WebSphere MQ

IBM

Programmable Command Formats and Administration Interface

SC34-6060-03

Note!
Before using this information and the product it supports, be sure to read the general information under Appendix H, "Notices", on page 391.

Fourth edition (March 2003)

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

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

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

Contents

Figures

Tables

About this book

The first section of this book describes the facilities available on WebSphere® MQ products for writing programs using the WebSphere MQ Programmable Command Formats (PCFs) to administer IBM® WebSphere MQ systems either locally or remotely.

The second section of this book describes the administration interface for WebSphere MQ. This part of the product is referred to as the *WebSphere MQ Administration Interface (MQAI)*.

The MQAI is a programming interface that simplifies the use of PCF messages to configure WebSphere MQ.

The term UNIX® systems is used to denote the following UNIX operating systems, unless otherwise stated:
- AIX®
- AT&T GIS (NCR) UNIX
- Compaq Tru64 UNIX
- HP-UX
- Linux (for Intel and zSeries™)
- SINIX and DC/OSx
- Solaris (SPARC and Intel Platform Editions)

The term Windows® is used throughout this book to denote the following Windows operating systems, unless stated otherwise:
- Windows NT®
- Windows 2000
- Windows XP

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

WebSphere MQ product	**Windows 95**	**Windows 98**	**Windows NT**	**Windows 2000**	**Windows XP**
WebSphere MQ for Windows client	✓	✓	✓	✓	✓
WebSphere MQ for Windows	No	No	✓	✓	✓

Who this book is for

Primarily, this book is for system programmers who write programs to monitor and administer WebSphere MQ products.

What you need to know to understand this book

For the first section of this book (PCFs) you need:
- Experience in writing systems management applications
- An understanding of the Message Queue Interface (MQI)
- Experience of WebSphere MQ programs in general, or familiarity with the content of the other books in the WebSphere MQ library.

For the second section of this book (MQAI) you need:

- Some knowledge of WebSphere MQ
- Knowledge of how to write programs in the C programming language or in Visual Basic for Windows.

How to use this book

The first part of this book describes PCFs.

PCFs are the formats of command and response messages that are sent between a WebSphere MQ systems management application, or other program, and a WebSphere MQ queue manager.

Chapter 1, “Introduction to Programmable Command Formats”, on page 5 and Chapter 2, “Using Programmable Command Formats”, on page 9 contain introduction and guidance material. You are advised to read both of these chapters.

The reference material starts in Chapter 3, “Definitions of the Programmable Command Formats”, on page 17. See Chapter 12, “Example of using PCFs”, on page 221 for an example of how PCFs can be used.

The second part of this book describes the MQAI.

The first four chapters introduce the Message Queuing Administration Interface and tell you how to use it.

Chapter 17, “MQAI reference”, on page 255 contains the reference information.

Chapter 18, “Examples of using the MQAI”, on page 315 provides some example programs.

Chapter 19, “Advanced topics”, on page 335 describes indexing, data conversion, and the message descriptor.

Appendixes

The error codes that apply to PCF commands and responses are listed in Appendix A, “Error codes”, on page 341.

The values of PCF constants are given in Appendix B, “MQ constants”, on page 357.

The various header, COPY, and INCLUDE files that are provided to assist applications with the processing of PCF commands are identified in Appendix C, “Header, COPY, and INCLUDE files”, on page 377.

The MQAI return codes are given in Appendix D, “MQAI Return codes”, on page 381

The MQAI constants are given in Appendix E, “MQAI Constants in C”, on page 383

The MQAI header files are given in Appendix F, “MQAI Header files”, on page 387

The MQAI user and system selectors are given in Appendix G, “MQAI Selectors”, on page 389

Summary of changes

This section describes changes in this edition of *WebSphere MQ Programmable Command Formats and Administration Interface*. Changes since the previous edition of the book are marked by vertical lines to the left of the changes.

Changes for this edition (SC34-6060-03)

This edition adds the new CommandLevel constant MQCMDL_LEVEL_531, and makes some corrections.

Changes for the previous edition (SC34-6060-02)

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

Changes for the earlier editions (SC34-6060-00 and -01)

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

- Changes throughout the book to reflect the rebranding of MQSeries to WebSphere MQ.
- Adding the platforms Windows XP, Linux for zSeries, and Linux for Intel.
- New SSL error codes in Change, Copy, and Create Channel
- New error codes in Change Queue Manager
- New open parameters in Inquire Queue Status (Response)
- New MQIACF constants
- New MQQSOT constants
- New MQQSUM constants
- New MQQSO constants
- New MQRCCF constants
- The Programmable Command Formats manual has been merged with the Administration Interface Programming Guide and Reference.
- Authentication Information commands have been introduced for SSL "Change, Copy, and Create Authentication Information Object" on page 21, "Delete Authentication Information Object" on page 76, "Inquire Authentication Information Object" on page 82, "Inquire Authentication Information Object (Response)" on page 83, "Inquire Authentication Information Object Names" on page 84, "Inquire Authentication Information Object Names (Response)" on page 85
- SSL parameters have been introduced.
- A new PCF parameter (LocalAddress) has been introduced.
- A new PCF parameter (BatchHeartbeat) has been introduced.
- A new PCF parameter (ConfigurationEvent) has been introduced.
- A new PCF parameter (RemoteQMgrName) has been introduced.
- A new PCF parameter (NameCount) has been introduced.

