID, (PMQFUNC) NULL,           &cc, &rc);
  if (cc == MQCC_OK) MQZEP(hconfig, MQZID_FIND_USERID, (PMQFUNC) mqs_userid_find, &cc, &rc);

  if (cc != MQCC_OK)
  {
    cc = MQCC_FAILED;
    rc = MQRC_INITIALIZATION_FAILED;
  }

  /*****************************************************************/
  /* For the primary initialisation, Obtain the Userid and Password  */
  /* and put it into this components shared storage                  */
  /*****************************************************************/

  if ((cc == MQCC_OK) && (options == MQZIO_PRIMARY))
  {
    char *env_userid   = getenv("MQS_USERID");
    char *env_password = getenv("MQS_PASSWORD");

    if ((ComponentDataLength == sizeof(USERDATA)) &&
        (env_userid   != NULL) &&
        (env_password != NULL))
    {
      PUSERDATA UserData = (PUSERDATA) data;

      strncpy(UserData->UserID,   env_userid  , sizeof(MQCHAR12));
      strncpy(UserData->Password, env_password, sizeof(MQCHAR12));
    }
    else
    {
      cc = MQCC_FAILED;
      rc = MQRC_INITIALIZATION_FAILED;
    }
  }

  /*****************************************************************/
  /* Set the return code and reason                                */
  /*****************************************************************/

  *CompCode_ptr = cc;
  *Reason_ptr   = rc;

  return;
}

/*****************************************************************/
/*                                                               */
/* Function Name:  mqs_userid_find                               */
/*                                                               */
/* Description:    Find the userID/password                      */
/*                                                               */
/*****************************************************************/
void MQENTRY mqs_userid_find (MQCHAR48    QMgrName,
                              MQCHAR12    UserID,
                              MQCHAR12    Password,
                              PMQBYTE     ComponentData,
                              PMQLONG     Continuation,
                              PMQLONG     CompCode_ptr,
                              PMQLONG     Reason_ptr)
{
  PUSERDATA pUserData = (PUSERDATA) ComponentData;

  /*****************************************************************/
  /* Return the UserID & password to the caller              */
  /*****************************************************************/

  memcpy (UserID,   pUserData->UserID,   sizeof(MQCHAR12));
  memcpy (Password, pUserData->Password, sizeof(MQCHAR12));

  /*****************************************************************/
  /* Set the continuation, return code & reason codes           */
  /*****************************************************************/
```

```
  *Continuation = MQZCI_DEFAULT;
  *CompCode_ptr = MQCC_OK;
  *Reason_ptr   = MQRC_NONE;

  return;
}
```

Chapter 15. Installable services interface

This chapter provides reference information for the installable services.

The functions and data types are in alphabetic order within the group for each service type.

Table 25. Installable services functions

Service type	*Functions*	*page*
All	MQZEP - Add component entry point	396
	MQHCONFIG – Configuration handle	398
	PMQFUNC – Pointer to function	398
Authorization	MQZ_CHECK_AUTHORITY	399
	MQZ_COPY_ALL_AUTHORITY	405
	MQZ_DELETE_AUTHORITY	408
	MQZ_GET_AUTHORITY	411
	MQZ_GET_EXPLICIT_AUTHORITY	415
	MQZ_INIT_AUTHORITY	419
	MQZ_SET_AUTHORITY	422
	MQZ_TERM_AUTHORITY	426
Name	MQZ_DELETE_NAME	429
	MQZ_INIT_NAME	432
	MQZ_INSERT_NAME	435
	MQZ_LOOKUP_NAME	438
	MQZ_TERM_NAME	441
User Identifier	MQZ_FIND_USERID	444
	MQZ_INIT_USERID	447
	MQZ_TERM_USERID	450

How the functions are shown

For each function there is a description, including the function identifier (for MQZEP).

The *parameters* are shown listed in a box. The parameters must occur in the order shown and they must all be present.

Parameters and data types

Each parameter name is followed by its data type in parentheses. These are the elementary data types described in the *MQSeries Application Programming Reference*.

The C language invocation is also given, after the description of the parameters.

MQZEP – Add component entry point

This function is invoked by a service component, during initialization, to add an entry point to the entry point vector for that service component.

```
MQZEP (Hconfig, Function, EntryPoint, CompCode, Reason)
```

Parameters

Hconfig (MQHCONFIG) – input
Configuration handle.

This handle represents the component which is being configured for this particular installable service. It must be the same as the one passed to the component configuration function by the queue manager on the component initialization call.

Function (MQLONG) – input
Function identifier.

Valid values for this are defined for each installable service.

If MQZEP is called more than once for the same function, the last call made provides the entry point which is used.

EntryPoint (PMQFUNC) – input
Function entry point.

This is the address of the entry point provided by the component to perform the function.

The value NULL is valid, and indicates that the function is not provided by this component. NULL is assumed for entry points which are not defined using MQZEP.

CompCode (MQLONG) – output
Completion code.

It is one of the following:

MQCC_OK
Successful completion.
MQCC_FAILED
Call failed.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_FUNCTION_ERROR
(2281, X'8E9') Function identifier not valid for service.

MQRC_HCONFIG_ERROR
(2280, X'8E8') Configuration handle not valid.

For more information on these reason codes, see the *MQSeries Application Programming Reference*.

C invocation

```
MQZEP (Hconfig, Function, EntryPoint, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONFIG  Hconfig;     /* Configuration handle */
MQLONG     Function;    /* Function identifier */
PMQFUNC    EntryPoint;  /* Function entry point */
MQLONG     CompCode;    /* Completion code */
MQLONG     Reason;      /* Reason code qualifying CompCode */
```

MQHCONFIG – Configuration handle

The MQHCONFIG data type represents a configuration handle, that is, the component that is being configured for a particular installable service. A configuration handle must be aligned on its natural boundary.

Note: Applications must test variables of this type for equality only.

MQHCONFIG C declaration

```
typedef MQLONG MQHCONFIG;
```

PMQFUNC – Pointer to function

Pointer to a function.

PMQFUNC C declaration

```
typedef void MQPOINTER PMQFUNC;
```

MQZ_CHECK_AUTHORITY – Check authority

This function is provided by an authorization service component, and is invoked by the queue manager to check whether an entity has authority to perform a particular action, or actions, on a specified object.

The function identifier for this function (for MQZEP) is MQZID_CHECK_AUTHORITY.

```
MQZ_CHECK_AUTHORITY (QMgrName, EntityName, EntityType, ObjectName,
                     ObjectType, Authority, ComponentData, Continuation,
                     CompCode, Reason)
```

Parameters

QMgrName (MQCHAR48) – input
Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

EntityName (MQCHAR12) – input
Entity name.

The name of the entity whose authorization to the object is to be checked. The maximum length of the string is 12 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

It is not essential for this entity to be known to the underlying security service. If it is not known, the authorizations of the special **nobody** group (to which all entities are assumed to belong) are used for the check. An all-blank name is valid and can be used in this way.

EntityType (MQLONG) – input
Entity type.

The type of entity specified by *EntityName*. It is one of the following:

MQZAET_PRINCIPAL
Principal.
MQZAET_GROUP
Group.

ObjectName (MQCHAR48) – input
Object name.

The name of the object to which access is required. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType (MQLONG) – input
Object type.

The type of entity specified by *ObjectName*. It is one of the following:

MQOT_CHANNEL
Channel.
MQOT_PROCESS
Process definition.
MQOT_Q
Queue.
MQOT_Q_MGR
Queue manager.

Authority (MQLONG) – input
Authority to be checked.

If one authorization is being checked, this field is equal to the appropriate authorization operation (MQZAO_* constant). If more than one authorization is being checked, it is the bitwise OR of the corresponding MQZAO_* constants.

The following authorizations apply to use of the MQI calls:

MQZAO_CONNECT
Ability to use the MQCONN call.

MQZAO_BROWSE
Ability to use the MQGET call with a browse option.

This allows the MQGMO_BROWSE_FIRST, MQGMO_BROWSE_MSG_UNDER_CURSOR, or MQGMO_BROWSE_NEXT option to be specified on the MQGET call.

MQZAO_INPUT
Ability to use the MQGET call with an input option.

This allows the MQOO_INPUT_SHARED, MQOO_INPUT_EXCLUSIVE, or MQOO_INPUT_AS_Q_DEF option to be specified on the MQOPEN call.

MQZAO_OUTPUT
Ability to use the MQPUT call.

This allows the MQOO_OUTPUT option to be specified on the MQOPEN call.

MQZAO_INQUIRE
Ability to use the MQINQ call.

This allows the MQOO_INQUIRE option to be specified on the MQOPEN call.

MQZAO_SET
Ability to use the MQSET call.

This allows the MQOO_SET option to be specified on the MQOPEN call.

MQZAO_PASS_IDENTITY_CONTEXT
Ability to pass identity context.

This allows the MQOO_PASS_IDENTITY_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_PASS_IDENTITY_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_PASS_ALL_CONTEXT
Ability to pass all context.

This allows the MQOO_PASS_ALL_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_PASS_ALL_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_SET_IDENTITY_CONTEXT
Ability to set identity context.

This allows the MQOO_SET_IDENTITY_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_SET_IDENTITY_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_SET_ALL_CONTEXT
Ability to set all context.

This allows the MQOO_SET_ALL_CONTEXT option to be specified on the MQOPEN call, and the MQPMO_SET_ALL_CONTEXT option to be specified on the MQPUT and MQPUT1 calls.

MQZAO_ALTERNATE_USER_AUTHORITY
Ability to use alternate user authority.

This allows the MQOO_ALTERNATE_USER_AUTHORITY option to be specified on the MQOPEN call, and the MQPMO_ALTERNATE_USER_AUTHORITY option to be specified on the MQPUT1 call.

MQZAO_ALL_MQI
All of the MQI authorizations.

This enables all of the authorizations described above.

The following authorizations apply to administration of a queue manager:

MQZAO_CREATE
Ability to create objects of a specified type.

MQZAO_DELETE
Ability to delete a specified object.

MQZAO_DISPLAY
Ability to display the attributes of a specified object.

MQZAO_CHANGE
Ability to change the attributes of a specified object.

MQZAO_CLEAR
Ability to delete all messages from a specified queue.

MQZAO_AUTHORIZE
Ability to authorize other users for a specified object.

MQZAO_ALL_ADMIN
All of the administration authorizations.

The following authorizations apply to both use of the MQI and to administration of a queue manager:

MQZAO_NONE
No authorizations.

MQZAO_ALL
All authorizations.

ComponentData (MQBYTE×*ComponentDataLength*) – input/output
Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) – output
Continuation indicator set by component.

The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_CHECK_AUTHORITY this has the same effect as MQZCI_STOP.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

CompCode (MQLONG) – output
Completion code.

It is one of the following:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.
MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

For more information on these reason codes, see the *MQSeries Application Programming Reference*.

C invocation

```
MQZ_CHECK_AUTHORITY (QMgrName, EntityName, EntityType, ObjectName,
                     ObjectType, Authority, ComponentData,
                     &Continuation, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR12  EntityName;        /* Entity name */
MQLONG    EntityType;        /* Entity type */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQLONG    Authority;         /* Authority to be checked */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_COPY_ALL_AUTHORITY – Copy all authority

This function is provided by an authorization service component. It is invoked by the queue manager to copy all of the authorizations that are currently in force for a reference object to another object.

The function identifier for this function (for MQZEP) is MQZID_COPY_ALL_AUTHORITY.

```
MQZ_COPY_ALL_AUTHORITY (QMgrName, RefObjectName, ObjectName,
                        ObjectType, ComponentData, Continuation, CompCode,
                        Reason)
```

Parameters

QMgrName (MQCHAR48) – input
Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

RefObjectName (MQCHAR48) – input
Reference object name.

The name of the reference object, the authorizations for which are to be copied. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

ObjectName (MQCHAR48) – input
Object name.

The name of the object for which accesses are to be set. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

ObjectType (MQLONG) – input
Object type.

The type of object specified by *RefObjectName* and *ObjectName*. It is one of the following:

MQOT_PROCESS
Process definition.
MQOT_Q
Queue.
MQOT_Q_MGR
Queue manager.

ComponentData (MQBYTE×*ComponentDataLength*) – input/output
Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) – output
Continuation indicator set by component.

The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_COPY_ALL_AUTHORITY this has the same effect as MQZCI_STOP.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

CompCode (MQLONG) – output
Completion code.

It is one of the following:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

MQRC_UNKNOWN_REF_OBJECT
(2294, X'8F6') Reference object unknown.

For more information on these reason codes, see the *MQSeries Application Programming Reference*.

C invocation

```
MQZ_COPY_ALL_AUTHORITY (QMgrName, RefObjectName, ObjectName, ObjectType,
                        ComponentData, &Continuation, &CompCode,
                        &Reason);
```

Declare the parameters as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR48  RefObjectName;     /* Reference object name */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_DELETE_AUTHORITY – Delete authority

This function is provided by an authorization service component, and is invoked by the queue manager to delete all of the authorizations associated with the specified object.

The function identifier for this function (for MQZEP) is MQZID_DELETE_AUTHORITY.

```
MQZ_DELETE_AUTHORITY (QMgrName, ObjectName, ObjectType,
                      ComponentData, Continuation, CompCode, Reason)
```

Parameters

QMgrName (MQCHAR48) – input
Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

ObjectName (MQCHAR48) – input
Object name.

The name of the object for which accesses are to be deleted. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType (MQLONG) – input
Object type.

The type of entity specified by *ObjectName*. It is one of the following:

MQOT_PROCESS
Process definition.
MQOT_Q
Queue.
MQOT_Q_MGR
Queue manager.

ComponentData (MQBYTE×*ComponentDataLength*) – input/output
Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) – output
Continuation indicator set by component.

The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_DELETE_AUTHORITY this has the same effect as MQZCI_STOP.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

CompCode (MQLONG) – output
Completion code.

It is one of the following:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

For more information on these reason codes, see the *MQSeries Application Programming Reference*.

C invocation

```
MQZ_DELETE_AUTHORITY (QMgrName, ObjectName, ObjectType, ComponentData,
                      &Continuation, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_GET_AUTHORITY – Get authority

This function is provided by an authorization service component, and is invoked by the queue manager to retrieve the authority that an entity has to access the specified object.

The function identifier for this function (for MQZEP) is MQZID_GET_AUTHORITY.

```
MQZ_GET_AUTHORITY (QMgrName, EntityName, EntityType, ObjectName,
                   ObjectType, Authority, ComponentData, Continuation,
                   CompCode, Reason)
```

Parameters

QMgrName (MQCHAR48) – input
Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

EntityName (MQCHAR12) – input
Entity name.

The name of the entity whose access to the object is to be retrieved. The maximum length of the string is 12 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

EntityType (MQLONG) – input
Entity type.

The type of entity specified by *EntityName*. The following value can be specified:

MQZAET_PRINCIPAL
Principal.
MQZAET_GROUP
Group.

ObjectName (MQCHAR48) – input
Object name.

The name of the object for which the entity's authority is to be retrieved. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType (MQLONG) – input
Object type.

The type of entity specified by *ObjectName*. It is one of the following:

MQOT_PROCESS
Process definition.
MQOT_Q
Queue.
MQOT_Q_MGR
Queue manager.

Authority (MQLONG) – output
Authority of entity.

If the entity has one authority, this field is equal to the appropriate authorization operation (MQZAO_* constant). If it has more than one authority, this field is the bitwise OR of the corresponding MQZAO_* constants.

ComponentData (MQBYTE×*ComponentDataLength*) – input/output
Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) – output
Continuation indicator set by component.

The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_GET_AUTHORITY this has the same effect as MQZCI_CONTINUE.
MQZCI_CONTINUE
Continue with next component.
MQZCI_STOP
Do not continue with next component.

CompCode (MQLONG) – output
Completion code.

It is one of the following:

MQCC_OK
Successful completion.
MQCC_FAILED
Call failed.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.
MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.
MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.
MQRC_UNKNOWN_AUTH_ENTITY
(2293, X'8F5') Authorization entity unknown to service.
MQRC_UNKNOWN_ENTITY
(2292, X'8F4') Entity unknown to service.

For more information on these reason codes, see the *MQSeries Application Programming Reference*.

C invocation

```
MQZ_GET_AUTHORITY (QMgrName, EntityName, EntityType, ObjectName,
                   ObjectType, &Authority, ComponentData,
                   &Continuation, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR12  EntityName;        /* Entity name */
MQLONG    EntityType;        /* Entity type */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQLONG    Authority;         /* Authority of entity */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_GET_EXPLICIT_AUTHORITY – Get explicit authority

This function is provided by an authorization service component, and is invoked by the queue manager to retrieve the authority that a named group has to access a specified object (but without the additional authority of the **nobody** group), or the authority that the primary group of the named principal has to access a specified object.

The function identifier for this function (for MQZEP) is MQZID_GET_EXPLICIT_AUTHORITY.

```
MQZ_GET_EXPLICIT_AUTHORITY (QMgrName, EntityName, EntityType,
                            ObjectName, ObjectType, Authority, AuthorityMask,
                            ComponentData, Continuation, CompCode, Reason)
```

Parameters

QMgrName (MQCHAR48) – input
Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

EntityName (MQCHAR12) – input
Entity name.

The name of the entity whose access to the object is to be retrieved. The maximum length of the string is 12 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

EntityType (MQLONG) – input
Entity type.

The type of entity specified by *EntityName*. The following value can be specified:

MQZAET_PRINCIPAL
Principal.
MQZAET_GROUP
Group.

ObjectName (MQCHAR48) – input
Object name.

The name of the object for which the entity's authority is to be retrieved. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType (MQLONG) – input
Object type.

The type of entity specified by *ObjectName*. It is one of the following:

MQOT_PROCESS
Process definition.
MQOT_Q
Queue.
MQOT_Q_MGR
Queue manager.

Authority (MQLONG) – output
Authority of entity.

If the entity has one authority, this field is equal to the appropriate authorization operation (MQZAO_* constant). If it has more than one authority, this field is the bitwise OR of the corresponding MQZAO_* constants.

AuthorityMask (MQLONG) – input
Mask for relevant authorities.

Only the authorities which correspond to a bit which is set on this mask should be affected by this call.

ComponentData (MQBYTE×*ComponentDataLength*) – input/output
Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) – output
Continuation indicator set by component.

The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_GET_EXPLICIT_AUTHORITY this has the same effect as MQZCI_CONTINUE.
MQZCI_CONTINUE
Continue with next component.
MQZCI_STOP
Do not continue with next component.

CompCode (MQLONG) – output
Completion code.

It is one of the following:

MQCC_OK
Successful completion.
MQCC_FAILED
Call failed.

Reason (MQLONG) – output

Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.
MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.
MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.
MQRC_UNKNOWN_AUTH_ENTITY
(2293, X'8F5') Authorization entity unknown to service.
MQRC_UNKNOWN_ENTITY
(2292, X'8F4') Entity unknown to service.

For more information on these reason codes, see the *MQSeries Application Programming Reference*.

C invocation

```
MQZ_GET_EXPLICIT_AUTHORITY (QMgrName, EntityName, EntityType,
                            ObjectName, ObjectType, &Authority,
                            AuthorityMask, ComponentData,
                            &Continuation, &CompCode,
                            &Reason);
```

Declare the parameters as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR12  EntityName;        /* Entity name */
MQLONG    EntityType;        /* Entity type */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQLONG    Authority;         /* Authority of entity */
MQLONG    AuthorityMask;     /* Mask for relevant authorities */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_INIT_AUTHORITY – Initialize authorization service component

This function is provided by an authorization service component, and is invoked by the queue manager during configuration of the component. It is expected to call MQZEP in order to provide information to the queue manager.

The function identifier for this function (for MQZEP) is MQZID_INIT_AUTHORITY.

```
MQZ_INIT_AUTHORITY (Hconfig, Options, QMgrName, ComponentDataLength,
                    ComponentData, Version, CompCode, Reason)
```

Parameters

Hconfig (MQHCONFIG) – input
Configuration handle.

This handle represents the particular component being initialized. It is to be used by the component when calling the queue manager with the MQZEP function.

Options (MQLONG) – input
Initialization options.

It is one of the following:

MQZIO_PRIMARY
Primary initialization.
MQZIO_SECONDARY
Secondary initialization.

QMgrName (MQCHAR48) – input
Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

ComponentDataLength (MQLONG) – input
Length of component data.

Length in bytes of the *ComponentData* area. This length is defined in the component configuration data.

ComponentData (MQBYTE×*ComponentDataLength*) – input/output
Component data.

This is initialized to all zeroes before calling the component's primary initialization function. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions (including the initialization function) provided by this component are preserved, and presented the next time one of this component's functions is called.

Version (MQLONG) – input/output
Version number.

On input to the initialization function, this identifies the *highest* version number that the queue manager supports. The initialization function must change this, if necessary, to the version of the interface which *it* supports. If on return the queue manager does not support the version returned by the component, it calls the component's MQZ_TERM_AUTHORITY function and makes no further use of this component.

The following value is suppported:

MQZAS_VERSION_1
Version 1.

CompCode (MQLONG) – output
Completion code.

It is one of the following:

MQCC_OK
Successful completion.
MQCC_FAILED
Call failed.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_INITIALIZATION_FAILED
(2286, X'8EE') Initialization failed for an undefined reason.
MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

For more information on these reason codes, see the *MQSeries Application Programming Reference*.

C invocation

```
MQZ_INIT_AUTHORITY (Hconfig, Options, QMgrName, ComponentDataLength,
                    ComponentData, &Version, &CompCode,
                    &Reason);
```

Declare the parameters as follows:

```
MQHCONFIG  Hconfig;              /* Configuration handle */
MQLONG     Options;              /* Initialization options */
MQCHAR48   QMgrName;             /* Queue manager name */
MQLONG     ComponentDataLength;  /* Length of component data */
MQBYTE     ComponentData[n];     /* Component data */
MQLONG     Version;              /* Version number */
MQLONG     CompCode;             /* Completion code */
MQLONG     Reason;               /* Reason code qualifying CompCode */
```

MQZ_SET_AUTHORITY – Set authority

This function is provided by an authorization service component, and is invoked by the queue manager to set the authority that an entity has to access the specified object.

The function identifier for this function (for MQZEP) is MQZID_SET_AUTHORITY.

Note: This function overrides any existing authorities. To preserve any existing authorities you must set them again with this function.

```
MQZ_SET_AUTHORITY (QMgrName, EntityName, EntityType, ObjectName,
                   ObjectType, Authority, ComponentData, Continuation,
                   CompCode, Reason)
```

Parameters

QMgrName (MQCHAR48) – input
Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

EntityName (MQCHAR12) – input
Entity name.

The name of the entity whose access to the object is to be set. The maximum length of the string is 12 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

EntityType (MQLONG) – input
Entity type.

The type of entity specified by *EntityName*. The following value can be specified:

MQZAET_GROUP
Group.

ObjectName (MQCHAR48) – input
Object name.

The name of the object to which access is required. The maximum length of the string is 48 characters; if it is shorter than that it is padded to the right with blanks. The name is not terminated by a null character.

If *ObjectType* is MQOT_Q_MGR, this name is the same as *QMgrName*.

ObjectType (MQLONG) – input
Object type.

The type of entity specified by *ObjectName*. It is one of the following:

MQOT_PROCESS
Process definition.
MQOT_Q
Queue.
MQOT_Q_MGR
Queue manager.

Authority (MQLONG) – input
Authority to be checked.

If one authorization is being set, this field is equal to the appropriate authorization operation (MQZAO_* constant). If more than one authorization is being set, it is the bitwise OR of the corresponding MQZAO_* constants.

ComponentData (MQBYTE×*ComponentDataLength*) – input/output
Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_AUTHORITY call.

Continuation (MQLONG) – output
Continuation indicator set by component.

The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_SET_AUTHORITY this has the same effect as MQZCI_STOP.
MQZCI_CONTINUE
Continue with next component.
MQZCI_STOP
Do not continue with next component.

CompCode (MQLONG) – output
Completion code.

It is one of the following:

MQCC_OK
Successful completion.
MQCC_FAILED
Call failed.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_NOT_AUTHORIZED
(2035, X'7F3') Not authorized for access.
MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.
MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.
MQRC_UNKNOWN_AUTH_ENTITY
(2293, X'8F5') Authorization entity unknown to service.
MQRC_UNKNOWN_ENTITY
(2292, X'8F4') Entity unknown to service.

For more information on these reason codes, see the *MQSeries Application Programming Reference*.

C invocation

```
MQZ_SET_AUTHORITY (QMgrName, EntityName, EntityType, ObjectName,
                   ObjectType, Authority, ComponentData,
                   &Continuation, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR12  EntityName;        /* Entity name */
MQLONG    EntityType;        /* Entity type */
MQCHAR48  ObjectName;        /* Object name */
MQLONG    ObjectType;        /* Object type */
MQLONG    Authority;         /* Authority to be checked */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_TERM_AUTHORITY – Terminate authorization service component

This function is provided by an authorization service component, and is invoked by the queue manager when it no longer requires the services of this component. The function must perform any cleanup required by the component.

The function identifier for this function (for MQZEP) is MQZID_TERM_AUTHORITY.

```
MQZ_TERM_AUTHORITY (Hconfig, Options, QMgrName, ComponentData,
                    CompCode, Reason)
```

Parameters

Hconfig (MQHCONFIG) – input
Configuration handle.

This handle represents the particular component being terminated.

Options (MQLONG) – input
Termination options.

It is one of the following:

MQZTO_PRIMARY
Primary termination.
MQZTO_SECONDARY
Secondary termination.

QMgrName (MQCHAR48) – input
Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the authorization service interface does not require the component to make use of it in any defined manner.

ComponentData (MQBYTE×*ComponentDataLength*) – input/output
Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter on the MQZ_INIT_AUTHORITY call.

When the MQZ_TERM_AUTHORITY call has completed, the queue manager discards this data.

CompCode (MQLONG) – output
Completion code.

It is one of the following:

MQCC_OK
: Successful completion.

MQCC_FAILED
: Call failed.

Reason (MQLONG) – output
: Reason code qualifying *CompCode*.

  If *CompCode* is MQCC_OK:

  MQRC_NONE
  : (0, X'000') No reason to report.

  If *CompCode* is MQCC_FAILED:

  MQRC_SERVICE_NOT_AVAILABLE
  : (2285, X'8ED') Underlying service not available.

  MQRC_TERMINATION_FAILED
  : (2287, X'8FF') Termination failed for an undefined reason.

  For more information on these reason codes, see the *MQSeries Application Programming Reference*.

C invocation

```
MQZ_TERM_AUTHORITY (Hconfig, Options, QMgrName, ComponentData,
                    &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONFIG  Hconfig;           /* Configuration handle */
MQLONG     Options;           /* Termination options */
MQCHAR48   QMgrName;          /* Queue manager name */
MQBYTE     ComponentData[n];  /* Component data */
MQLONG     CompCode;          /* Completion code */
MQLONG     Reason;            /* Reason code qualifying CompCode */
```

MQZ_DELETE_NAME – Delete name from service

This function is provided by a name service component, and is invoked by the queue manager to delete an entry for the specified queue.

The function identifier for this function (for MQZEP) is MQZID_DELETE_NAME.

```
MQZ_DELETE_NAME (QMgrName, QName, ComponentData, Continuation,
                 CompCode, Reason)
```

Parameters

QMgrName (MQCHAR48) – input
Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the name service interface does not require the component to make use of it in any defined manner.

QName (MQCHAR48) – input
Queue name.

The name of the queue for which an entry is to be deleted. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

ComponentData (MQBYTE×*ComponentDataLength*) – input/output
Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_NAME call.

Continuation (MQLONG) – output
Continuation indicator set by component.

For MQZ_DELETE_NAME, the queue manager does not attempt to invoke another component, whatever is returned in *Continuation*.

The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.
MQZCI_STOP
Do not continue with next component.

CompCode (MQLONG) – output
Completion code.

It is one of the following:

MQCC_OK
: Successful completion.

MQCC_WARNING
: Warning (partial completion).

MQCC_FAILED
: Call failed.

Reason (MQLONG) – output
: Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
: (0, X'000') No reason to report.

If *CompCode* is MQCC_WARNING:

MQRC_UNKNOWN_Q_NAME
: (2288, X'8F0') Queue name not found.

  Note: It may not be possible to return this code if the underlying service simply responds with success for this case.

If *CompCode* is MQCC_FAILED:

MQRC_SERVICE_ERROR
: (2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
: (2285, X'8ED') Underlying service not available.

For more information on these reason codes, see the *MQSeries Application Programming Reference*.

C invocation

```
MQZ_DELETE_NAME (QMgrName, QName, ComponentData, &Continuation,
                 &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR48  QName;             /* Queue name */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_INIT_NAME – Initialize name service component

This function is provided by a name service component, and is invoked by the queue manager during configuration of the component. It is expected to call MQZEP in order to provide information to the queue manager.

The function identifier for this function (for MQZEP) is MQZID_INIT_NAME.

```
MQZ_INIT_NAME (Hconfig, Options, QMgrName, ComponentDataLength,
               ComponentData, Version, CompCode, Reason)
```

Parameters

Hconfig (MQHCONFIG) – input
Configuration handle.

This handle represents the particular component being initialized. It is to be used by the component when calling the queue manager with the MQZEP function.

Options (MQLONG) – input
Initialization options.

It is one of the following:

MQZIO_PRIMARY
Primary initialization.
MQZIO_SECONDARY
Secondary initialization.

QMgrName (MQCHAR48) – input
Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the name service interface does not require the component to make use of it in any defined manner.

ComponentDataLength (MQLONG) – input
Length of component data.

Length in bytes of the *ComponentData* area. This length is defined in the component configuration data.

ComponentData (MQBYTE×*ComponentDataLength*) – input/output
Component data.

This is initialized to all zeroes before calling the component's primary initialization function. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions (including the initialization function) provided by this component are preserved, and presented the next time one of this component's functions is called.

Component data is in shared memory accessible to all processes. Therefore primary initialization is the first process initialization and secondary initialization is any subsequent process initialization.

Version (MQLONG) – input/output
Version number.

On input to the initialization function, this identifies the *highest* version number that the queue manager supports. The initialization function must change this, if necessary, to the version of the interface which *it* supports. If on return the queue manager does not support the version returned by the component, it calls the component's MQZ_TERM_NAME function and makes no further use of this component.

The following value is suppported:

MQZNS_VERSION_1
Version 1.

CompCode (MQLONG) – output
Completion code.

It is one of the following:

MQCC_OK
Successful completion.
MQCC_FAILED
Call failed.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_INITIALIZATION_FAILED
(2286, X'8EE') Initialization failed for an undefined reason.
MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

For more information on these reason codes, see the *MQSeries Application Programming Reference*.

C invocation

```
MQZ_INIT_NAME (Hconfig, Options, QMgrName, ComponentDataLength,
               ComponentData, &Version, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONFIG  Hconfig;              /* Configuration handle */
MQLONG     Options;              /* Initialization options */
MQCHAR48   QMgrName;             /* Queue manager name */
MQLONG     ComponentDataLength;  /* Length of component data */
MQBYTE     ComponentData[n];     /* Component data */
MQLONG     Version;              /* Version number */
MQLONG     CompCode;             /* Completion code */
MQLONG     Reason;               /* Reason code qualifying CompCode */
```

MQZ_INSERT_NAME – Insert name in service

This function is provided by a name service component, and is invoked by the queue manager to insert an entry for the specified queue, containing the name of the queue manager that owns the queue. If the queue is already defined in the service, the call fails.

The function identifier for this function (for MQZEP) is MQZID_INSERT_NAME.

```
MQZ_INSERT_NAME (QMgrName, QName, ResolvedQMgrName, ComponentData,
                 Continuation, CompCode, Reason)
```

Parameters

QMgrName (MQCHAR48) – input
Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the name service interface does not require the component to make use of it in any defined manner.

QName (MQCHAR48) – input
Queue name.

The name of the queue for which an entry is to be inserted. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

ResolvedQMgrName (MQCHAR48) – input
Resolved queue manager name.

The name of the queue manager to which the queue resolves. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

ComponentData (MQBYTE×*ComponentDataLength*) – input/output
Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_NAME call.

Continuation (MQLONG) – output
Continuation indicator set by component.