Changes

- New PCF commands Inquire Queue Status "Inquire Queue Status (Response)" on page 168 and Inquire Queue Status (Response) "Inquire Queue Status (Response)" on page 168 have been introduced.
- The Refresh Cluster command has been updated."Refresh Cluster" on page 173
- The Reset Cluster command has been updated."Reset Cluster" on page 177
- The Stop Channel command has been updated. "Stop Channel" on page 186
- The Suspend Queue Manager Cluster command has been updated. "Suspend Queue Manager Cluster" on page 189
- A new parameter structure MQCFBS - PCF byte string parameter has been introduced. Chapter 11, "MQCFBS — PCF byte string parameter", on page 219

Part 1. Programmable Command Formats

Chapter 1. Introduction to Programmable Command Formats

This chapter introduces WebSphere MQ Programmable Command Formats (PCFs) and their relationship to other parts of the WebSphere MQ products. It includes:
- "The problem PCF commands solve"
- "What PCFs are"
- "Other administration interfaces" on page 6
- "The WebSphere MQ Administration Interface (MQAI)" on page 7

The Programmable Command Formats described in this book are supported by:

WebSphere MQ for AIX, V5.3
WebSphere MQ for HP-UX, V5.3
WebSphere MQ for iSeries™, V5.3
WebSphere MQ for Linux for Intel and Linux for zSeries, V5.3
WebSphere MQ for Solaris, V5.3
WebSphere MQ for Windows, V5.3
WebSphere MQ for z/OS™, V5.3
MQSeries for OS/2® Warp, V5.3
MQSeries for Compaq NonStop Kernel, V5.1
| MQSeries for Compaq OpenVMS Alpha, V5.1
MQSeries for Sun Solaris, Intel Platform Edition, V5.1

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.

WebSphere MQ 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 WebSphere MQ 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 administration interfaces

Administration of WebSphere MQ objects can be carried out in other ways.

WebSphere MQ for iSeries

In addition to PCFs, there are two further administration interfaces:

OS/400® Control Language (CL)

This can be used to issue administration commands to WebSphere MQ for iSeries. 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 servers 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.

WebSphere MQ Commands (MQSC)

These provide a uniform method of issuing commands across WebSphere MQ platforms. The general format of the commands is shown in the *WebSphere MQ Script (MQSC) Command Reference* manual.

To issue the commands on an iSeries server, you can either:

1. Create a list of commands in a Script file, and then run the file using the STRMQMMQSC command, or
2. Use the **runmqsc** command from a QSHELL, and issue the MQSC commands interactively.

MQSC responses are designed to be human readable, whereas PCF command and response formats are intended for program use.

Note: MQSC responses to commands issued from a script file are returned in a spool file.

WebSphere MQ for z/OS

WebSphere MQ for z/OS supports the WebSphere MQ commands (MQSC). With z/OS these commands can be entered from the z/OS console, or sent to the system command input queue. More information about issuing the commands is given in the *WebSphere MQ Script (MQSC) Command Reference* manual, and in the *WebSphere MQ for z/OS System Administration Guide*.

Note: PCF commands are not supported by WebSphere MQ for z/OS.

MQSeries for Compaq NonStop Kernel, V5.1

In addition to PCFs, there are three further administrative interfaces:

- WebSphere MQ commands (MQSC)
- Control commands
- Message Queue Management (MQM) facility

  MQSeries for Compaq NonStop Kernel, V5.1 provides a panel interface for some of the functions.

WebSphere MQ for Windows, and UNIX systems and MQSeries for Compaq OpenVMS Alpha and OS/2

In addition to PCFs, there are four further administrative interfaces:

WebSphere MQ commands (MQSC)

You can use the MQSC as single commands issued at the Windows, or UNIX system command line. To issue more complicated, or multiple commands, the MQSC can be built into a file that you run from the Windows, or UNIX system command line. MQSC can be sent to a remote queue manager. For full details see the *WebSphere MQ Script (MQSC) Command Reference* manual.

Control commands

WebSphere MQ for Windows, and UNIX systems provides another type of command for some of the functions. These are the *control commands* that you issue at the system command line. Reference material for these commands is contained in the *WebSphere MQ System Administration Guide* manual.

WebSphere MQ Explorer (Windows only)

The WebSphere MQ Explorer is an application that runs under the Microsoft® Management Console (MMC). It provides a graphical user interface for controlling resources in a network. For full details see the *WebSphere MQ System Administration Guide* manual.

The WebSphere MQ Administration Interface (MQAI)

| In addition to the methods described in “Other administration interfaces” on
| page 6, WebSphere MQ for Windows, AIX, iSeries, Linux, HP-UX, and Solaris and
| MQSeries for OS/2 Warp support the WebSphere MQ Administration Interface
| (MQAI).

The MQAI is a programming interface to WebSphere MQ that gives you an alternative to the MQI, for sending and receiving PCFs. The MQAI uses *data bags* which allow you to handle properties (or parameters) of objects more easily than using PCFs directly via the MQI.

The MQAI provides easier programming access to PCF messages by passing parameters into the data bag, so that only one statement is required for each structure. This removes the need for the programmer to handle arrays and allocate storage, and provides some isolation from the details of PCF.

The MQAI administers WebSphere MQ by sending PCF messages to the command server and waiting for a response.

The MQAI is described in the second section of this manual. See the *WebSphere MQ Using Java* book for a description of a component object model interface to the MQAI.

Other administration

Chapter 2. Using Programmable Command Formats

This chapter describes how to use the PCFs in a systems management application program for WebSphere MQ remote administration. The chapter includes:

- "PCF command messages"
- "Responses" on page 11
- "Authority checking for PCF commands" on page 13

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 Chapter 6, "MQCFH - PCF header", on page 193). 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 377.

Message descriptor for a PCF command

The WebSphere MQ message descriptor is fully documented in the *WebSphere MQ Application Programming Reference* manual.

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 iSeries, OS/2, Windows 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 iSeries, OS/2, Windows, or UNIX systems, set this field to the coded character-set identifier 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 MQCI_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 Chapter 6, “MQCFH - PCF header”, on page 193 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 might return a structure even when it is not requested. This is shown in the definition of the response (Chapter 3) as *always returned*. The reason that, for these responses, it is necessary to name the objects in the response to identify 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 might follow as described below.

Error response

If the command has an error, one or more error response messages are sent (more than one might be sent even for a command that would normally have only 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 that 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 a duplicate selector or one that is not valid. 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 3, “Definitions of the Programmable Command Formats”, on page 17.