For MQZ_INSERT_NAME, the queue manager does not attempt to invoke another component, whatever is returned in *Continuation*.

The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.
MQZCI_STOP
Do not continue with next component.

CompCode (MQLONG) – output
Completion code.

It is one of the following:

MQCC_OK
Successful completion.
MQCC_FAILED
Call failed.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_Q_ALREADY_EXISTS
(2290, X'8F2') Queue object already exists.
MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.
MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

For more information on these reason codes, see the *MQSeries Application Programming Reference*.

C invocation

```
MQZ_INSERT_NAME (QMgrName, QName, ResolvedQMgrName, ComponentData,
                 &Continuation, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR48  QName;             /* Queue name */
MQCHAR48  ResolvedQMgrName;  /* Resolved queue manager name */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_LOOKUP_NAME – Lookup name in service

This function is provided by a name service component, and is invoked by the queue manager to retrieve the name of the owning queue manager, for a specified queue.

The function identifier for this function (for MQZEP) is MQZID_LOOKUP_NAME.

```
MQZ_LOOKUP_NAME (QMgrName, QName, ResolvedQMgrName, ComponentData,
                 Continuation, CompCode, Reason)
```

Parameters

QMgrName (MQCHAR48) – input
Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the name service interface does not require the component to make use of it in any defined manner.

QName (MQCHAR48) – input
Queue name.

The name of the queue which is to be resolved. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

ResolvedQMgrName (MQCHAR48) – output
Resolved queue manager name.

If the function completes successfully, this is the name of the queue manager that owns the queue.

The name returned by the service component must be padded on the right with blanks to the full length of the parameter; the name *must not* be terminated by a null character, or contain leading or embedded blanks.

ComponentData (MQBYTE×*ComponentDataLength*) – input/output
Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

Component data is in shared memory accessible to all processes.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_NAME call.

Continuation (MQLONG) – output
Continuation indicator set by component.

For MQZ_LOOKUP_NAME, the queue manager decides whether to invoke another name service component, as follows:

- If *CompCode* is MQCC_OK, no further components are invoked, whatever value is returned in *Continuation*.
- If *CompCode* is not MQCC_OK, a further component is invoked, unless *Continuation* is MQZCI_STOP. This value should not be set without good reason.

The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.
MQZCI_CONTINUE
Continue with next component.
MQZCI_STOP
Do not continue with next component.

CompCode (MQLONG) – output
Completion code.

It is one of the following:

MQCC_OK
Successful completion.
MQCC_FAILED
Call failed.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.
MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.
MQRC_UNKNOWN_Q_NAME
(2288, X'8F0') Queue name not found.

For more information on these reason codes, see the *MQSeries Application Programming Reference*.

C invocation

```
MQZ_LOOKUP_NAME (QMgrName, QName, ResolvedQMgrName, ComponentData,
                 &Continuation, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR48  QName;             /* Queue name */
MQCHAR48  ResolvedQMgrName;  /* Resolved queue manager name */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_TERM_NAME – Terminate name service component

This function is provided by a name service component, and is invoked by the queue manager when it no longer requires the services of this component. The function must perform any cleanup required by the component.

The function identifier for this function (for MQZEP) is MQZID_TERM_NAME.

```
MQZ_TERM_NAME (Hconfig, Options, QMgrName, ComponentData, CompCode,
               Reason)
```

Parameters

Hconfig (MQHCONFIG) – input
Configuration handle.

This handle represents the particular component being terminated.

Options (MQLONG) – input
Termination options.

It is one of the following:

MQZTO_PRIMARY
Primary termination.
MQZTO_SECONDARY
Secondary termination.

QMgrName (MQCHAR48) – input
Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the name service interface does not require the component to make use of it in any defined manner.

ComponentData (MQBYTE×*ComponentDataLength*) – input/output
Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

Component data is in shared memory accessible to all processes.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter on the MQZ_INIT_NAME call.

When the MQZ_TERM_NAME call has completed, the queue manager discards this data.

CompCode (MQLONG) – output
Completion code.

It is one of the following:

MQCC_OK
Successful completion.
MQCC_FAILED
Call failed.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.
MQRC_TERMINATION_FAILED
(2287, X'8FF') Termination failed for an undefined reason.

For more information on these reason codes, see the *MQSeries Application Programming Reference*.

C invocation

```
MQZ_TERM_NAME (Hconfig, Options, QMgrName, ComponentData, &CompCode,
               &Reason);
```

Declare the parameters as follows:

```
MQHCONFIG  Hconfig;           /* Configuration handle */
MQLONG     Options;           /* Termination options */
MQCHAR48   QMgrName;          /* Queue manager name */
MQBYTE     ComponentData[n];  /* Component data */
MQLONG     CompCode;          /* Completion code */
MQLONG     Reason;            /* Reason code qualifying CompCode */
```

MQZ_FIND_USERID – Find user ID

This function is provided by a user ID service component, and is invoked by the queue manager to find the user ID, and optionally the password, to be associated with an application, when the application issues an MQCONN call.

The function identifier for this function (for MQZEP) is MQZID_FIND_USERID.

```
MQZ_FIND_USERID (QMgrName, Userid, Password, ComponentData,
                 Continuation, CompCode, Reason)
```

Parameters

QMgrName (MQCHAR48) – input
Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the user ID service interface does not require the component to make use of it in any defined manner.

Userid (MQCHAR12) – output
User identifier.

The user identifier to be associated with this application. The value returned by the service component must be padded on the right with blanks to the full length of the parameter; the value *must not* be terminated by a null character.

Password (MQCHAR12) – output
Password.

The password associated with the user identifier. The value returned by the service component must be padded on the right with blanks to the full length of the parameter; the value *must not* be terminated by a null character.

If it is not necessary to return a password, the parameter should be set to blanks.

ComponentData (MQBYTE×*ComponentDataLength*) – input/output
Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter of the MQZ_INIT_USERID call.

Continuation (MQLONG) – output
Continuation indicator set by component.

The following values can be specified:

MQZCI_DEFAULT
Continuation dependent on queue manager.

For MQZ_FIND_USERID this has the same effect as MQZCI_CONTINUE.

MQZCI_CONTINUE
Continue with next component.

MQZCI_STOP
Do not continue with next component.

CompCode (MQLONG) – output
Completion code.

It is one of the following:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_SERVICE_ERROR
(2289, X'8F1') Unexpected error occurred accessing service.

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

MQRC_USER_ID_NOT_AVAILABLE
(2291, X'8F3') Unable to determine the user ID.

For more information on these reason codes, see the *MQSeries Application Programming Reference*.

C invocation

```
MQZ_FIND_USERID (QMgrName, Userid, Password, ComponentData,
                 &Continuation, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQCHAR48  QMgrName;          /* Queue manager name */
MQCHAR12  Userid;            /* User identifier */
MQCHAR12  Password;          /* Password */
MQBYTE    ComponentData[n];  /* Component data */
MQLONG    Continuation;      /* Continuation indicator set by
                                component */
MQLONG    CompCode;          /* Completion code */
MQLONG    Reason;            /* Reason code qualifying CompCode */
```

MQZ_INIT_USERID – Initialize user identifier service component

This function is provided by a user ID service component, and is invoked by the queue manager during configuration of the component. It is expected to call MQZEP in order to provide information to the queue manager.

```
MQZ_INIT_USERID (Hconfig, Options, QMgrName, ComponentDataLength,
                 ComponentData, Version, CompCode, Reason)
```

Parameters

Hconfig (MQHCONFIG) – input
Configuration handle.

This handle represents the particular component being initialized. It is to be used by the component when calling the queue manager with the MQZEP function.

Options (MQLONG) – input
Initialization options.

It always has the following value:

MQZIO_PRIMARY
Primary initialization.

QMgrName (MQCHAR48) – input
Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the user identifier service interface does not require the component to make use of it in any defined manner.

ComponentDataLength (MQLONG) – input
Length of component data.

Length in bytes of the *ComponentData* area. This length is defined in the component configuration data.

ComponentData (MQBYTE×*ComponentDataLength*) – input/output
Component data.

This is initialized to all zeroes before calling the component's primary initialization function. This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions (including the initialization function) provided by this component are preserved, and presented the next time one of this component's functions is called.

Version (MQLONG) – input/output
Version number.

On input to the initialization function, this identifies the *highest* version number that the queue manager supports. The initialization function must

change this, if necessary, to the version of the interface which *it* supports. If on return the queue manager does not support the version returned by the component, it calls the component's MQZ_TERM_USERID function and makes no further use of this component.

The following value is suppported:

MQZUS_VERSION_1
Version 1.

CompCode (MQLONG) – output
Completion code.

It is one of the following:

MQCC_OK
Successful completion.
MQCC_FAILED
Call failed.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_INITIALIZATION_FAILED
(2286, X'8EE') Initialization failed for an undefined reason.
MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.

For more information on these reason codes, see the *MQSeries Application Programming Reference*.

C invocation

```
MQZ_INIT_USERID (Hconfig, Options, QMgrName, ComponentDataLength,
                 ComponentData, &Version, &CompCode,
                 &Reason);
```

Declare the parameters as follows:

```
MQHCONFIG  Hconfig;              /* Configuration handle */
MQLONG     Options;              /* Initialization options */
MQCHAR48   QMgrName;             /* Queue manager name */
MQLONG     ComponentDataLength;  /* Length of component data */
MQBYTE     ComponentData[n];     /* Component data */
MQLONG     Version;              /* Version number */
MQLONG     CompCode;             /* Completion code */
MQLONG     Reason;               /* Reason code qualifying CompCode */
```

MQZ_TERM_USERID – Terminate user identifier service component

This function is provided by a user identifier service component, and is invoked by the queue manager when it no longer requires the services of this component. The function must perform any cleanup required by the component.

The function identifier for this function (for MQZEP) is MQZID_TERM_USERID.

```
MQZ_TERM_USERID (Hconfig, Options, QMgrName, ComponentData,
                 CompCode, Reason)
```

Parameters

Hconfig (MQHCONFIG) – input
Configuration handle.

This handle represents the particular component being terminated.

Options (MQLONG) – input
Termination options.

This always has the following value:

MQZTO_PRIMARY
Primary termination.

QMgrName (MQCHAR48) – input
Queue manager name.

The name of the queue manager calling the component. This name is padded with blanks to the full length of the parameter; the name is not terminated by a null character.

The queue-manager name is passed to the component for information; the user ID service interface does not require the component to make use of it in any defined manner.

ComponentData (MQBYTE×*ComponentDataLength*) – input/output
Component data.

This data is kept by the queue manager on behalf of this particular component; any changes made to it by any of the functions provided by this component are preserved, and presented the next time one of this component's functions is called.

The length of this data area is passed by the queue manager in the *ComponentDataLength* parameter on the MQZ_INIT_USERID call.

When the MQZ_TERM_USERID call has completed, the queue manager discards this data.

CompCode (MQLONG) – output
Completion code.

It is one of the following:

MQCC_OK
Successful completion.

MQCC_FAILED
Call failed.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

If *CompCode* is MQCC_OK:

MQRC_NONE
(0, X'000') No reason to report.

If *CompCode* is MQCC_FAILED:

MQRC_SERVICE_NOT_AVAILABLE
(2285, X'8ED') Underlying service not available.
MQRC_TERMINATION_FAILED
(2287, X'8FF') Termination failed for an undefined reason.

For more information on these reason codes, see the *MQSeries Application Programming Reference*.

C invocation

```
MQZ_TERM_USERID (Hconfig, Options, QMgrName, ComponentData,
                 &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONFIG  Hconfig;           /* Configuration handle */
MQLONG     Options;           /* Termination options */
MQCHAR48   QMgrName;          /* Queue manager name */
MQBYTE     ComponentData[n];  /* Component data */
MQLONG     CompCode;          /* Completion code */
MQLONG     Reason;            /* Reason code qualifying CompCode */
```

Part 4. Appendixes

Appendix A. Error codes

This book contains the return codes associated with PCFs. The return codes associated with the Application Programming Interface (API) are listed in the *MQSeries Application Programming Reference*.

For each command message a completion code and a reason code are set by the command server to indicate success or failure.

Applications must not depend upon errors being checked for in a specific order, except where specifically noted. If more than one completion code or reason code could arise from a call, the particular error reported depends on the implementation.

In the descriptions that follow, references to a *remote system* mean a system that is remote from the system to which the command was issued.

Completion code

This is returned in the *CompCode* field of the MQCFH – PCF header of the response message. The following are the completion codes:

MQCC_OK
: Command completed successfully.

MQCC_WARNING
: Command completed with warning.

MQCC_FAILED
: Command failed.

MQCC_UNKNOWN
: Whether command succeeded is not known.

The initial value of this field is MQCC_OK.

Reason code

This is returned in the *Reason* field of the MQCFH – PCF header of the response message. The reason code is a qualification to the *CompCode*.

If there is no special reason to report, MQRC_NONE is returned. Typically, a successful call returns MQCC_OK and MQRC_NONE.

If the *CompCode* is either MQCC_WARNING or MQCC_FAILED, the command server always reports a qualifying reason.

Reason codes are returned with MQCC_FAILED unless otherwise stated.

Descriptions of the MQRC_* error codes are given in the *MQSeries Application Programming Reference*. The following is a list, in alphabetic order, of the MQRCCF_* reason codes:

MQRCCF_ALLOCATE_FAILED
Allocation failed.

An attempt to allocate a conversation to a remote system failed. The error may be due to an invalid entry in the channel definition, or it may be that the listening program at the remote system is not running.

Corrective action: Ensure that the channel definition is correct, and start the listening program if necessary. If the error persists, consult your systems administrator.

MQRCCF_ATTR_VALUE_ERROR
Attribute value not valid.

One or more of the attribute values specified was not valid. The error response message contains the failing attribute selectors (with parameter identifier MQIACF_PARAMETER_ID).

Corrective action: Specify only valid attribute values.

MQRCCF_BATCH_INT_ERROR
Batch interval not valid.

The batch interval specified was not valid.

Corrective action: Specify a valid batch interval value.

MQRCCF_BATCH_INT_WRONG_TYPE
Batch interval parameter not allowed for this channel type.

The *BatchInterval* parameter is allowed only for sender and server channels.

Corrective action: Remove the parameter.

MQRCCF_BATCH_SIZE_ERROR
Batch size not valid.

The batch size specified was not valid.

Corrective action: Specify a valid batch size value.

MQRCCF_BIND_FAILED
Bind failed.

The bind to a remote system during session negotiation has failed.

Corrective action: Consult your systems administrator.

MQRCCF_CCSID_ERROR
Coded character-set identifier error.

In a command message, one of the following occurred:

- The *CodedCharSetId* field in the message descriptor of the command does not match the coded character-set identifier of the queue manager at which the command is being processed, or
- The *CodedCharSetId* field in a string parameter structure within the message text of the command is not
  - MQCCSI_DEFAULT, or
  - the coded character-set identifier of the queue manager at which the command is being processed, as in the *CodedCharSetId* field in the message descriptor.

The error response message contains the correct value.

This reason can also occur if a ping cannot be performed because the coded character-set identifiers are not compatible. In this case the correct value is not returned.

Corrective action: Construct the command with the correct coded character-set identifier, and specify this in the message descriptor when sending the command. For ping, use a suitable coded character-set identifier.

MQRCCF_CELL_DIR_NOT_AVAILABLE
Cell directory is not available.

The *Scope* attribute of the queue is to be MQSCO_CELL, but no name service supporting a cell directory has been configured.

Corrective action: Configure the queue manager with a suitable name service.

MQRCCF_CFH_COMMAND_ERROR
Command identifier not valid.

The MQCFH *Command* field value was not valid.

Corrective action: Specify a valid command identifier.

MQRCCF_CFH_CONTROL_ERROR
Control option not valid.

The MQCFH *Control* field value was not valid.

Corrective action: Specify a valid control option.

MQRCCF_CFH_LENGTH_ERROR
Structure length not valid.

The MQCFH *StrucLength* field value was not valid.

Corrective action: Specify a valid structure length.

MQRCCF_CFH_MSG_SEQ_NUMBER_ERR
Message sequence number not valid.

The MQCFH *MsgSeqNumber* field value was not valid.

Corrective action: Specify a valid message sequence number.

MQRCCF_CFH_PARM_COUNT_ERROR
Parameter count not valid.

The MQCFH *ParameterCount* field value was not valid.

Corrective action: Specify a valid parameter count.

MQRCCF_CFH_TYPE_ERROR
Type not valid.

The MQCFH *Type* field value was not valid.

Corrective action: Specify a valid type.

MQRCCF_CFH_VERSION_ERROR
Structure version number is not valid.

The MQCFH *Version* field value was not valid.

Corrective action: Specify a valid structure version number.

MQRCCF_CFIL_COUNT_ERROR
Count of parameter values not valid.

The MQCFIL *Count* field value was not valid.

Corrective action: Specify a valid count of parameter values.

MQRCCF_CFIL_DUPLICATE_VALUE
Duplicate parameter.

In the MQCFIL structure, a duplicate parameter was detected in the list selector.

Corrective action: Check for and remove duplicate parameters.

MQRCCF_CFIL_LENGTH_ERROR
Structure length not valid.

The MQCFIL *StrucLength* field value was not valid.

Corrective action: Specify a valid structure length.

MQRCCF_CFIL_PARM_ID_ERROR
Parameter identifier is not valid.

The MQCFIL *Parameter* field value was not valid.

Corrective action: Specify a valid parameter identifier.

MQRCCF_CFIN_DUPLICATE_PARM
Duplicate parameter.

A MQCFIN duplicate parameter was detected.

Corrective action: Check for and remove duplicate parameters.

MQRCCF_CFIN_LENGTH_ERROR
Structure length not valid.

The MQCFIN *StrucLength* field value was not valid.

Corrective action: Specify a valid structure length.

MQRCCF_CFIN_PARM_ID_ERROR
Parameter identifier is not valid.

The MQCFIN *Parameter* field value was not valid.

Corrective action: Specify a valid parameter identifier.

MQRCCF_CFSL_DUPLICATE_PARM
Duplicate parameter.

A MQCFSL duplicate parameter was detected.

Corrective action: Check for and remove duplicate parameters.

This reason can occur if the same parameter is repeated with an MQCFST structure followed by an MQCFSL structure.

MQRCCF_CFSL_TOTAL_LENGTH_ERROR
Total string length error.

The total length of the strings (not including trailing blanks) in a MQCFSL structure exceeds the maximum allowable for the parameter.

Corrective action: Check that the structure has been specified correctly, and if so reduce the number of strings.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

A MQCFST duplicate parameter was detected.

Corrective action: Check for and remove duplicate parameters.

This reason can occur if the same parameter is repeated with an MQCFSL structure followed by an MQCFST structure.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

The MQCFST *StrucLength* field value was not valid. The value was not a multiple of four or was inconsistent with the MQCFST *StringLength* field value.

Corrective action: Specify a valid structure length.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier is not valid.

The MQCFST *Parameter* field value was not valid.

Corrective action: Specify a valid parameter identifier.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

The MQCFST *StringLength* field value was not valid. The value was negative or greater than the maximum permitted length of the parameter specified in the *Parameter* field.

Corrective action: Specify a valid string length for the parameter.

MQRCCF_CHAD_ERROR
Channel automatic definition error.

The *ChannelAutoDef* value was not valid.

Corrective action: Specify MQCHAD_ENABLED or MQCHAD_DISABLED.

MQRCCF_CHAD_EVENT_ERROR
Channel automatic definition event error.

The *ChannelAutoDefEvent* value was not valid.

Corrective action: Specify MQEVR_ENABLED or MQEVR_DISABLED.

MQRCCF_CHAD_EVENT_WRONG_TYPE
Channel automatic definition event parameter not allowed for this channel type.

The *ChannelAutoDefEvent* parameter is allowed only for receiver and server-connection channels.

Corrective action: Remove the parameter.

MQRCCF_CHAD_EXIT_ERROR
Channel automatic definition exit name error.

The *ChannelAutoDefExit* value contained characters that are not allowed for program names on the platform in question.

Corrective action: Specify a valid name.

MQRCCF_CHAD_EXIT_WRONG_TYPE
Channel automatic definition exit parameter not allowed for this channel type.

The *ChannelAutoDefExit* parameter is allowed only for receiver and server-connection channels.

Corrective action: Remove the parameter.

MQRCCF_CHAD_WRONG_TYPE
Channel automatic definition parameter not allowed for this channel type.

The *ChannelAutoDef* parameter is allowed only for receiver and server-connection channels.

Corrective action: Remove the parameter.

MQRCCF_CHANNEL_ALREADY_EXISTS
Channel already exists.

An attempt was made to create a channel but the channel already existed and *Replace* was not specified as MQRP_YES.

Corrective action: Specify *Replace* as MQRP_YES or use a different name for the channel to be created.

MQRCCF_CHANNEL_DISABLED
Channel disabled.

An attempt was made to use a channel, but the channel was disabled.

Corrective action: Start the channel.

MQRCCF_CHANNEL_IN_USE
Channel in use.

An attempt was made to perform an operation on a channel, but the channel is currently active.

Corrective action: Stop the channel or wait for it to terminate.

MQRCCF_CHANNEL_INDOUBT
Channel in-doubt.

The requested operation cannot complete because the channel is in doubt.

Corrective action: Examine the status of the channel, and either restart a channel to resolve the in-doubt state, or resolve the channel.

MQRCCF_CHANNEL_NAME_ERROR
Channel name error.

The *ChannelName* parameter contained characters that are not allowed for channel names.

Corrective action: Specify a valid name.

MQRCCF_CHANNEL_NOT_ACTIVE
Channel not active.

An attempt was made to stop a channel, but the channel was already stopped.

Corrective action: No action is required.

MQRCCF_CHANNEL_NOT_FOUND
Channel not found.

The channel specified does not exist.

Corrective action: Specify the name of a channel which exists.

MQRCCF_CHANNEL_TABLE_ERROR
Channel table value not valid.

The *ChannelTable* specified was not valid, or was not appropriate for the channel type specified on an Inquire Channel or Inquire Channel Names command.

Corrective action: Specify a valid channel table value.

MQRCCF_CHANNEL_TYPE_ERROR
Channel type not valid.

The *ChannelType* specified was not valid, or did not match the type of an existing channel being copied, changed or replaced.

Corrective action: Specify a valid channel type.

MQRCCF_CHL_INST_TYPE_ERROR
Channel instance type not valid.

The *ChannelInstanceType* specified was not valid.

Corrective action: Specify a valid channel instance type.

MQRCCF_CHL_STATUS_NOT_FOUND
Channel status not found.

For Inquire Channel Status, no channel status is available for the specified channel. This may indicate that the channel has not been used.

Corrective action: None, unless this is unexpected, in which case consult your systems administrator.

MQRCCF_COMMAND_FAILED
Command failed.

The command has failed.

Corrective action: Refer to the previous error messages for this command.

MQRCCF_COMMIT_FAILED
Commit failed.

An error was received when an attempt was made to commit a unit of work.

Corrective action: Consult your systems administrator.

MQRCCF_CONFIGURATION_ERROR
Configuration error.

A configuration error was detected in the channel definition or communication subsystem, and allocation of a conversation was not possible. This may be caused by one of the following:

- For LU 6.2, either the *ModeName* or the *TpName* is incorrect. The *ModeName* must match that on the remote system, and the *TpName* must be specified. (On OS/400, these are held in the communications Side Object.)
- For LU 6.2, the session may not be established.

- For TCP/IP, the *ConnectionName* in the channel definition cannot be resolved to a network address. This may be because the name has not been correctly specified, or because the name server is not available.

Corrective action: Identify the error and take appropriate action.

MQRCCF_CONN_NAME_ERROR
Error in connection name parameter.

The *ConnectionName* parameter contains one or more blanks at the start of the name.

Corrective action: Specify a valid connection name.

MQRCCF_CONNECTION_CLOSED
Connection closed.

An error occurred while receiving data from a remote system. The connection to the remote system has unexpectedly terminated.

Corrective action: Contact your systems administrator.

MQRCCF_CONNECTION_REFUSED
Connection refused.

The attempt to establish a connection to a remote system was rejected. The remote system might not be configured to allow a connection from this system.

- For LU 6.2 either the user ID or the password supplied to the remote system is incorrect.
- For TCP/IP the remote system may not recognize the local system as valid, or the TCP/IP listener program may not be started.

Corrective action: Correct the error or restart the listener program.

MQRCCF_DATA_CONV_VALUE_ERROR
Data conversion value not valid.

The value specified for *DataConversion* is not valid.

Corrective action: Specify a valid value.

MQRCCF_DATA_TOO_LARGE
Data too large.

The data to be sent exceeds the maximum that can be supported for the command.

Corrective action: Reduce the size of the data.

MQRCCF_DISC_INT_ERROR
Disconnection interval not valid.

The disconnection interval specified was not valid.

Corrective action: Specify a valid disconnection interval.

MQRCCF_DISC_INT_WRONG_TYPE
Disconnection interval not allowed for this channel type.

The *DiscInterval* parameter is only allowed for sender or server channel types.

Corrective action: Remove the parameter.

MQRCCF_DYNAMIC_Q_SCOPE_ERROR
Dynamic queue scope error.

The *Scope* attribute of the queue is to be MQSCO_CELL, but this is not allowed for a dynamic queue.

Corrective action: Predefine the queue if it is to have cell scope.

MQRCCF_ENCODING_ERROR
Encoding error.

The *Encoding* field in the message descriptor of the command does not match that required for the platform at which the command is being processed.

Corrective action: Construct the command with the correct encoding, and specify this in the message descriptor when sending the command.

MQRCCF_ENTRY_ERROR
Invalid connection name.

The connection name in the channel definition could not be resolved into a network address. Either the name server does not contain the entry, or the name server was not available.

Corrective action: Ensure that the connection name is correctly specified and that the name server is available.

MQRCCF_ESCAPE_TYPE_ERROR
Escape type not valid.

The value specified for *EscapeType* is not valid.

Corrective action: Specify a valid value.

MQRCCF_FORCE_VALUE_ERROR
Force value not valid.

The force value specified was not valid.

Corrective action: Specify a valid force value.

MQRCCF_HB_INTERVAL_ERROR
Heartbeat interval not valid.

The *HeartbeatInterval* value was not valid.

Corrective action: Specify a value in the range 0-999 999.

MQRCCF_HB_INTERVAL_WRONG_TYPE
Heartbeat interval parameter not allowed for this channel type.

The *HeartbeatInterval* parameter is allowed only for receiver and requester channels.

Corrective action: Remove the parameter.

MQRCCF_HOST_NOT_AVAILABLE
Remote system not available.

An attempt to allocate a conversation to a remote system was unsuccessful. The error may be transitory, and the allocate may succeed later.

This reason can occur if the listening program at the remote system is not running.

Corrective action: Ensure that the listening program is running, and retry the operation.

MQRCCF_INDOUBT_VALUE_ERROR
In-doubt value not valid.

The value specified for *InDoubt* is not valid.

Corrective action: Specify a valid value.

MQRCCF_LIKE_OBJECT_WRONG_TYPE
New and existing objects have different type.

An attempt was made to create an object based on the definition of an existing object, but the new and existing objects had different types.

Corrective action: Ensure that the new object has the same type as the one on which it is based.

MQRCCF_LISTENER_NOT_STARTED
Listener not started.

The listener program could not be started. Either the communications subsystem has not been started or there are too many jobs waiting in the queue.

Corrective action: Ensure the communications subsystem is started or retry the operation later.

MQRCCF_LONG_RETRY_ERROR
Long retry count not valid.

The long retry count value specified was not valid.

Corrective action: Specify a valid long retry count value.

MQRCCF_LONG_RETRY_WRONG_TYPE
Long retry parameter not allowed for this channel type.

The *LongRetryCount* parameter is only allowed for sender or server channel types.

Corrective action: Remove the parameter.

MQRCCF_LONG_TIMER_ERROR
Long timer not valid.

The long timer (long retry wait interval) value specified was not valid.

Corrective action: Specify a valid long timer value.

MQRCCF_LONG_TIMER_WRONG_TYPE
Long timer parameter not allowed for this channel type.

The *LongRetryInterval* parameter is only allowed for sender or server channel types.

Corrective action: Remove the parameter.

MQRCCF_MAX_MSG_LENGTH_ERROR
Maximum message length not valid.

The maximum message length value specified was not valid.

Corrective action: Specify a valid maximum message length.

MQRCCF_MCA_NAME_ERROR
Message channel agent name error.

The *MCAName* value contained characters that are not allowed for program names on the platform in question.

Corrective action: Specify a valid name.

MQRCCF_MCA_NAME_WRONG_TYPE
Message channel agent name not allowed for this channel type.

The *MCAName* parameter is only allowed for sender, server or requester channel types.

Corrective action: Remove the parameter.

MQRCCF_MCA_TYPE_ERROR
Message channel agent type not valid.

The *MCAType* value specified was not valid.

Corrective action: Specify a valid value.

MQRCCF_MD_FORMAT_ERROR
Format not valid.

The MQMD *Format* field value was not MQFMT_ADMIN.

Corrective action: Specify the valid format.

MQRCCF_MISSING_CONN_NAME
Connection name parameter required but missing.

The *ConnectionName* parameter is required for sender or requester channel types, but is not present.

Corrective action: Add the parameter.

MQRCCF_MQCONN_FAILED
MQCONN call failed.

Corrective action: Check whether the queue manager is active.

MQRCCF_MQGET_FAILED
MQGET call failed.

Corrective action: Check whether the queue manager is active, and the queues involved are correctly set up, and enabled for MQGET.

MQRCCF_MQINQ_FAILED
MQINQ call failed.

Corrective action: Check whether the queue manager is active.

MQRCCF_MQOPEN_FAILED
MQOPEN call failed.

Corrective action: Check whether the queue manager is active, and the queues involved are correctly set up.

MQRCCF_MQPUT_FAILED
MQPUT call failed.

Corrective action: Check whether the queue manager is active, and the queues involved are correctly set up, and not inhibited for puts.

MQRCCF_MQSET_FAILED
MQSET call failed.

Corrective action: Check whether the queue manager is active.

MQRCCF_MR_COUNT_ERROR
Message retry count not valid.

The *MsgRetryCount* value was not valid.

Corrective action: Specify a value in the range 0-999 999 999.

MQRCCF_MR_COUNT_WRONG_TYPE
Message-retry count parameter not allowed for this channel type.

The *MsgRetryCount* parameter is allowed only for receiver and requester channels.

Corrective action: Remove the parameter.

MQRCCF_MR_EXIT_NAME_ERROR
Channel message-retry exit name error.

The *MsgRetryExit* value contained characters that are not allowed for program names on the platform in question.

Corrective action: Specify a valid name.

MQRCCF_MR_EXIT_NAME_WRONG_TYPE
Message-retry exit parameter not allowed for this channel type.

The *MsgRetryExit* parameter is allowed only for receiver and requester channels.

Corrective action: Remove the parameter.

MQRCCF_MR_INTERVAL_ERROR
Message retry interval not valid.

The *MsgRetryInterval* value was not valid.

Corrective action: Specify a value in the range 0-999 999 999.

MQRCCF_MR_INTERVAL_WRONG_TYPE
Message-retry interval parameter not allowed for this channel type.

The *MsgRetryInterval* parameter is allowed only for receiver and requester channels.

Corrective action: Remove the parameter.

MQRCCF_MSG_EXIT_NAME_ERROR
Channel message exit name error.

The *MsgExit* value contained characters that are not allowed for program names on the platform in question.

Corrective action: Specify a valid name.

MQRCCF_MSG_LENGTH_ERROR
Message length not valid.

A message length error was detected. The message data length was inconsistent with the length implied by the parameters in the message, or a positional parameter was out of sequence.

Corrective action: Specify a valid message length, and check that positional parameters are in the correct sequence.

MQRCCF_MSG_SEQ_NUMBER_ERROR
Message sequence number not valid.

The message sequence number parameter value was not valid.

Corrective action: Specify a valid message sequence number.

MQRCCF_MSG_TRUNCATED
Message truncated.

The command server received a message that is larger than its maximum valid message size.

Corrective action: Check the message contents are correct.

MQRCCF_NO_COMMS_MANAGER
Communications manager not available.

The communications subsystem is not available.

Corrective action: Ensure that the communications subsystem has been started.

MQRCCF_NO_STORAGE
Not enough storage available.