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 WebSphere MQ 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.

WebSphere MQ for iSeries

In order to process any PCF command, the user ID must have *dsp* authority for the WebSphere MQ object on the target system.

In addition, WebSphere MQ object authority checks are performed for certain PCF commands, as shown in Table 1 on page 15.

In most cases these are the same checks as those performed by the equivalent WebSphere MQ CL commands issued on a local system. See the *WebSphere MQ for iSeries, V5.3 System Administration* book for more information on the mapping from WebSphere MQ authorities to OS/400 system authorities, and the authority requirements for the WebSphere MQ CL commands. Details of security concerning exits are given in the *WebSphere MQ Intercommunication* manual.

To process any of the following commands the user ID must be a member of the group profile QMQMADM:
- Ping Channel
- Change Channel
- Copy Channel
- Create Channel
- Delete Channel
- Reset Channel
- Resolve Channel
- Start Channel
- Stop Channel
- Start Channel Initiator
- Start Channel Listener

WebSphere MQ for Windows, 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, WebSphere MQ object authority checks are performed for certain PCF commands, as shown in Table 1 on page 15.

To process any of the following commands the user ID must belong to group *mqm*.

Note: For Windows **only,** the user ID can belong to group *Administrators* or group *mqm*.

- Change Channel
- Copy Channel
- Create Channel
- Delete Channel
- Ping Channel
- Reset Channel
- Start Channel
- Stop Channel
- Start Channel Initiator
- Start Channel Listener
- Resolve Channel
- Reset Cluster
- Refresh Cluster
- Suspend Queue Manager
- Resume Queue Manager

MQSeries for Compaq OpenVMS Alpha and Compaq NSK

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, WebSphere MQ object authority checks are performed for certain PCF commands, as shown in Table 1 on page 15.

To process any of the following commands the user ID must belong to group *mqm*:

- Change Channel
- Copy Channel
- Create Channel
- Delete Channel
- Ping Channel
- Reset Channel
- Start Channel
- Stop Channel
- Start Channel Initiator
- Start Channel Listener
- Resolve Channel
- Reset Cluster
- Refresh Cluster
- Suspend Queue Manager
- Resume Queue Manager

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.

Table 1. Windows, Compaq OpenVMS Alpha, Compaq NSK, and UNIX systems - object authorities

Command	**WebSphere MQ object authority**	**Class authority (for object type)**
Change Authentication Information	chg	n/a
Change Namelist	chg	n/a
Change Queue	chg	n/a
Change Queue Manager	chg	n/a
Change Process	chg	n/a
Clear Queue	clr	n/a
Copy Authentication Information	*from:* dsp	crt
Copy Authentication Information (Replace) *see Note 1*	*from:* dsp *to:* chg	n/a
Copy Namelist	*from:* dsp	crt
Copy Namelist (Replace) *see Note 1*	*from:* dsp *to:* chg	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 Authentication Information	*(system default authentication information)* dsp	crt
Create Authentication Information (Replace) *see Note 1*	*(system default authentication information)* dsp *to:* chg	n/a
Create Namelist	*(system default namelist)* dsp	crt
Create Namelist (Replace) *see Note 1*	*(system default namelist)* 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 Authentication Information	dlt	n/a
Delete Namelist	dlt	n/a
Delete Process	dlt	n/a
Delete Queue	dlt	n/a

Table 1. Windows, Compaq OpenVMS Alpha, Compaq NSK, and UNIX systems - object authorities (continued)

Command	WebSphere MQ object authority	Class authority (for object type)
Inquire Authentication Information	dsp	n/a
Inquire Namelist	dsp	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
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.

WebSphere MQ 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 *WebSphere MQ Intercommunication* manual.

Chapter 3. Definitions of the Programmable Command Formats

The chapter discusses:
- "How the definitions are shown"
- "PCF commands and responses in groups" on page 19

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 Chapter 6, "MQCFH - PCF header", on page 193 for details of the command identifier.

Notes:

1. The PCFs listed in "PCF commands and responses in groups" on page 19 are available on all platforms to which this book applies, unless specific limitations are shown at the start of a structure.
2. WebSphere MQ Version 5.3 products can use the WebSphere MQ Administration Interface (MQAI), which provides a simplified way for applications written in the C and Visual Basic programming language to build and send PCF commands.

   On WebSphere MQ for Windows, V5.3 you can use the Microsoft Active Directory Services Interface (ADSI), as well as PCFs, to inquire about and set parameters.

   For information on the MQAI see the second section of this manual, and for information on using Microsoft ADSI see the *WebSphere MQ for Windows, V5.3 Using the Component Object Model Interface* book.

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 5, "Structures used for commands and responses", on page 191). 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, “MQ constants”, on page 357.

Error codes

At the end of each command format definition there is a list of error codes that might be returned by that command. Full descriptions are given in the alphabetic list in Appendix A, “Error codes”, on page 341.

Error codes applicable to all commands

In addition to those listed under each command format, any command might return the following in the response format header (descriptions of the MQRC_* error codes are given in the *WebSphere MQ Application Programming Reference* manual):

Reason (MQLONG)
The value can be:

MQRC_NONE
(0, X’000’) No reason to report.

MQRC_MSG_TOO_BIG_FOR_Q
(2030, X’7EE’) Message length greater than maximum for queue.

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.

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_SEQ_NUMBER_ERROR
Message sequence number not valid.

MQRCCF_MSG_TRUNCATED
Message truncated.

MQRCCF_MSG_LENGTH_ERROR
Message length not valid.

MQRCCF_COMMAND_FAILED
Command failed.

PCF commands and responses in groups

The commands and data responses are given in alphabetic order in this book.