Insufficient storage is available.

Corrective action: Consult your systems administrator.

MQRCCF_NOT_XMIT_Q
Queue is not a transmission queue.

The queue specified in the channel definition is not a transmission queue.

Corrective action: Ensure that the queue is specified correctly in the channel definition, and that it is correctly defined to the queue manager.

MQRCCF_NPM_SPEED_ERROR
Nonpersistent message speed not valid.

The *NonPersistentMsgSpeed* value was not valid.

Corrective action: Specify MQNPMS_NORMAL or MQNPMS_FAST.

MQRCCF_NPM_SPEED_WRONG_TYPE
Nonpersistent message speed parameter not allowed for this channel type.

The *NonPersistentMsgSpeed* parameter is allowed only for sender, receiver, server, and requester channels.

Corrective action: Remove the parameter.

MQRCCF_OBJECT_ALREADY_EXISTS
Object already exists.

An attempt was made to create an object, but the object already existed and the *Replace* parameter was not specified as MQRP_YES.

Corrective action: Specify *Replace* as MQRP_YES, or use a different name for the object to be created.

MQRCCF_OBJECT_NAME_ERROR
Object name not valid.

An object name was specified using characters that were not valid.

Corrective action: Specify only valid characters for the name.

MQRCCF_OBJECT_OPEN
Object is open.

An attempt was made to delete or change an object that was in use.

Corrective action: Wait until the object is not in use, and then retry the operation. Alternatively specify *Force* as MQFC_YES for a change command.

MQRCCF_OBJECT_WRONG_TYPE
Object has wrong type.

An attempt was made to replace a queue object with one of a different type.

Corrective action: Ensure that the new object is the same type as the one it is replacing.

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

The MQCFH *ParameterCount* field value was more than the maximum for the command.

Corrective action: Specify a parameter count that is valid for the command.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

The MQCFH *ParameterCount* field value was less than the minimum required for the command.

Corrective action: Specify a parameter count that is valid for the command.

MQRCCF_PARM_SEQUENCE_ERROR
Parameter sequence not valid.

The sequence of parameters is not valid for this command.

Corrective action: Specify the positional parameters in a valid sequence for the command.

MQRCCF_PING_DATA_COMPARE_ERROR
Ping Channel command failed.

The Ping Channel command failed with a data compare error. The data offset that failed is returned in the message (with parameter identifier MQIACF_ERROR_OFFSET).

Corrective action: Consult your systems administrator.

MQRCCF_PING_DATA_COUNT_ERROR
Data count not valid.

The Ping Channel *DataCount* value was not valid.

Corrective action: Specify a valid data count value.

MQRCCF_PING_ERROR
Ping error.

A ping operation can only be issued for a sender or server channel. If the local channel is a receiver channel, you must issue the ping from a remote queue manager.

Corrective action: Reissue the ping request for a different channel of the correct type, or for a receiver channel from a different queue manager.

MQRCCF_PURGE_VALUE_ERROR
Purge value not valid.

The *Purge* value was not valid.

Corrective action: Specify a valid purge value.

MQRCCF_PUT_AUTH_ERROR
Put authority value not valid.

The *PutAuthority* value was not valid.

Corrective action: Specify a valid authority value.

MQRCCF_PUT_AUTH_WRONG_TYPE
Put authority parameter not allowed for this channel type.

The *PutAuthority* parameter is only allowed for receiver or requester channel types.

Corrective action: Remove the parameter.

MQRCCF_Q_ALREADY_IN_CELL
Queue already exists in cell.

An attempt was made to define a queue with cell scope, or to change the scope of an existing queue from queue-manager scope to cell scope, but a queue with that name already existed in the cell.

Corrective action: Do one of the following:

- Delete the existing queue and retry the operation.
- Change the scope of the existing queue from cell to queue-manager and retry the operation.
- Create the new queue with a different name.

MQRCCF_Q_TYPE_ERROR
Queue type not valid.

The *QType* value was not valid.

Corrective action: Specify a valid queue type.

MQRCCF_Q_WRONG_TYPE
Action not valid for the queue of specified type.

An attempt was made to perform an action on a queue of the wrong type.

Corrective action: Specify a queue of the correct type.

MQRCCF_QUIESCE_VALUE_ERROR
Quiesce value not valid.

The *Quiesce* value was not valid.

Corrective action: Specify a valid quiesce value.

MQRCCF_RCV_EXIT_NAME_ERROR
Channel receive exit name error.

The *ReceiveExit* value contained characters that are not allowed for program names on the platform in question.

Corrective action: Specify a valid name.

MQRCCF_RECEIVE_FAILED
Receive failed.

The receive operation failed.

Corrective action: Correct the error and retry the operation.

MQRCCF_RECEIVED_DATA_ERROR
Received data error.

An error occurred while receiving data from a remote system. This may be caused by a communications failure.

Corrective action: Consult your systems administrator.

MQRCCF_REMOTE_QM_TERMINATING
Remote queue manager terminating.

The channel is ending because the remote queue manager is terminating.

Corrective action: Restart the remote queue manager.

MQRCCF_REMOTE_QM_UNAVAILABLE
Remote queue manager not available.

The channel cannot be started because the remote queue manager is not available.

Corrective action: Start the remote queue manager.

MQRCCF_REPLACE_VALUE_ERROR
Replace value not valid.

The *Replace* value was not valid.

Corrective action: Specify a valid replace value.

MQRCCF_SEC_EXIT_NAME_ERROR
Channel security exit name error.

The *SecurityExit* value contained characters that are not allowed for program names on the platform in question.

Corrective action: Specify a valid name.

MQRCCF_SEND_EXIT_NAME_ERROR
Channel send exit name error.

The *SendExit* value contained characters that are not allowed for program names on the platform in question.

Corrective action: Specify a valid name.

MQRCCF_SEND_FAILED
Send failed.

An error occurred while sending data to a remote system. This may be caused by a communications failure.

Corrective action: Consult your systems administrator.

MQRCCF_SEQ_NUMBER_WRAP_ERROR
Sequence wrap number not valid.

The *SeqNumberWrap* value was not valid.

Corrective action: Specify a valid sequence wrap number.

MQRCCF_SHORT_RETRY_ERROR
Short retry count not valid.

The *ShortRetryCount* value was not valid.

Corrective action: Specify a valid short retry count value.

MQRCCF_SHORT_RETRY_WRONG_TYPE
Short retry parameter not allowed for this channel type.

The *ShortRetryCount* parameter is only allowed for sender or server channel types.

Corrective action: Remove the parameter.

MQRCCF_SHORT_TIMER_ERROR
Short timer value not valid.

The *ShortRetryInterval* value was not valid.

Corrective action: Specify a valid short timer value.

MQRCCF_SHORT_TIMER_WRONG_TYPE
Short timer parameter not allowed for this channel type.

The *ShortRetryInterval* parameter is only allowed for sender or server channel types.

Corrective action: Remove the parameter.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

The structure *Type* value was not valid.

Corrective action: Specify a valid structure type.

MQRCCF_SUPPRESSED_BY_EXIT
Action suppressed by exit program.

An attempt was made to define a channel automatically, but this was inhibited by the channel automatic definition exit. The *AuxErrorDataInt1* parameter contains the feedback code from the exit indicating why it inhibited the channel definition.

Corrective action: Examine the value of the *AuxErrorDataInt1* parameter, and take any action that is appropriate.

MQRCCF_TERMINATED_BY_SEC_EXIT
Channel terminated by security exit.

A channel security exit terminated the channel.

Corrective action: Check that the channel is attempting to connect to the correct queue manager, and if so that the security exit is specified correctly, and is working correctly, at both ends.

MQRCCF_UNKNOWN_Q_MGR
Queue manager not known.

The queue manager specified was not known.

Corrective action: Specify the name of the queue manager to which the command is sent, or blank.

MQRCCF_UNKNOWN_REMOTE_CHANNEL
Remote channel not known.

There is no definition of the referenced channel at the remote system.

Corrective action: Ensure that the local channel is correctly defined. If it is, add an appropriate channel definition at the remote system.

MQRCCF_USER_EXIT_NOT_AVAILABLE
User exit not available.

The channel was terminated because the user exit specified does not exist.

Corrective action: Ensure that the user exit is correctly specified and the program is available.

MQRCCF_XMIT_PROTOCOL_TYPE_ERR
Transmission protocol type not valid.

The *TransportType* value was not valid.

Corrective action: Specify a valid transmission protocol type.

MQRCCF_XMIT_Q_NAME_ERROR
Transmission queue name error.

The *XmitQName* parameter contains characters that are not allowed for queue names.

This reason code also occurs if the parameter is not present when a sender or server channel is being created, and no default value is available.

Corrective action: Specify a valid name, or add the parameter.

MQRCCF_XMIT_Q_NAME_WRONG_TYPE
Transmission queue name not allowed for this channel type.

The *XmitQName* parameter is only allowed for sender or server channel types.

Corrective action: Remove the parameter.

Appendix B. Constants

This appendix specifies the values of the named constants that apply to events, commands, responses, and installable services.

The constants are grouped according to the parameter or field to which they relate. All of the names of the constants in a group begin with a common prefix of the form "MQ*xxxx*_", where *xxxx* represents a string of 0 through 4 characters that indicates the nature of the values defined in that group. The constants are ordered alphabetically by the prefix.

Notes:

1. For constants with numeric values, the values are shown in both decimal and hexadecimal forms.
2. Hexadecimal values are represented using the notation `X'hhhh'`, where each "h" denotes a single hexadecimal digit.
3. Character values are shown delimited by single quotation marks; the quotation marks are not part of the value.
4. Blanks in character values are represented by one or more occurrences of the symbol "b".

List of constants

The following sections list all of the named constants mentioned in this book, and show their values.

MQ_* (Lengths of character string and byte fields)

MQ_ACCOUNTING_TOKEN_LENGTH	32	X'00000020'
MQ_APPL_IDENTITY_DATA_LENGTH	32	X'00000020'
MQ_APPL_NAME_LENGTH	28	X'0000001C'
MQ_APPL_ORIGIN_DATA_LENGTH	4	X'00000004'
MQ_AUTHENTICATOR_LENGTH	8	X'00000008'
MQ_BRIDGE_NAME_LENGTH	24	X'00000018'
MQ_CHANNEL_DATE_LENGTH	12	X'0000000C'
MQ_CHANNEL_DESC_LENGTH	64	X'00000040'
MQ_CHANNEL_NAME_LENGTH	20	X'00000014'
MQ_CHANNEL_TIME_LENGTH	8	X'00000008'
MQ_CONN_NAME_LENGTH	264	X'00000108'
MQ_CORREL_ID_LENGTH	24	X'00000018'
MQ_CREATION_DATE_LENGTH	12	X'0000000C'
MQ_CREATION_TIME_LENGTH	8	X'00000008'
MQ_EXIT_DATA_LENGTH	32	X'00000020'
MQ_EXIT_NAME_LENGTH	(environment specific)	
MQ_EXIT_USER_AREA_LENGTH	16	X'00000010'
MQ_FORMAT_LENGTH	8	X'00000008'
MQ_LTERM_OVERRIDE_LENGTH	8	X'00000008'
MQ_LUWID_LENGTH	16	X'00000010'
MQ_MCA_JOB_NAME_LENGTH	28	X'0000001C'
MQ_MCA_NAME_LENGTH	20	X'00000014'

MQ_MFS_MAP_NAME_LENGTH	8	X'00000008'
MQ_MODE_NAME_LENGTH	8	X'00000008'
MQ_MSG_HEADER_LENGTH	4000	X'00000FA0'
MQ_MSG_ID_LENGTH	24	X'00000018'
MQ_NAMELIST_DESC_LENGTH	64	X'00000040'
MQ_NAMELIST_NAME_LENGTH	48	X'00000030'
MQ_OBJECT_INSTANCE_ID_LENGTH	24	X'00000018'
MQ_PASSWORD_LENGTH	12	X'0000000C'
MQ_PROCESS_APPL_ID_LENGTH	256	X'00000100'
MQ_PROCESS_DESC_LENGTH	64	X'00000040'
MQ_PROCESS_ENV_DATA_LENGTH	128	X'00000080'
MQ_PROCESS_NAME_LENGTH	48	X'00000030'
MQ_PROCESS_USER_DATA_LENGTH	128	X'00000080'
MQ_PUT_APPL_NAME_LENGTH	28	X'0000001C'
MQ_PUT_DATE_LENGTH	8	X'00000008'
MQ_PUT_TIME_LENGTH	8	X'00000008'
MQ_Q_DESC_LENGTH	64	X'00000040'
MQ_Q_MGR_DESC_LENGTH	64	X'00000040'
MQ_Q_MGR_NAME_LENGTH	48	X'00000030'
MQ_Q_NAME_LENGTH	48	X'00000030'
MQ_SHORT_CONN_NAME_LENGTH	20	X'00000014'
MQ_STORAGE_CLASS_LENGTH	8	X'00000008'
MQ_TOTAL_EXIT_DATA_LENGTH	999	X'000003E7'
MQ_TOTAL_EXIT_NAME_LENGTH	999	X'000003E7'
MQ_TP_NAME_LENGTH	64	X'00000040'
MQ_TRAN_INSTANCE_ID_LENGTH	16	X'00000010'
MQ_TRIGGER_DATA_LENGTH	64	X'00000040'
MQ_USER_ID_LENGTH	12	X'0000000C'

MQBT_* (Bridge type)

MQBT_OTMA	1	X'00000001'

MQCACF_* (Character attribute command format parameter)

MQCACF_FIRST	3001	X'00000BB9'
MQCACF_FROM_Q_NAME	3001	X'00000BB9'
MQCACF_TO_Q_NAME	3002	X'00000BBA'
MQCACF_FROM_PROCESS_NAME	3003	X'00000BBB'
MQCACF_TO_PROCESS_NAME	3004	X'00000BBC'
MQCACF_FROM_CHANNEL_NAME	3007	X'00000BBF'
MQCACF_TO_CHANNEL_NAME	3008	X'00000BC0'
MQCACF_Q_NAMES	3011	X'00000BC3'
MQCACF_PROCESS_NAMES	3012	X'00000BC4'
MQCACF_ESCAPE_TEXT	3014	X'00000BC6'
MQCACF_LOCAL_Q_NAMES	3015	X'00000BC7'
MQCACF_MODEL_Q_NAMES	3016	X'00000BC8'
MQCACF_ALIAS_Q_NAMES	3017	X'00000BC9'
MQCACF_REMOTE_Q_NAMES	3018	X'00000BCA'
MQCACF_SENDER_CHANNEL_NAMES	3019	X'00000BCB'
MQCACF_SERVER_CHANNEL_NAMES	3020	X'00000BCC'
MQCACF_REQUESTER_CHANNEL_NAMES	3021	X'00000BCD'
MQCACF_RECEIVER_CHANNEL_NAMES	3022	X'00000BCE'
MQCACF_OBJECT_Q_MGR_NAME	3023	X'00000BCF'
MQCACF_APPL_NAME	3024	X'00000BD0'

MQCACF_USER_IDENTIFIER	3025	X'00000BD1'
MQCACF_AUX_ERROR_DATA_STR_1	3026	X'00000BD2'
MQCACF_AUX_ERROR_DATA_STR_2	3027	X'00000BD3'
MQCACF_AUX_ERROR_DATA_STR_3	3028	X'00000BD4'
MQCACF_BRIDGE_NAME	3029	X'00000BD5'
MQCACF_LAST_USED	3029	X'00000BD5'

MQCACH_* (Channel character attribute command format parameter)

MQCACH_FIRST	3501	X'00000DAD'
MQCACH_CHANNEL_NAME	3501	X'00000DAD'
MQCACH_DESC	3502	X'00000DAE'
MQCACH_MODE_NAME	3503	X'00000DAF'
MQCACH_TP_NAME	3504	X'00000DB0'
MQCACH_XMIT_Q_NAME	3505	X'00000DB1'
MQCACH_CONNECTION_NAME	3506	X'00000DB2'
MQCACH_MCA_NAME	3507	X'00000DB3'
MQCACH_SEC_EXIT_NAME	3508	X'00000DB4'
MQCACH_MSG_EXIT_NAME	3509	X'00000DB5'
MQCACH_SEND_EXIT_NAME	3510	X'00000DB6'
MQCACH_RCV_EXIT_NAME	3511	X'00000DB7'
MQCACH_CHANNEL_NAMES	3512	X'00000DB8'
MQCACH_SEC_EXIT_USER_DATA	3513	X'00000DB9'
MQCACH_MSG_EXIT_USER_DATA	3514	X'00000DBA'
MQCACH_SEND_EXIT_USER_DATA	3515	X'00000DBB'
MQCACH_RCV_EXIT_USER_DATA	3516	X'00000DBC'
MQCACH_USER_ID	3517	X'00000DBD'
MQCACH_PASSWORD	3518	X'00000DBE'
MQCACH_LAST_MSG_TIME	3524	X'00000DC4'
MQCACH_LAST_MSG_DATE	3525	X'00000DC5'
MQCACH_MCA_USER_ID	3527	X'00000DC7'
MQCACH_CHANNEL_START_TIME	3528	X'00000DC8'
MQCACH_CHANNEL_START_DATE	3529	X'00000DC9'
MQCACH_MCA_JOB_NAME	3530	X'00000DCA'
MQCACH_LAST_LUWID	3531	X'00000DCB'
MQCACH_CURRENT_LUWID	3532	X'00000DCC'
MQCACH_FORMAT_NAME	3533	X'00000DCD'
MQCACH_MR_EXIT_NAME	3534	X'00000DCE'
MQCACH_MR_EXIT_USER_DATA	3535	X'00000DCF'
MQCACH_LAST_USED	(environment specific)	

MQCDC_* (Channel data conversion)

MQCDC_NO_SENDER_CONVERSION	0	X'00000000'
MQCDC_SENDER_CONVERSION	1	X'00000001'

MQCFC_* (Command format control options)

MQCFC_NOT_LAST	0	X'00000000'
MQCFC_LAST	1	X'00000001'

MQCFH_* (Command format header structure length)

MQCFH_STRUC_LENGTH	36	X'00000024'

MQCFH_* (Command format header version)

MQCFH_VERSION_1	1	X'00000001'
MQCFH_CURRENT_VERSION	1	X'00000001'

MQCFIL_* (Command format integer-list parameter structure length)

MQCFIL_STRUC_LENGTH_FIXED	16	X'00000010'

MQCFIN_* (Command format integer parameter structure length)

MQCFIN_STRUC_LENGTH	16	X'00000010'

MQCFSL_* (Command format string-list parameter structure length)

MQCFSL_STRUC_LENGTH_FIXED	24	X'00000018'

MQCFST_* (Command format string parameter structure length)

MQCFST_STRUC_LENGTH_FIXED	20	X'00000014'

MQCFT_* (Command structure type)

MQCFT_COMMAND	1	X'00000001'
MQCFT_RESPONSE	2	X'00000002'
MQCFT_INTEGER	3	X'00000003'
MQCFT_STRING	4	X'00000004'
MQCFT_INTEGER_LIST	5	X'00000005'
MQCFT_STRING_LIST	6	X'00000006'
MQCFT_EVENT	7	X'00000007'

MQCHAD_* (Channel auto-definition event reporting)

MQCHAD_DISABLED	0	X'00000000'
MQCHAD_ENABLED	1	X'00000001'

MQCHS_* (Channel status)

MQCHS_BINDING	1	X'00000001'
MQCHS_STARTING	2	X'00000002'
MQCHS_RUNNING	3	X'00000003'
MQCHS_STOPPING	4	X'00000004'
MQCHS_RETRYING	5	X'00000005'
MQCHS_STOPPED	6	X'00000006'
MQCHS_REQUESTING	7	X'00000007'
MQCHS_PAUSED	8	X'00000008'
MQCHS_INITIALIZING	13	X'0000000D'

MQCHT_* (Channel type)

MQCHT_SENDER	1	X'00000001'
MQCHT_SERVER	2	X'00000002'
MQCHT_RECEIVER	3	X'00000003'
MQCHT_REQUESTER	4	X'00000004'
MQCHT_ALL	5	X'00000005'
MQCHT_CLNTCONN	6	X'00000006'
MQCHT_SVRCONN	7	X'00000007'

MQCMD_* (Command identifier)

MQCMD_CHANGE_Q_MGR	1	X'00000001'
MQCMD_INQUIRE_Q_MGR	2	X'00000002'
MQCMD_CHANGE_PROCESS	3	X'00000003'
MQCMD_COPY_PROCESS	4	X'00000004'
MQCMD_CREATE_PROCESS	5	X'00000005'
MQCMD_DELETE_PROCESS	6	X'00000006'
MQCMD_INQUIRE_PROCESS	7	X'00000007'
MQCMD_CHANGE_Q	8	X'00000008'
MQCMD_CLEAR_Q	9	X'00000009'
MQCMD_COPY_Q	10	X'0000000A'
MQCMD_CREATE_Q	11	X'0000000B'
MQCMD_DELETE_Q	12	X'0000000C'
MQCMD_INQUIRE_Q	13	X'0000000D'
MQCMD_RESET_Q_STATS	17	X'00000011'
MQCMD_INQUIRE_Q_NAMES	18	X'00000012'
MQCMD_INQUIRE_PROCESS_NAMES	19	X'00000013'
MQCMD_INQUIRE_CHANNEL_NAMES	20	X'00000014'
MQCMD_CHANGE_CHANNEL	21	X'00000015'
MQCMD_COPY_CHANNEL	22	X'00000016'
MQCMD_CREATE_CHANNEL	23	X'00000017'
MQCMD_DELETE_CHANNEL	24	X'00000018'
MQCMD_INQUIRE_CHANNEL	25	X'00000019'
MQCMD_PING_CHANNEL	26	X'0000001A'
MQCMD_RESET_CHANNEL	27	X'0000001B'
MQCMD_START_CHANNEL	28	X'0000001C'
MQCMD_STOP_CHANNEL	29	X'0000001D'
MQCMD_START_CHANNEL_INIT	30	X'0000001E'
MQCMD_START_CHANNEL_LISTENER	31	X'0000001F'
MQCMD_ESCAPE	38	X'00000026'
MQCMD_RESOLVE_CHANNEL	39	X'00000027'
MQCMD_PING_Q_MGR	40	X'00000028'
MQCMD_INQUIRE_CHANNEL_STATUS	42	X'0000002A'
MQCMD_Q_MGR_EVENT	44	X'0000002C'
MQCMD_PERFM_EVENT	45	X'0000002D'
MQCMD_CHANNEL_EVENT	46	X'0000002E'

MQET_* (Escape type)

MQET_MQSC	1	X'00000001'

MQEVR_* (Event reporting)

MQEVR_DISABLED	0	X'00000000'
MQEVR_ENABLED	1	X'00000001'

MQFC_* (Force control)

MQFC_NO	0	X'00000000'
MQFC_YES	1	X'00000001'

MQIACF_* (Integer attribute command format parameter)

MQIACF_FIRST	1001	X'000003E9'
MQIACF_Q_MGR_ATTRS	1001	X'000003E9'
MQIACF_Q_ATTRS	1002	X'000003EA'
MQIACF_PROCESS_ATTRS	1003	X'000003EB'
MQIACF_FORCE	1005	X'000003ED'
MQIACF_REPLACE	1006	X'000003EE'
MQIACF_PURGE	1007	X'000003EF'
MQIACF_QUIESCE	1008	X'000003F0'
MQIACF_ALL	1009	X'000003F1'
MQIACF_PARAMETER_ID	1012	X'000003F4'
MQIACF_ERROR_ID	1013	X'000003F5'
MQIACF_ERROR_IDENTIFIER	1013	X'000003F5'
MQIACF_SELECTOR	1014	X'000003F6'
MQIACF_CHANNEL_ATTRS	1015	X'000003F7'
MQIACF_ESCAPE_TYPE	1017	X'000003F9'
MQIACF_ERROR_OFFSET	1018	X'000003FA'
MQIACF_REASON_QUALIFIER	1020	X'000003FC'
MQIACF_COMMAND	1021	X'000003FD'
MQIACF_OPEN_OPTIONS	1022	X'000003FE'
MQIACF_AUX_ERROR_DATA_INT_1	1070	X'0000042E'
MQIACF_AUX_ERROR_DATA_INT_2	1071	X'0000042F'
MQIACF_CONV_REASON_CODE	1072	X'00000430'
MQIACF_BRIDGE_TYPE	1073	X'00000431'
MQIACF_LAST_USED	1073	X'00000431'

MQIACH_* (Channel Integer attribute command format parameter)

MQIACH_FIRST	1501	X'000005DD'
MQIACH_XMIT_PROTOCOL_TYPE	1501	X'000005DD'
MQIACH_BATCH_SIZE	1502	X'000005DE'
MQIACH_DISC_INTERVAL	1503	X'000005DF'
MQIACH_SHORT_TIMER	1504	X'000005E0'
MQIACH_SHORT_RETRY	1505	X'000005E1'
MQIACH_LONG_TIMER	1506	X'000005E2'
MQIACH_LONG_RETRY	1507	X'000005E3'
MQIACH_PUT_AUTHORITY	1508	X'000005E4'
MQIACH_SEQUENCE_NUMBER_WRAP	1509	X'000005E5'
MQIACH_MAX_MSG_LENGTH	1510	X'000005E6'
MQIACH_CHANNEL_TYPE	1511	X'000005E7'
MQIACH_DATA_COUNT	1512	X'000005E8'
MQIACH_MSG_SEQUENCE_NUMBER	1514	X'000005EA'
MQIACH_DATA_CONVERSION	1515	X'000005EB'
MQIACH_IN_DOUBT	1516	X'000005EC'
MQIACH_MCA_TYPE	1517	X'000005ED'

MQIACH_CHANNEL_INSTANCE_TYPE	1523	X'000005F3'
MQIACH_CHANNEL_INSTANCE_ATTRS	1524	X'000005F4'
MQIACH_CHANNEL_ERROR_DATA	1525	X'000005F5'
MQIACH_CHANNEL_TABLE	1526	X'000005F6'
MQIACH_CHANNEL_STATUS	1527	X'000005F7'
MQIACH_INDOUBT_STATUS	1528	X'000005F8'
MQIACH_LAST_SEQ_NUMBER	1529	X'000005F9'
MQIACH_CURRENT_MSGS	1531	X'000005FB'
MQIACH_CURRENT_SEQ_NUMBER	1532	X'000005FC'
MQIACH_MSGS	1534	X'000005FE'
MQIACH_BYTES_SENT	1535	X'000005FF'
MQIACH_BYTES_RCVD	1536	X'00000600'
MQIACH_BATCHES	1537	X'00000601'
MQIACH_BUFFERS_SENT	1538	X'00000602'
MQIACH_BUFFERS_RCVD	1539	X'00000603'
MQIACH_LONG_RETRIES_LEFT	1540	X'00000604'
MQIACH_SHORT_RETRIES_LEFT	1541	X'00000605'
MQIACH_MCA_STATUS	1542	X'00000606'
MQIACH_STOP_REQUESTED	1543	X'00000607'
MQIACH_MR_COUNT	1544	X'00000608'
MQIACH_MR_INTERVAL	1545	X'00000609'
MQIACH_NPM_SPEED	1562	X'0000061A'
MQIACH_HB_INTERVAL	1563	X'0000061B'
MQIACH_BATCH_INTERVAL	1564	X'0000061C'

MQOT_* (Object type)

MQOT_Q	1	X'00000001'
MQOT_PROCESS	3	X'00000003'
MQOT_Q_MGR	5	X'00000005'
MQOT_CHANNEL	6	X'00000006'
MQOT_ALL	1001	X'000003E9'
MQOT_ALIAS_Q	1002	X'000003EA'
MQOT_MODEL_Q	1003	X'000003EB'
MQOT_LOCAL_Q	1004	X'000003EC'
MQOT_REMOTE_Q	1005	X'000003ED'
MQOT_SENDER_CHANNEL	1007	X'000003EF'
MQOT_SERVER_CHANNEL	1008	X'000003F0'
MQOT_REQUESTER_CHANNEL	1009	X'000003F1'
MQOT_RECEIVER_CHANNEL	1010	X'000003F2'
MQOT_CURRENT_CHANNEL	1011	X'000003F3'
MQOT_SAVED_CHANNEL	1012	X'000003F4'

MQPO_* (Purge option)

MQPO_NO	0	X'00000000'
MQPO_YES	1	X'00000001'

MQQO_* (Quiesce option)

MQQO_NO	0	X'00000000'
MQQO_YES	1	X'00000001'

MQQSIE_* (Service interval events)

MQQSIE_NONE	0	X'00000000'
MQQSIE_HIGH	1	X'00000001'
MQQSIE_OK	2	X'00000002'

MQQT_* (Queue type)

MQQT_LOCAL	1	X'00000001'
MQQT_MODEL	2	X'00000002'
MQQT_ALIAS	3	X'00000003'
MQQT_REMOTE	6	X'00000006'
MQQT_ALL	1001	X'000003E9'

MQRCCF_* (Reason code for command format)

Note: the following list is in **numeric order**.

MQRCCF_CFH_TYPE_ERROR	3001	X'00000BB9'
MQRCCF_CFH_LENGTH_ERROR	3002	X'00000BBA'
MQRCCF_CFH_VERSION_ERROR	3003	X'00000BBB'
MQRCCF_CFH_MSG_SEQ_NUMBER_ERR	3004	X'00000BBC'
MQRCCF_CFH_CONTROL_ERROR	3005	X'00000BBD'
MQRCCF_CFH_PARM_COUNT_ERROR	3006	X'00000BBE'
MQRCCF_CFH_COMMAND_ERROR	3007	X'00000BBF'
MQRCCF_COMMAND_FAILED	3008	X'00000BC0'
MQRCCF_CFIN_LENGTH_ERROR	3009	X'00000BC1'
MQRCCF_CFST_LENGTH_ERROR	3010	X'00000BC2'
MQRCCF_CFST_STRING_LENGTH_ERR	3011	X'00000BC3'
MQRCCF_FORCE_VALUE_ERROR	3012	X'00000BC4'
MQRCCF_STRUCTURE_TYPE_ERROR	3013	X'00000BC5'
MQRCCF_CFIN_PARM_ID_ERROR	3014	X'00000BC6'
MQRCCF_CFST_PARM_ID_ERROR	3015	X'00000BC7'
MQRCCF_MSG_LENGTH_ERROR	3016	X'00000BC8'
MQRCCF_CFIN_DUPLICATE_PARM	3017	X'00000BC9'
MQRCCF_CFST_DUPLICATE_PARM	3018	X'00000BCA'
MQRCCF_PARM_COUNT_TOO_SMALL	3019	X'00000BCB'
MQRCCF_PARM_COUNT_TOO_BIG	3020	X'00000BCC'
MQRCCF_Q_ALREADY_IN_CELL	3021	X'00000BCD'
MQRCCF_Q_TYPE_ERROR	3022	X'00000BCE'
MQRCCF_MD_FORMAT_ERROR	3023	X'00000BCF'
MQRCCF_REPLACE_VALUE_ERROR	3025	X'00000BD1'
MQRCCF_CFIL_DUPLICATE_VALUE	3026	X'00000BD2'
MQRCCF_CFIL_COUNT_ERROR	3027	X'00000BD3'
MQRCCF_CFIL_LENGTH_ERROR	3028	X'00000BD4'
MQRCCF_QUIESCE_VALUE_ERROR	3029	X'00000BD5'
MQRCCF_MSG_SEQ_NUMBER_ERROR	3030	X'00000BD6'
MQRCCF_PING_DATA_COUNT_ERROR	3031	X'00000BD7'
MQRCCF_PING_DATA_COMPARE_ERROR	3032	X'00000BD8'
MQRCCF_CHANNEL_TYPE_ERROR	3034	X'00000BDA'