They can be usefully grouped as follows:

Authentication Information commands

"Change, Copy, and Create Authentication Information Object" on page 21
"Delete Authentication Information Object" on page 76
"Inquire Authentication Information Object" on page 82

Queue Manager commands

"Change Queue Manager" on page 66
"Inquire Queue Manager" on page 151
"Ping Queue Manager" on page 173

Namelist commands

"Change, Copy, and Create Namelist" on page 47
"Delete Namelist" on page 77
"Inquire Namelist" on page 129
"Inquire Namelist Names" on page 132

Process commands

"Change, Copy, and Create Process" on page 49
"Delete Process" on page 78
"Inquire Process" on page 133
"Inquire Process Names" on page 136

Queue commands

"Change, Copy, and Create Queue" on page 53
"Clear Queue" on page 75
"Delete Queue" on page 79
"Inquire Queue" on page 137
"Inquire Queue Names" on page 163

Channel commands

"Change, Copy and Create Channel" on page 23
"Delete Channel" on page 76
"Inquire Channel" on page 85
"Inquire Channel Names" on page 103
"Inquire Channel Status" on page 105
"Ping Channel" on page 170
"Reset Channel" on page 175
"Resolve Channel" on page 181
"Start Channel" on page 183
"Start Channel Initiator" on page 185

"Start Channel Listener" on page 186
"Stop Channel" on page 186

Statistics command

"Reset Queue Statistics" on page 179

Escape command

"Escape" on page 81

Cluster commands

"Inquire Cluster Queue Manager" on page 116
"Refresh Cluster" on page 173
"Reset Cluster" on page 177
"Resume Queue Manager Cluster" on page 182
"Suspend Queue Manager Cluster" on page 189

Data responses to commands

"Escape (Response)" on page 81
"Inquire Authentication Information Object (Response)" on page 83
"Inquire Channel (Response)" on page 95
"Inquire Channel Names (Response)" on page 105
"Inquire Channel Status (Response)" on page 112
"Inquire Cluster Queue Manager (Response)" on page 120
"Inquire Namelist (Response)" on page 131
"Inquire Namelist Names (Response)" on page 133
"Inquire Process (Response)" on page 135
"Inquire Process Names (Response)" on page 137
"Inquire Queue (Response)" on page 145
"Inquire Queue Manager (Response)" on page 154
"Inquire Queue Names (Response)" on page 165
"Reset Queue Statistics (Response)" on page 180

Chapter 4. Definitions of Programmable Command Formats

This chapter contains reference material for the Programmable Command Formats (PCFs) of commands and responses sent between a WebSphere MQ systems management application program and a WebSphere MQ queue manager.

Change, Copy, and Create Authentication Information Object

Note: These commands are supported only on the WebSphere MQ Version 5.3 platforms: AIX, HP-UX, Linux, z/OS, Solaris, Windows, and iSeries.

The Change authentication information (MQCMD_CHANGE_AUTH_INFO) command changes the specified attributes in an authentication information object. For any optional parameters that are omitted, the value does not change.

The Copy authentication information (MQCMD_COPY_AUTH_INFO) command creates a new authentication information object using, for attributes not specified in the command, the attribute values of an existing authentication information object.

The Create authentication information (MQCMD_CREATE_AUTH_INFO) command creates an authentication information object. Any attributes that are not defined explicitly are set to the default values on the destination queue manager. A system default authentication information object exists and default values are taken from it.

Required parameters (Change authentication information):
AuthInfoType

Required parameters (Copy authentication information):
FromAuthInfoName, *ToAuthInfoName*, *AuthInfoType*

Required parameters (Create authentication information):
AuthInfoName, *AuthInfoType*, *AuthInfoConnName*

Optional parameters:
LDAPUserName, *LDAPPassword*, *AuthInfoDesc*

Required parameters (Change authentication information)

AuthInfoType (MQCFIN)
The type of authentication information object (parameter identifier: MQIA_AUTH_INFO_TYPE).

The value can be:

MQAIT_CRL_LDAP
This defines this authentication information object as specifying Certificate Revocation Lists that are held on the LDAP. Please see the *WebSphere MQ Security* book for more information.

Required parameters (Copy authentication information)

AuthInfoType (MQCFIN)
The type of authentication information object (parameter identifier: MQIA_AUTH_INFO_TYPE).

The value can be:

MQAIT_CRL_LDAP
: This defines this authentication information object as specifying Certificate Revocation Lists that are held on the LDAP. Please see the *WebSphere MQ Security* book for more information.

FromAuthInfoName (MQCFST)
: The name of the authentication information object definition to be copied from (parameter identifier: MQCACF_FROM_AUTH_INFO_NAME).

  The maximum length of the string is MQ_AUTH_INFO_NAME_LENGTH.

ToAuthInfoName (MQCFST)
: The name of the authentication information object to copy to (parameter identifier: MQCACF_TO_AUTH_INFO_NAME).

  The maximum length of the string is MQ_AUTH_INFO_NAME_LENGTH.

Required parameters (Create authentication information)

AuthInfoName (MQCFST)
: authentication information object name (parameter identifier: MQCA_AUTH_INFO_NAME).

  The maximum length of the string is MQ_AUTH_INFO_NAME_LENGTH.

AuthInfoConnName (MQCFST)
: The connection name of the authentication information object (parameter identifier: MQCA_AUTH_INFO_CONN_NAME).

  The maximum length of the string is MQ_AUTH_INFO_CONN_NAME_LENGTH.

AuthInfoType (MQCFIN)
: The type of authentication information object (parameter identifier: MQIA_AUTH_INFO_TYPE).

  The value can be:

  MQAIT_CRL_LDAP
  : This defines this authentication information object as specifying Certificate Revocation Lists that are held on the LDAP. Please see the *WebSphere MQ Security* book for more information.

Optional parameters

LDAPUserName (MQCFST)
: The LDAP user name (parameter identifier: MQCA_LDAP_USER_NAME).

  The maximum length is MQ_DISTINGUISHED_NAME_LENGTH.

LDAPPassword (MQCFST)
: The LDAP password (parameter identifier: MQCA_LDAP_PASSWORD).

  The maximum length is MQ_LDAP_PASSWORD_LENGTH.

AuthInfoDesc (MQCFST)
: The description of the authentication information object(parameter identifier: MQCA_AUTH_INFO_DESC).