MQRCCF_PARM_SEQUENCE_ERROR	3035	X'00000BDB'
MQRCCF_XMIT_PROTOCOL_TYPE_ERR	3036	X'00000BDC'
MQRCCF_BATCH_SIZE_ERROR	3037	X'00000BDD'
MQRCCF_DISC_INT_ERROR	3038	X'00000BDE'
MQRCCF_SHORT_RETRY_ERROR	3039	X'00000BDF'
MQRCCF_SHORT_TIMER_ERROR	3040	X'00000BE0'
MQRCCF_LONG_RETRY_ERROR	3041	X'00000BE1'
MQRCCF_LONG_TIMER_ERROR	3042	X'00000BE2'
MQRCCF_SEQ_NUMBER_WRAP_ERROR	3043	X'00000BE3'
MQRCCF_MAX_MSG_LENGTH_ERROR	3044	X'00000BE4'
MQRCCF_PUT_AUTH_ERROR	3045	X'00000BE5'
MQRCCF_PURGE_VALUE_ERROR	3046	X'00000BE6'
MQRCCF_CFIL_PARM_ID_ERROR	3047	X'00000BE7'
MQRCCF_MSG_TRUNCATED	3048	X'00000BE8'
MQRCCF_CCSID_ERROR	3049	X'00000BE9'
MQRCCF_ENCODING_ERROR	3050	X'00000BEA'
MQRCCF_DATA_CONV_VALUE_ERROR	3052	X'00000BEC'
MQRCCF_INDOUBT_VALUE_ERROR	3053	X'00000BED'
MQRCCF_ESCAPE_TYPE_ERROR	3054	X'00000BEE'
MQRCCF_CHANNEL_TABLE_ERROR	3062	X'00000BF6'
MQRCCF_MCA_TYPE_ERROR	3063	X'00000BF7'
MQRCCF_CHL_INST_TYPE_ERROR	3064	X'00000BF8'
MQRCCF_CHL_STATUS_NOT_FOUND	3065	X'00000BF9'
MQRCCF_CFSL_DUPLICATE_PARM	3066	X'00000BFA'
MQRCCF_CFSL_TOTAL_LENGTH_ERROR	3067	X'00000BFB'
MQRCCF_OBJECT_ALREADY_EXISTS	4001	X'00000FA1'
MQRCCF_OBJECT_WRONG_TYPE	4002	X'00000FA2'
MQRCCF_LIKE_OBJECT_WRONG_TYPE	4003	X'00000FA3'
MQRCCF_OBJECT_OPEN	4004	X'00000FA4'
MQRCCF_ATTR_VALUE_ERROR	4005	X'00000FA5'
MQRCCF_UNKNOWN_Q_MGR	4006	X'00000FA6'
MQRCCF_Q_WRONG_TYPE	4007	X'00000FA7'
MQRCCF_OBJECT_NAME_ERROR	4008	X'00000FA8'
MQRCCF_ALLOCATE_FAILED	4009	X'00000FA9'
MQRCCF_HOST_NOT_AVAILABLE	4010	X'00000FAA'
MQRCCF_CONFIGURATION_ERROR	4011	X'00000FAB'
MQRCCF_CONNECTION_REFUSED	4012	X'00000FAC'
MQRCCF_ENTRY_ERROR	4013	X'00000FAD'
MQRCCF_SEND_FAILED	4014	X'00000FAE'
MQRCCF_RECEIVED_DATA_ERROR	4015	X'00000FAF'
MQRCCF_RECEIVE_FAILED	4016	X'00000FB0'
MQRCCF_CONNECTION_CLOSED	4017	X'00000FB1'
MQRCCF_NO_STORAGE	4018	X'00000FB2'
MQRCCF_NO_COMMS_MANAGER	4019	X'00000FB3'
MQRCCF_LISTENER_NOT_STARTED	4020	X'00000FB4'
MQRCCF_BIND_FAILED	4024	X'00000FB8'
MQRCCF_CHANNEL_INDOUBT	4025	X'00000FB9'
MQRCCF_MQCONN_FAILED	4026	X'00000FBA'
MQRCCF_MQOPEN_FAILED	4027	X'00000FBB'
MQRCCF_MQGET_FAILED	4028	X'00000FBC'
MQRCCF_MQPUT_FAILED	4029	X'00000FBD'
MQRCCF_PING_ERROR	4030	X'00000FBE'
MQRCCF_CHANNEL_IN_USE	4031	X'00000FBF'
MQRCCF_CHANNEL_NOT_FOUND	4032	X'00000FC0'

MQRCCF_UNKNOWN_REMOTE_CHANNEL	4033	X'00000FC1'
MQRCCF_REMOTE_QM_UNAVAILABLE	4034	X'00000FC2'
MQRCCF_REMOTE_QM_TERMINATING	4035	X'00000FC3'
MQRCCF_MQINQ_FAILED	4036	X'00000FC4'
MQRCCF_NOT_XMIT_Q	4037	X'00000FC5'
MQRCCF_CHANNEL_DISABLED	4038	X'00000FC6'
MQRCCF_USER_EXIT_NOT_AVAILABLE	4039	X'00000FC7'
MQRCCF_COMMIT_FAILED	4040	X'00000FC8'
MQRCCF_CHANNEL_ALREADY_EXISTS	4042	X'00000FCA'
MQRCCF_DATA_TOO_LARGE	4043	X'00000FCB'
MQRCCF_CHANNEL_NAME_ERROR	4044	X'00000FCC'
MQRCCF_XMIT_Q_NAME_ERROR	4045	X'00000FCD'
MQRCCF_MCA_NAME_ERROR	4047	X'00000FCF'
MQRCCF_SEND_EXIT_NAME_ERROR	4048	X'00000FD0'
MQRCCF_SEC_EXIT_NAME_ERROR	4049	X'00000FD1'
MQRCCF_MSG_EXIT_NAME_ERROR	4050	X'00000FD2'
MQRCCF_RCV_EXIT_NAME_ERROR	4051	X'00000FD3'
MQRCCF_XMIT_Q_NAME_WRONG_TYPE	4052	X'00000FD4'
MQRCCF_MCA_NAME_WRONG_TYPE	4053	X'00000FD5'
MQRCCF_DISC_INT_WRONG_TYPE	4054	X'00000FD6'
MQRCCF_SHORT_RETRY_WRONG_TYPE	4055	X'00000FD7'
MQRCCF_SHORT_TIMER_WRONG_TYPE	4056	X'00000FD8'
MQRCCF_LONG_RETRY_WRONG_TYPE	4057	X'00000FD9'
MQRCCF_LONG_TIMER_WRONG_TYPE	4058	X'00000FDA'
MQRCCF_PUT_AUTH_WRONG_TYPE	4059	X'00000FDB'
MQRCCF_MISSING_CONN_NAME	4061	X'00000FDD'
MQRCCF_CONN_NAME_ERROR	4062	X'00000FDE'
MQRCCF_MQSET_FAILED	4063	X'00000FDF'
MQRCCF_CHANNEL_NOT_ACTIVE	4064	X'00000FE0'
MQRCCF_TERMINATED_BY_SEC_EXIT	4065	X'00000FE1'
MQRCCF_DYNAMIC_Q_SCOPE_ERROR	4067	X'00000FE3'
MQRCCF_CELL_DIR_NOT_AVAILABLE	4068	X'00000FE4'
MQRCCF_MR_COUNT_ERROR	4069	X'00000FE5'
MQRCCF_MR_COUNT_WRONG_TYPE	4070	X'00000FE6'
MQRCCF_MR_EXIT_NAME_ERROR	4071	X'00000FE7'
MQRCCF_MR_EXIT_NAME_WRONG_TYPE	4072	X'00000FE8'
MQRCCF_MR_INTERVAL_ERROR	4073	X'00000FE9'
MQRCCF_MR_INTERVAL_WRONG_TYPE	4074	X'00000FEA'
MQRCCF_NPM_SPEED_ERROR	4075	X'00000FEB'
MQRCCF_NPM_SPEED_WRONG_TYPE	4076	X'00000FEC'
MQRCCF_HB_INTERVAL_ERROR	4077	X'00000FED'
MQRCCF_HB_INTERVAL_WRONG_TYPE	4078	X'00000FEE'
MQRCCF_CHAD_ERROR	4079	X'00000FEF'
MQRCCF_CHAD_WRONG_TYPE	4080	X'00000FF0'
MQRCCF_CHAD_EVENT_ERROR	4081	X'00000FF1'
MQRCCF_CHAD_EVENT_WRONG_TYPE	4082	X'00000FF2'
MQRCCF_CHAD_EXIT_ERROR	4083	X'00000FF3'
MQRCCF_CHAD_EXIT_WRONG_TYPE	4084	X'00000FF4'
MQRCCF_SUPPRESSED_BY_EXIT	4085	X'00000FF5'
MQRCCF_BATCH_INT_ERROR	4086	X'00000FF6'
MQRCCF_BATCH_INT_WRONG_TYPE	4087	X'00000FF7'

MQRP_* (Replace option)

MQRP_NO	0	X'00000000'
MQRP_YES	1	X'00000001'

MQRQ_* (Reason qualifier)

MQRQ_CONN_NOT_AUTHORIZED	1	X'00000001'
MQRQ_OPEN_NOT_AUTHORIZED	2	X'00000002'
MQRQ_CLOSE_NOT_AUTHORIZED	3	X'00000003'
MQRQ_CMD_NOT_AUTHORIZED	4	X'00000004'
MQRQ_Q_MGR_STOPPING	5	X'00000005'
MQRQ_Q_MGR_QUIESCING	6	X'00000006'
MQRQ_CHANNEL_STOPPED_OK	7	X'00000007'
MQRQ_CHANNEL_STOPPED_ERROR	8	X'00000008'
MQRQ_CHANNEL_STOPPED_RETRY	9	X'00000009'
MQRQ_CHANNEL_STOPPED_DISABLED	10	X'0000000A'
MQRQ_BRIDGE_STOPPED_OK	11	X'0000000B'
MQRQ_BRIDGE_STOPPED_ERROR	12	X'0000000C'

MQZAET_* (Authority service entity type)

MQZAET_PRINCIPAL	1	X'00000001'
MQZAET_GROUP	2	X'00000002'

MQZAO_* (Authority service authorization type)

MQZAO_CONNECT	1	X'00000001'
MQZAO_BROWSE	2	X'00000002'
MQZAO_INPUT	4	X'00000004'
MQZAO_OUTPUT	8	X'00000008'
MQZAO_INQUIRE	16	X'00000010'
MQZAO_SET	32	X'00000020'
MQZAO_PASS_IDENTITY_CONTEXT	64	X'00000040'
MQZAO_PASS_ALL_CONTEXT	128	X'00000080'
MQZAO_SET_IDENTITY_CONTEXT	256	X'00000100'
MQZAO_SET_ALL_CONTEXT	512	X'00000200'
MQZAO_ALTERNATE_USER_AUTHORITY	1024	X'00000400'
MQZAO_ALL_MQI	2047	X'000007FF'
MQZAO_CREATE	65536	X'00010000'
MQZAO_DELETE	131072	X'00020000'
MQZAO_DISPLAY	262144	X'00040000'
MQZAO_CHANGE	524288	X'00080000'
MQZAO_CLEAR	1048576	X'00100000'
MQZAO_AUTHORIZE	8388608	X'00800000'
MQZAO_ALL_ADMIN	10354688	X'009E0000'
MQZAO_NONE	0	X'00000000'
MQZAO_ALL	10356735	X'009E07FF'

MQZAS_* (Authority service version)

MQZAS_VERSION_1	1	X'00000001'

MQZCI_* (Continuation indicator)

MQZCI_DEFAULT	0	X'00000000'
MQZCI_CONTINUE	0	X'00000000'
MQZCI_STOP	1	X'00000001'

MQZID_* (Function identifier, all services)

MQZID_INIT	0	X'00000000'
MQZID_TERM	1	X'00000001'

MQZID_* (Function identifier, authority service)

MQZID_INIT_AUTHORITY	0	X'00000000'
MQZID_TERM_AUTHORITY	1	X'00000001'
MQZID_CHECK_AUTHORITY	2	X'00000002'
MQZID_COPY_ALL_AUTHORITY	3	X'00000003'
MQZID_DELETE_AUTHORITY	4	X'00000004'
MQZID_SET_AUTHORITY	5	X'00000005'
MQZID_GET_AUTHORITY	6	X'00000006'
MQZID_GET_EXPLICIT_AUTHORITY	7	X'00000007'

MQZID_* (Function identifier, name service)

MQZID_INIT_NAME	0	X'00000000'
MQZID_TERM_NAME	1	X'00000001'
MQZID_LOOKUP_NAME	2	X'00000002'
MQZID_INSERT_NAME	3	X'00000003'
MQZID_DELETE_NAME	4	X'00000004'

MQZID_* (Function identifier, userid service)

MQZID_INIT_USERID	0	X'00000000'
MQZID_TERM_USERID	1	X'00000001'
MQZID_FIND_USERID	2	X'00000002'

MQZIO_* (Initialization options)

MQZIO_PRIMARY	0	X'00000000'
MQZIO_SECONDARY	1	X'00000001'

MQZNS_* (Name service version)

MQZNS_VERSION_1	1	X'00000001'

MQZTO_* (Termination options)

MQZTO_PRIMARY	0	X'00000000'
MQZTO_SECONDARY	1	X'00000001'

MQZUS_* (Userid service version)

MQZUS_VERSION_1	1	X'00000001'

Appendix C. Header, COPY, and INCLUDE files

Various header, COPY, and INCLUDE files are provided to assist applications with the processing of:

- Event messages
- PCF commands and responses
- Installable services

These are described below for each of the supported programming languages. Not all of the files are available in all environments.

C header files

The following header files are provided for the C programming language.

Table 26. C header files

Filename	Contents relating to this book
CMQC	Elementary data types, some named constants for events and PCF commands
CMQCFC	PCF structures, additional named constants for events and PCF commands
CMQXC	Named constants for events and PCF commands relating to channels
CMQZC	Function prototypes, data types, and named constants for installable services (available only on OS/2, UNIX systems, and Windows NT)

COBOL COPY files

The following COPY files are provided for the COBOL programming language. Two COPY files are provided for each structure; one COPY file has initial values, the other does not.

Table 27. COBOL COPY files

File name (with initial values)	File name (without initial values)	Contents relating to this book
CMQV	–	Some named constants for events and PCF commands (not available on DOS clients and Windows clients)
CMQCFV	–	Additional named constants for events and PCF commands (available only on MVS/ESA)
CMQXV	–	Named constants for events and PCF commands relating to channels (available only on MVS/ESA and OS/400)
CMQCFHV	CMQCFHL	Header structure for events and PCF commands (available only on MVS/ESA)
CMQCFINV	CMQCFINL	Single-integer parameter structure for events and PCF commands (available only on MVS/ESA)
CMQCFILV	CMQCFILL	Integer-list parameter structure for events and PCF commands (available only on MVS/ESA)
CMQCFSTV	CMQCFSTL	Single-string parameter structure for events and PCF commands (available only on MVS/ESA)
CMQCFSLV	CMQCFSLL	String-list parameter structure for events and PCF commands (available only on MVS/ESA)

PL/I INCLUDE files

The following INCLUDE files are provided for the PL/I programming language. These files are available only on AIX, MVS/ESA, OS/2, and Windows NT.

Table 28. PL/I INCLUDE files

Filename	Contents relating to this book
CMQP	Some named constants for events and PCF commands
CMQCFP	PCF structures, and additional named constants for events and PCF commands
CMQXP	Named constants for events and PCF commands relating to channels

System/390 Assembler COPY files

The following COPY files are provided for the System/390 Assembler programming language. These files are available only on MVS/ESA.

Table 29. System/390 Assembler COPY files

Filename	Contents relating to this book
CMQA	Some named constants for events and PCF commands
CMQCFA	Additional named constants for events and PCF commands
CMQXA	Named constants for events and PCF commands relating to channels
CMQCFHA	Header structure for events and PCF commands
CMQCFINA	Single-integer parameter structure for events and PCF commands
CMQCFILA	Integer-list parameter structure for events and PCF commands
CMQCFSTA	Single-string parameter structure for events and PCF commands
CMQCFSLA	String-list parameter structure for events and PCF commands

Appendix D. Notices

The following paragraph does not apply to any 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 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.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. 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. Subject to IBM's valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact Laboratory Counsel, MP151, IBM United Kingdom Laboratories, Hursley Park, Winchester, Hampshire, England SO21 2JN. Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, New York 10594, U.S.A.