  The maximum length is MQ_AUTH_INFO_DESC_LENGTH.

Change, Copy and Create Channel

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.

The channel commands are supported on all platforms.

Required parameters :
ChannelName, *ChannelType*

Optional parameters (any ChannelType):
ChannelDesc, *SecurityExit*, *SendExit*, *ReceiveExit*, *MaxMsgLength*, *SecurityUserData*, *SendUserData*, *ReceiveUserData*, *SSLCipherSpec*, *SSLPeerName*, *TransportType*

Optional parameters (sender ChannelType):
ModeName, *TpName*, *MCAName*, *BatchSize*, *DiscInterval*, *ShortRetryCount*, *ShortRetryInterval*, *LongRetryCount*, *LongRetryInterval*, *SeqNumberWrap*, *DataConversion*, *MCAType*, *MCAUserIdentifier*, *UserIdentifier*, *Password*, *HeartbeatInterval*, *NonPersistentMsgSpeed*, *BatchInterval*, *LocalAddress*, *BatchHeartbeat*, *MsgExit*, *MsgUserData*, *XmitQName*, *ConnectionName*

Optional parameters (server ChannelType):
ModeName, *TpName*, *MCAName*, *BatchSize*, *DiscInterval*, *ShortRetryCount*, *ShortRetryInterval*, *LongRetryCount*, *LongRetryInterval*, *SeqNumberWrap*, *DataConversion*, *MCAType*, *MCAUserIdentifier*, *UserIdentifier*, *Password*, *HeartbeatInterval*, *NonPersistentMsgSpeed*, *BatchInterval*, *LocalAddress*, *BatchHeartbeat*, *MsgExit*, *MsgUserData*, *XmitQName*, *ConnectionName*, *SSLClientAuth*

Optional parameters (receiver ChannelType):
BatchSize, *PutAuthority*, *SeqNumberWrap*, *MCAUserIdentifier*, *MsgRetryExit*, *MsgRetryUserData*, *MsgRetryCount*, *MsgRetryInterval*, *HeartbeatInterval*, *NonPersistentMsgSpeed*, *SSLClientAuth*, *MsgExit*, *MsgUserData*

Optional parameters (requester ChannelType):
ModeName, *TpName*, *MCAName*, *BatchSize*, *PutAuthority*, *SeqNumberWrap*, *MCAType*, *MCAUserIdentifier*, *UserIdentifier*, *Password*, *MsgRetryExit*, *MsgRetryUserData*, *MsgRetryCount*, *MsgRetryInterval* *HeartbeatInterval*, *NonPersistentMsgSpeed*, *SSLClientAuth*, *LocalAddress*, *MsgExit*, *MsgUserData*, *ConnectionName*

Optional parameters (server-connection ChannelType):
MCAUserIdentifier, *SSLClientAuth*, *HeartbeatInterval*, *PutAuthority*

Optional parameters (client-connection ChannelType):
ModeName, *TpName* *QMgrName*, *UserIdentifier*, *Password*, *LocalAddress*,*HeartbeatInterval*, *ConnectionName*

Optional parameters (cluster-receiver ChannelType):
ModeName, *TpName*, *DiscInterval*, *ShortRetryCount*, *ShortRetryInterval*, *LongRetryCount*, *LongRetryInterval*, *DataConversion*, *BatchSize*, *PutAuthority*, *SeqNumberWrap*, *MCAUserIdentifier*, *MsgRetryExit*, *MsgRetryUserData*, *MsgRetryCount*, *MsgRetryInterval*, *HeartbeatInterval*, *NonPersistentMsgSpeed*, *BatchInterval*, *ClusterName*, *ClusterNamelist*, *NetworkPriority*, *SSLClientAuth*, *LocalAddress*, *BatchHeartbeat*, *MCAType*, *MsgExit*, *MsgUserData*, *ConnectionName*

Optional parameters (cluster-sender ChannelType):
ModeName, *TpName*, *MCAName*, *BatchSize*, *DiscInterval*, *ShortRetryCount*, *ShortRetryInterval*, *LongRetryCount*, *LongRetryInterval*, *SeqNumberWrap*, *DataConversion*, *MCAType*, *MCAUserIdentifier*, *UserIdentifier*, *Password*, *HeartbeatInterval*, *NonPersistentMsgSpeed*, *BatchInterval*, *ClusterName*, *ClusterNamelist*, *LocalAddress*, *BatchHeartbeat*, *MsgExit*, *MsgUserData*, *ConnectionName*

Optional parameters (Change channel — requester, client-connection, cluster-sender, and cluster-receiver channel types):
ConnectionName

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.

The channel commands are supported on all platforms.

Required parameters :
FromChannelName, *ToChannelName*, *ChannelType*

Optional parameters (any ChannelType):
ChannelDesc, *SecurityExit*, *SendExit*, *ReceiveExit*, *MaxMsgLength*, *SecurityUserData*, *SendUserData*, *ReceiveUserData*, *SSLCipherSpec*, *SSLPeerName*

Optional parameters (sender and server ChannelTypes):
ModeName, *TpName*, *MCAName*, *BatchSize*, *DiscInterval*, *ShortRetryCount*, *ShortRetryInterval*, *LongRetryCount*, *LongRetryInterval*, *SeqNumberWrap*, *DataConversion*, *MCAType*, *MCAUserIdentifier*, *UserIdentifier*, *Password*, *HeartbeatInterval*, *NonPersistentMsgSpeed*, *BatchInterval*, *LocalAddress*, *BatchHeartbeat*

Optional parameters (receiver ChannelType):
BatchSize, *PutAuthority*, *SeqNumberWrap*, *MCAUserIdentifier*, *MsgRetryExit*, *MsgRetryUserData*, *MsgRetryCount*, *MsgRetryInterval*, *HeartbeatInterval*, *NonPersistentMsgSpeed*, *SSLClientAuth*

Optional parameters (requester ChannelType):
ModeName, *TpName*, *MCAName*, *BatchSize*, *PutAuthority*, *SeqNumberWrap*, *MCAType*, *MCAUserIdentifier*, *UserIdentifier*, *Password*, *MsgRetryExit*, *MsgRetryUserData*, *MsgRetryCount*, *MsgRetryInterval* *HeartbeatInterval*, *NonPersistentMsgSpeed*, *SSLClientAuth*, *LocalAddress*

Optional parameters (server-connection ChannelType):
MCAUserIdentifier, *SSLClientAuth*, *HeartbeatInterval*, *PutAuthority*

Optional parameters (client-connection ChannelType):
ModeName, *TpName* *QMgrName*, *UserIdentifier*, *Password*, *LocalAddress*,*HeartbeatInterval*

Optional parameters (cluster-receiver ChannelType):
ModeName, *TpName*, *DiscInterval*, *ShortRetryCount*, *ShortRetryInterval*, *LongRetryCount*, *LongRetryInterval*, *DataConversion*, *BatchSize*, *PutAuthority*, *SeqNumberWrap*, *MCAUserIdentifier*, *MsgRetryExit*, *MsgRetryUserData*, *MsgRetryCount*, *MsgRetryInterval*, *HeartbeatInterval*, *NonPersistentMsgSpeed*, *BatchInterval*, *ClusterName*, *ClusterNamelist*, *NetworkPriority*, *SSLClientAuth*, *LocalAddress*, *BatchHeartbeat*, *MCAType*

Optional parameters (cluster-sender ChannelType):
ModeName, *TpName*, *MCAName*, *BatchSize*, *DiscInterval*, *ShortRetryCount*, *ShortRetryInterval*, *LongRetryCount*, *LongRetryInterval*, *SeqNumberWrap*, *DataConversion*, *MCAType*, *MCAUserIdentifier*, *UserIdentifier*, *Password*, *HeartbeatInterval*, *NonPersistentMsgSpeed*, *BatchInterval*, *ClusterName*,
| *ClusterNamelist*, *LocalAddress*, *BatchHeartbeat*

Create Channel

The Create Channel (MQCMD_CREATE_CHANNEL) command creates a WebSphere MQ 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.

The channel commands are supported on all platforms.

Required parameters :
ChannelName, *ChannelType*

Required parameters (Not server, receiver, or server-connection channel types):
ConnectionName

Required parameters (Not cluster-receiver, or cluster-sender channel types):
TransportType

Required parameters (Create channel — sender and server channel types only):
XmitQName

Optional parameters (any ChannelType):
ChannelDesc, *SecurityExit*, *SendExit*, *ReceiveExit*, *MaxMsgLength*,
| *SecurityUserData*, *SendUserData*, *ReceiveUserData*, *SSLCipherSpec*,
| *SSLPeerName*, *Replace*

Optional parameters (sender ChannelType):
ModeName, *TpName*, *MCAName*, *BatchSize*, *DiscInterval*, *ShortRetryCount*, *ShortRetryInterval*, *LongRetryCount*, *LongRetryInterval*, *SeqNumberWrap*, *DataConversion*, *MCAType*, *MCAUserIdentifier*, *UserIdentifier*, *Password*,
| *HeartbeatInterval*, *NonPersistentMsgSpeed*, *BatchInterval*, *LocalAddress*,
| *BatchHeartbeat*, *MsgExit*, *MsgUserData*

Optional parameters (server ChannelType):
ModeName, *TpName*, *MCAName*, *BatchSize*, *DiscInterval*, *ShortRetryCount*, *ShortRetryInterval*, *LongRetryCount*, *LongRetryInterval*, *SeqNumberWrap*, *DataConversion*, *MCAType*, *MCAUserIdentifier*, *UserIdentifier*, *Password*,
| *HeartbeatInterval*, *NonPersistentMsgSpeed*, *BatchInterval*, *LocalAddress*,
| *BatchHeartbeat*, *MsgExit*, *MsgUserData*, *ConnectionName*, *SSLClientAuth*

Optional parameters (receiver ChannelType):
BatchSize, *PutAuthority*, *SeqNumberWrap*, *MCAUserIdentifier*, *MsgRetryExit*, *MsgRetryUserData*, *MsgRetryCount*, *MsgRetryInterval*, *HeartbeatInterval*,
| *NonPersistentMsgSpeed*, *SSLClientAuth*, *MsgExit*, *MsgUserData*

Optional parameters (requester ChannelType):
ModeName, *TpName*, *MCAName*, *BatchSize*, *PutAuthority*, *SeqNumberWrap*, *MCAType*, *MCAUserIdentifier*, *UserIdentifier*, *Password*, *MsgRetryExit*, *MsgRetryUserData*, *MsgRetryCount*, *MsgRetryInterval* *HeartbeatInterval*,
| *NonPersistentMsgSpeed*, *SSLClientAuth*, *LocalAddress*, *MsgExit*,
MsgUserData

Optional parameters (server-connection ChannelType):
MCAUserIdentifier, *SSLClientAuth*, *HeartbeatInterval*, *PutAuthority*

Optional parameters (client-connection ChannelType):
ModeName, *TpName* *QMgrName*, *UserIdentifier*, *Password*, *LocalAddress*,*HeartbeatInterval*

Optional parameters (cluster-receiver ChannelType):
ModeName, *TpName*, *DiscInterval*, *ShortRetryCount*, *ShortRetryInterval*, *LongRetryCount*, *LongRetryInterval*, *DataConversion*, *BatchSize*, *PutAuthority*, *SeqNumberWrap*, *MCAUserIdentifier*, *MsgRetryExit*, *MsgRetryUserData*, *MsgRetryCount*, *MsgRetryInterval*, *HeartbeatInterval*, *NonPersistentMsgSpeed*, *BatchInterval*, *ClusterName*, *ClusterNamelist*, *NetworkPriority*, *SSLClientAuth*, *LocalAddress*, *BatchHeartbeat*, *MCAType*, *MsgExit*, *MsgUserData*, *TransportType*