Programming interface information

This book is intended to help you to write application programs that run under:

- MQSeries for AIX Version 5
- MQSeries for AS/400 Version 4 Release 2
- MQSeries for AT&T GIS UNIX Version 2 Release 2
- MQSeries for Digital OpenVMS Version 2 Release 2
- MQSeries for HP-UX Version 5
- MQSeries for MVS/ESA Version 1 Release 2
- MQSeries for OS/2 Warp Version 5
- MQSeries for SINIX and DC/OSx Version 2 Release 2
- MQSeries for SunOS Version 2 Release 2
- MQSeries for Sun Solaris Version 5
- MQSeries for Tandem NonStop Kernel Version 2 Release 2
- MQSeries for Windows NT Version 5
- MQSeries for Windows Version 2 Release 1

This book documents General-use Programming Interface and Associated Guidance Information provided by the MQSeries products listed above.

General-use Programming Interfaces allow the customer to write programs that obtain the services of the MQSeries products listed above.

Trademarks

The following terms are trademarks of the IBM Corporation in the United States or other countries or both:

AIX	AS/400	BookManager
C/400	CICS	IBM
MQ	MQSeries	MVS/ESA
NetView	OS/400	SAA

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

UNIX is a registered trademark in the United States and other countries licensed exclusively through X/Open Company Limited.

C-bus is a trademark of Corollary, Inc.

Microsoft, Windows, Windows NT, and the Windows logo are registered trademarks of Microsoft Corporation.

Java and HotJava are trademarks of Sun Microsystems, Inc.

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

Part 5. Glossary and Index

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 MVS/ESA. A complete list of MQSeries for MVS/ESA abend reason codes and their explanations is contained in the *MQSeries for MVS/ESA Messages and Codes* manual.

active log. See *recovery log*.

adapter. An interface between MQSeries for MVS/ESA and TSO, IMS, CICS, or batch address spaces. An adapter is an attachment facility that enables applications to access MQSeries services.

add-in task. A function provided by MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT that coordinates the passing of data between a Lotus Notes application and an MQSeries application.

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.

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 MVS/ESA, a component of the CICS adapter that handles unscheduled events occurring as a result of connection requests to MQSeries for MVS/ESA.

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 MVS address space that is connected to MQSeries for MVS/ESA.

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 MVS platform, CICS and IMS are examples of application environments.

application log. In Windows NT, a log that records significant application events.

application queue. A queue used by an application.

archive log. See *recovery log*.

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 open an MQSeries object.

authorization file. In MQSeries on UNIX systems, a file that provides security definitions for an object, a class of objects, or all classes of objects.

authorization service. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, a service that provides authority checking of commands and MQI calls for the user identifier associated with the command or call.

authorized program analysis report (APAR). A report of a problem caused by a suspected defect in a current, unaltered release of a program.

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*.

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 MVS/ESA
- A wrap-around inventory of all recent MQSeries for MVS/ESA activity

The BSDS is required if the MQSeries for MVS/ESA 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 MVS/ESA 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. (1) A time when significant information is written on the log. Contrast with *syncpoint*. (2) In MQSeries on UNIX systems, the point in time when a data record described in the log is the same as the data record in the queue. Checkpoints are generated automatically and are used during the system restart process.

CI. Control interval.

circular logging. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, the process of keeping all restart data in a ring of log files. Logging fills the first file in the ring and then moves on to the next, until all the files are full. At this point, logging goes back to the first file in the ring and starts again, if the space has been freed or is no longer needed. Circular logging is used during restart recovery, using the log to roll back transactions that were in progress when the system stopped. Contrast with *linear logging*.

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*.

coded character set identifier (CCSID). The name of a coded set of characters and their code point assignments.

command. In MQSeries, an instruction that can be carried out by the queue manager.

command prefix (CPF). In MQSeries for MVS/ESA, a character string that identifies the queue manager to which MQSeries for MVS/ESA commands are directed,

and from which MQSeries for MVS/ESA 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.

configuration file. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, a file that contains configuration information related to, for example, logs, communications, or installable services. Synonymous with *.ini file*. See also *stanza*.

connect. To provide a queue manager connection handle, which an application uses on subsequent MQI calls. The connection is made either by the MQCONN 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 command. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, a command that can be entered interactively from the operating system command line. Such a command requires only that the MQSeries product be installed; it does not require a special utility or program to run it.

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.

Control Language (CL). In MQSeries for AS/400, a language that can be used to issue commands, either at the command line or by writing a CL program.

controlled shutdown. See *quiesced shutdown*.

CPF. Command prefix.

D

DAE. Dump analysis and elimination.

data conversion interface (DCI). The MQSeries interface to which customer- or vendor-written programs that convert application data between different machine encodings and CCSIDs must conform. A part of the MQSeries Framework.

datagram. The simplest message that MQSeries supports. This type of message does not require a reply.

DCE. Distributed Computing Environment.

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.

dead-letter queue handler. An MQSeries-supplied utility that monitors a dead-letter queue (DLQ) and processes messages on the queue in accordance with a user-written rules table.

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 MVS/ESA before MQSeries for MVS/ESA has been started.

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 Computing Environment (DCE). Middleware that provides some basic services, making the development of distributed applications easier. DCE is defined by the Open Software Foundation (OSF).

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 MVS/ESA 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*.

dump analysis and elimination (DAE). An MVS service that enables an installation to suppress SVC dumps and ABEND SYSUDUMP dumps that are not needed because they duplicate previously written dumps.

dynamic queue. A local queue created when a program opens a model queue object. See also *permanent dynamic queue* and *temporary dynamic queue*.

E

environment. See *application environment*.

ESM. External security manager.

ESTAE. Extended specify task abnormal exit.

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 log. See *application log*.

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.

Event Viewer. A tool provided by Windows NT to examine and manage log files.

extended specify task abnormal exit (ESTAE). An MVS macro that provides recovery capability and gives control to the specified exit routine for processing, diagnosing an abend, or specifying a retry address.

external security manager (ESM). A security product that is invoked by the MVS System Authorization Facility. RACF is an example of an ESM.

F

FFST. First Failure Support Technology.

FIFO. First-in-first-out.

First Failure Support Technology (FFST). Used by MQSeries on UNIX systems, MQSeries for OS/2 Warp, MQSeries for Windows NT, and MQSeries for AS/400 to detect and report software problems.

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 MVS/ESA, regardless of the state of any currently active tasks. Contrast with *quiesced shutdown*.

Framework. In MQSeries, a collection of programming interfaces that allow customers or vendors to write programs that extend or replace certain functions provided in MQSeries products. The interfaces are:

- MQSeries data conversion interface (DCI)
- MQSeries message channel interface (MCI)
- MQSeries name service interface (NSI)
- MQSeries security enabling interface (SEI)
- MQSeries trigger monitor interface (TMI)

FRR. Functional recovery routine.

functional recovery routine (FRR). An MVS recovery/termination manager facility that enables a recovery routine to gain control in the event of a program interrupt.

G

GCPC. Generalized command preprocessor.

generalized command preprocessor (GCPC). An MQSeries for MVS/ESA component that processes MQSeries commands and runs them.

Generalized Trace Facility (GTF). An MVS 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. See also *browse*.

global trace. An MQSeries for MVS/ESA trace option where the trace data comes from the entire MQSeries for MVS/ESA subsystem.

GTF. Generalized Trace Facility.

H

handle. See *connection handle* and *object handle*.

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*.

in-doubt unit of recovery. In MQSeries for MVS/ESA, the status of a unit of recovery for which a syncpoint has been requested but not yet performed.

.ini file. See *configuration file*.

initialization input data sets. Data sets used by MQSeries for MVS/ESA 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.

installable services. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, additional functionality provided as independent components. The installation of each component is optional: in-house or third-party components can be used instead. See also *authorization service*, *name service*, and *user identifier service*.

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 MVS 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

linear logging. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, the process of keeping restart data in a sequence of files. New files are added to the sequence as necessary. The space in which the data is written is not reused until the queue manager is restarted. Contrast with *circular logging*.

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.

locale. On UNIX systems, a subset of a user's environment that defines conventions for a specific culture (such as time, numeric, or monetary formatting and character classification, collation, or conversion). The queue manager CCSID is derived from the locale of the user ID that created the queue manager.

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.

log. In MQSeries, a file recording the work done by queue managers while they receive, transmit, and deliver messages.

log control file. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, the file containing information needed to monitor the use of log files (for example, their size and location, and the name of the next available file).

log file. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, a file in which all significant changes to the data controlled by a queue manager are recorded. If the primary log files become full, MQSeries allocates secondary log files.

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.

mail-in database. A Lotus Notes database for sole use by the add-in task. It holds the request from a Lotus Notes application before the request is passed to the MQSeries application.

MCA. Message channel agent.

MCI. Message channel interface.

media image. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, the sequence of log records that contain an image of an object. The object can be recreated from this image.

message. (1) In message queuing applications, a communication sent between programs. See also *persistent message* and *nonpersistent message*. (2) 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 and a receiver) 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.

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.

MQSeries client. Part of an MQSeries product that can be installed on a system without installing the full queue manager. The MQSeries client accepts MQI calls from applications and communicates with a queue manager on a server system.

MQSeries commands (MQSC). Human readable commands, uniform across all platforms, that are used to manipulate MQSeries objects. Contrast with *programmable command format (PCF)*.

N

namelist. An MQSeries for MVS/ESA object that contains a list of queue names.

name service. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, the facility that determines which queue manager owns a specified queue.

name service interface (NSI). The MQSeries interface to which customer- or vendor-written programs that resolve queue-name ownership must conform. A part of the MQSeries Framework.

name transformation. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, an internal process that changes a queue manager name so that it is unique and valid for the system being used. Externally, the queue manager name remains unchanged.

New Technology File System (NTFS). A Windows NT recoverable file system that provides security for files.

nonpersistent message. A message that does not survive a restart of the queue manager. Contrast with *persistent message*.

NSI. Name service interface.

NTFS. New Technology File System.

null character. The character that is represented by X'00'.

O

OAM. Object authority manager.

object. In MQSeries, an object is a queue manager, a queue, a process definition, a channel, a namelist (MVS/ESA only), or a storage class (MVS/ESA only).

object authority manager (OAM). In MQSeries on UNIX systems and MQSeries for Windows NT, the default authorization service for command and object management. The OAM can be replaced by, or run in combination with, a customer-supplied security service.

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 MVS/ESA, an automatic process whereby a queue manager's active log is transferred to its archive log.

output log-buffer. In MQSeries for MVS/ESA, 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 MVS/ESA moves data (for example, queues and messages) from buffers in main storage to permanent backing storage (DASD).

PCF. Programmable command format.

PCF command. See *programmable command format*.

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 MVS/ESA, the term used to describe a set of backup copies of MQSeries for MVS/ESA 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*.

principal. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, a term used for a user identifier. Used by the object authority manager for checking authorizations to system resources.

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.

programmable command format (PCF). A type of MQSeries message used by:

- User administration applications, to put PCF commands onto the system command input queue of a specified queue manager
- User administration applications, to get the results of a PCF command from a specified queue manager
- A queue manager, as a notification that an event has occurred

Contrast with *MQSC*.

program temporary fix (PTF). A solution or by-pass of a problem diagnosed by IBM field engineering as the result of a defect in a current, unaltered release of a program.

PTF. Program temporary fix.

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. (1) 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*. (2) 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.

queuing. See *message queuing*.

quiesced shutdown. (1) In MQSeries, a shutdown of a queue manager that allows all connected applications to disconnect. Contrast with *immediate shutdown* and *preemptive shutdown*. (2) 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 MVS/ESA, data sets containing information needed to recover messages, queues, and the MQSeries subsystem. MQSeries for MVS/ESA 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*.

recovery termination manager (RTM). A program that handles all normal and abnormal termination of tasks by passing control to a recovery routine associated with the terminating function.

Registry. In Windows NT, a secure database that provides a single source for system and application configuration data.

Registry Editor. In Windows NT, the program item that allows the user to edit the Registry.

Registry Hive. In Windows NT, the structure of the data stored in the Registry.

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.

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.

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.

RESLEVEL. In MQSeries for MVS/ESA, an option that controls the number of CICS user IDs checked for API-resource security in MQSeries for MVS/ESA.

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 MVS/ESA, 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.

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*.

RTM. Recovery termination manager.

rules table. A control file containing one or more rules that the dead-letter queue handler applies to messages on the DLQ.

S

SAF. System Authorization Facility.

SDWA. System diagnostic work area.

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 MVS/ESA, 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.

shutdown. See *immediate shutdown*, *preemptive shutdown*, and *quiesced shutdown*.

signaling. In MQSeries for MVS/ESA 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 MVS/ESA 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.

stanza. A group of lines in a configuration file that assigns a value to a parameter modifying the behavior of a queue manager, client, or channel. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, a configuration (.ini) file may contain a number of stanzas.

storage class. In MQSeries for MVS/ESA, 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 MVS, a group of modules that provides function that is dependent on MVS. For example, MQSeries for MVS/ESA is an MVS subsystem.

supervisor call (SVC). An MVS 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 MVS/ESA, 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.

symptom string. Diagnostic information displayed in a structured format designed for searching the IBM software support database.

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.

System Authorization Facility (SAF). An MVS facility through which MQSeries for MVS/ESA 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 diagnostic work area (SDWA). Data recorded in a SYS1.LOGREC entry, which describes a program or hardware error.

system initialization table (SIT). A table containing parameters used by CICS on start up.

SYS1.LOGREC. A service aid containing information about program and hardware errors.

T

TACL. Tandem Advanced Command Language.

target library high-level qualifier (thlqual). High-level qualifier for MVS/ESA target data set names.

task control block (TCB). An MVS control block used to communicate information about tasks within an address space that are connected to an MVS subsystem such as MQSeries for MVS/ESA or CICS.

task switching. The overlapping of I/O operations and processing between several tasks. In MQSeries for MVS/ESA, 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 MVS/ESA.

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). See also *global trace* and *performance trace*.

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

UIS. User identifier service.

undelivered-message queue. See *dead-letter queue*.

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*.

user identifier service (UIS). In MQSeries for OS/2 Warp, the facility that allows MQI applications to associate a user ID, other than the default user ID, with MQSeries messages.

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.

Index

D

E

F

G

H

I

L

M

N

O

P

Q

R

S

T

U

V

W

X

Sending your comments to IBM

MQSeries

Programmable System Management

SC33-1482-06

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 request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should 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, use the Readers' Comment Form
- 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: WINVMD(IDRCF)
  - Internet: idrcf@winvmd.vnet.ibm.com

Whichever you use, ensure that you include:

- The publication number and title
- The page number or topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.

Readers' Comments

MQSeries

Programmable System Management

SC33-1482-06

Use this form to tell us what you think about this manual. If you have found errors in it, or if you want to express your opinion about it (such as organization, subject matter, appearance) or make suggestions for improvement, this is the form to use.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer. This form is provided for comments about the information in this manual and the way it is presented.

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.

Be sure to print your name and address below if you would like a reply.

Name	Address
Company or Organization	
Telephone	Email

You can send your comments POST FREE on this form from any one of these countries:

Australia	Finland	Iceland	Netherlands	Singapore	United States
Belgium	France	Israel	New Zealand	Spain	of America
Bermuda	Germany	Italy	Norway	Sweden	
Cyprus	Greece	Luxembourg	Portugal	Switzerland	
Denmark	Hong Kong	Monaco	Republic of Ireland	United Arab Emirates	

If your country is not listed here, your local IBM representative will be pleased to forward your comments to us. Or you can pay the postage and send the form direct to IBM (this includes mailing in the U.K.).

2 Fold along this line

By air mail
Par avion

IBRS/CCRI NUMBER: PHQ - D/1348/SO

NE PAS AFFRANCHIR

NO STAMP REQUIRED

REPONSE PAYEE
GRANDE-BRETAGNE

IBM United Kingdom Laboratories
Information Development Department (MP095)
Hursley Park,
WINCHESTER, Hants
SO21 2ZZ United Kingdom

3 Fold along this line

From: Name _____

Company or Organization _____

Address _____

_____

EMAIL _____

Telephone _____

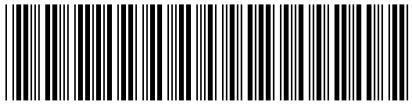
4 Fasten here with adhesive tape

Cut along this line

Cut along this line

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

SC33-1482-06



Spine information:

MQSeries Programmable System Management