Optional parameters (cluster-sender ChannelType):
ModeName, *TpName*, *MCAName*, *BatchSize*, *DiscInterval*, *ShortRetryCount*, *ShortRetryInterval*, *LongRetryCount*, *LongRetryInterval*, *SeqNumberWrap*, *DataConversion*, *MCAType*, *MCAUserIdentifier*, *UserIdentifier*, *Password*, *HeartbeatInterval*, *NonPersistentMsgSpeed*, *BatchInterval*, *ClusterName*, *ClusterNamelist*, *LocalAddress*, *BatchHeartbeat*, *MsgExit*, *MsgUserData*, *TransportType*

Required parameters

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

Specifies the name of the channel definition to be changed, or created

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

This parameter is required on all types of channel; on a CLUSSDR it can be different from on the other channel types. If your convention for naming channels includes the name of the queue manager, you can make a CLUSSDR definition using the `+QMNAME+` construction, and WebSphere MQ substitutes the correct repository queue manager name in place of `+QMNAME+`. This facility applies to AIX, HP-UX, Linux, OS/400, Solaris, and Windows only. See *WebSphere MQ Queue Manager Clusters* for more details.

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 changed, copied, or created. The value can 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.

MQCHT_CLUSRCVR
Cluster-receiver.

This value is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

MQCHT_CLUSSDR
Cluster-sender.

This value is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

Optional parameters

Replace (MQCFIN)
Replace channel definition (parameter identifier: MQIACF_REPLACE).

The value can be:

MQRP_YES
Replace existing definition.

If *ChannelType* is MQCHT_CLUSSDR, MQRP_YES can be specified only if the channel was created manually.

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 can be:

MQXPT_LU62
LU 6.2.

MQXPT_TCP
TCP.

MQXPT_NETBIOS
NetBIOS.

This value is supported in the following environments: OS/2, Windows.

MQXPT_SPX
SPX.

This value is supported in the following environments: OS/2, Windows.

MQXPT_DECNET
DECnet.

This value is supported in the following environment: Compaq OpenVMS Alpha.

MQXPT_UDP
UDP.

This value is supported in the following environment: AIX.

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 iSeries and UNIX systems, it is of the form

  ```
  libraryname(functionname)
  ```

  Note: On iSeries systems, the following form is also supported for compatibility with older releases:

  ```
  progname libname
  ```

  where *progname* occupies the first 10 characters, and *libname* the second 10 characters (both blank-padded to the right if necessary).
- On Windows, and OS/2 it is of the form

  ```
  dllname(functionname)
  ```

where *dllname* is specified without the suffix ".DLL".

- On Compaq OpenVMS Alpha, it is of the form

```
imagename(functionname)
```

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

MsgExit (MQCFSL)
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 exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

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, Solaris, Windows and Linux.

- 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 (MQCFSL)
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 exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

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, Solaris, Windows and Linux.

- 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 (MQCFSL)
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 exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

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, Solaris, Windows and Linux.

- 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 lower 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, Compaq OpenVMS Alpha, Compaq NonStop Kernel, Linux, HP-UX, OS/2, OS/400, Solaris, and Windows, the maximum message length is 100 MB (104 857 600 bytes).
- On UNIX systems not listed above, 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 (MQCFSL)
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, Solaris, Windows and Linux.

- 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 (MQCFSL)
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, Solaris, Windows and Linux.

- 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 (MQCFSL)
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, Solaris, Windows and Linux.

- 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 Compaq OpenVMS Alpha, OS/400, Compaq NonStop Kernel, UNIX systems, and Windows, this can be set only to blanks. The actual name is taken instead from the CPI-C Communications Side Object or (on Windows) from the CPI-C symbolic destination name properties.

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 Compaq OpenVMS Alpha, OS/400, Compaq NonStop Kernel, UNIX systems, and Windows, this can be set only to blanks. The actual name is taken instead from the CPI-C Communications Side Object or (on Windows) from the CPI-C symbolic destination name properties.

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 on LU. On OS/400, and UNIX systems, specify the name of the CPI-C communications side object. On Windows specify the CPI-C symbolic destination name.
- For MQXPT_TCP you can specify the host name or the network address of the remote machine. On a CLUSRCVR channel, the ConnectionName parameter is optional. On a CLUSRCVR channel, if you leave ConnectionName blank, WebSphere MQ generates a ConnectionName for you, assuming the default port and using the current IP address of the system.
- 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 WebSphere MQ default value (5e86 hex) is assumed.
- For MQXPT_UDP specify either the host name or the network address of the remote machine.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_REQUESTER, MQCHT_CLNTCONN, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

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, MQCHT_REQUESTER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

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 MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

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, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

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, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

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, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

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, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

DataConversion (MQCFIN)
Whether sender should convert application data (parameter identifier: MQIACH_DATA_CONVERSION).

This parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

The value can be:

MQCDC_NO_SENDER_CONVERSION
No conversion by sender.

MQCDC_SENDER_CONVERSION
Conversion by sender.

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, MQCHT_REQUESTER, or MQCHT_CLUSRCVR.

The value can 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, MQCHT_REQUESTER, or MQCHT_CLUSSDR.

This parameter is valid only on AIX, HP-UX, OS/400, Solaris, Windows and Linux.

The value can be:

MQMCAT_PROCESS
Process.

MQMCAT_THREAD
Thread.

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 WebSphere 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 MCA user identifier depends on the environment in which the MCA is running. MQ_MCA_USER_ID_LENGTH gives the maximum length for the environment for which your application is running. MQ_MAX_MCA_USER_ID_LENGTH gives the maximum for all supported environments.

On Windows, you can optionally qualify a user identifier with the domain name in the following format:

```
user@domain
```

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, MQCHT_CLNTCONN, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

- This parameter is supported in the following environments: Compaq OpenVMS Alpha, OS/2, OS/400, Compaq NonStop Kernel, UNIX systems.

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, MQCHT_CLNTCONN, or MQCHT_CLUSSDR.

- This parameter is supported in the following environments: Compaq OpenVMS Alpha, OS/2, OS/400, Compaq NonStop Kernel, UNIX systems.

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/400, Solaris, Windows, and Linux.

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 exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

This parameter is valid only for *ChannelType* values of MQCHT_RECEIVER, MQCHT_REQUESTER, or MQCHT_CLUSRCVR.

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, Solaris, Windows and Linux.

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, MQCHT_REQUESTER, or MQCHT_CLUSRCVR.

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, Solaris, Windows and Linux.

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, MQCHT_REQUESTER, or MQCHT_CLUSRCVR.

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, Solaris, Windows and Linux.

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, MQCHT_REQUESTER, or MQCHT_CLUSRCVR.

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.

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, MQCHT_REQUESTER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR, 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, OS/2, OS/400, Solaris, Windows and Linux.
- 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, Solaris, Windows and Linux.

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, Solaris, Windows and Linux.

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 might be lost if there is a transmission failure.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_RECEIVER, MQCHT_REQUESTER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR. The value can 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, Solaris, Windows and Linux.

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, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

ClusterName (MQCFST)
Cluster name (parameter identifier: MQCA_CLUSTER_NAME).

The name of the cluster to which the channel belongs. *ClusterName* and *ClusterNamelist* should not be specified together.

This parameter applies only to channels with a *ChannelType* of:
MQCHT_CLUSSDR
MQCHT_CLUSRCVR

The maximum length of the string is MQ_CLUSTER_NAME_LENGTH.

This value is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

ClusterNamelist (MQCFST)
Cluster namelist (parameter identifier: MQCA_CLUSTER_NAMELIST).

The name, of the namelist, that specifies a list of clusters to which the channel belongs. *ClusterName* and *ClusterNamelist* should not be specified together.

This parameter applies only to channels with a *ChannelType* of:
MQCHT_CLUSSDR
MQCHT_CLUSRCVR

This value is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

NetworkPriority (MQCFIN)
Network priority (parameter identifier: MQIACH_NETWORK_PRIORITY).

The priority for the network connection. If there are multiple paths available, distributed queuing selects the path with the highest priority.

The value must be in the range 0 (lowest) through 9 (highest).

This parameter applies only to channels with a *ChannelType* of MQCHT_CLUSRCVR

This value is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

`LocalAddress` (MQCFST)
Local communications address for the channel (parameter identifier: MQCACH_LOCAL_ADDRESS).

The maximum length of the string is MQ_LOCAL_ADDRESS_LENGTH.

The value that you specify depends on the transport type *(TransportType)* to be used:

TCP/IP
The value is the optional IP address and optional port or port range to be used for outbound TCP/IP communications. The format for this information is as follows:

```
[ip-addr][(low-port[,high-port])]
```

where `ip-addr` is specified in dotted decimal or alphanumeric form, and `low-port` and `high-port` are port numbers enclosed in parentheses. All are optional.

All Others
The value is ignored; no error is diagnosed.

Use this parameter if you want a channel to use a particular IP address, port, or port range for outbound communications. This is useful when a machine is connected to multiple networks with different IP addresses.

Examples of use

Value	Meaning
9.20.4.98	Channel binds to this address locally
9.20.4.98 (1000)	Channel binds to this address and port 1000 locally
9.20.4.98 (1000,2000)	Channel binds to this address and uses a port in the range 1000 to 2000 locally
(1000)	Channel binds to port 1000 locally
(1000,2000)	Channel binds to a port in the range 1000 to 2000 locally

This parameter is valid for the following channel types:
- MQCHT_SENDER
- MQCHT_SERVER
- MQCHT_REQUESTER
- MQCHT_CLNTCONN
- MQCHT_CLUSRCVR
- MQCHT_CLUSSDR

Note:
- Do not confuse this parameter with *ConnectionName*. The *LocalAddress* parameter specifies the characteristics of the local communications; the *ConnectionName* parameter specifies how to reach a remote queue manager.

BatchHeartbeat (MQCFIN)
The batch heartbeat interval (parameter identifier: MQIACH_BATCH_HB).

Batch heartbeating allows sender-type channels to determine whether the remote channel instance is still active, before going in-doubt. The value can be between 0 and 999999. A value of 0 indicates that batch heartbeating is not to be used. Batch heartbeat is measured in milliseconds.

This parameter is valid only for *ChannelType* values of MQCHT_SENDER, MQCHT_SERVER, MQCHT_CLUSSDR, or MQCHT_CLUSRCVR.

SSLCipherSpec (MQCFST)
CipherSpec (parameter identifier: MQCACH_SSL_CIPHER_SPEC).

The length of the string is MQ_SSL_CIPHER_SPEC_LENGTH.

This parameter is supported only on AIX, HP-UX, Linux, OS/400, Solaris, Windows, and z/OS.

This parameter is valid only for channels with a transport type (TRPTYPE) of TCP. If the TRPTYPE is not TCP, the data is ignored and no error message is issued.

The SSLCIPH values must specify the same CipherSpec on both ends of the channel.

Specify the name of the CipherSpec that you are using. Alternatively, on OS/400, and z/OS, you can specify the two-digit hexadecimal code.

The following table shows the CipherSpecs that can be used with WebSphere MQ SSL.

Table 2. CipherSpecs that can be used with WebSphere MQ SSL support

CipherSpec name	Hash algorithm	Encryption algorithm	Encryption bits
NULL_MD5[1]	MD5	None	0
NULL_SHA[1]	SHA	None	0
RC4_MD5_EXPORT[1]	MD5	RC4	40
RC4_MD5_US[2]	MD5	RC4	128
RC4_SHA_US[2]	SHA	RC4	128
RC2_MD5_EXPORT[1]	MD5	RC2	40
DES_SHA_EXPORT[1]	SHA	DES	56
RC4_56_SHA_EXPORT1024[3,4,5]	SHA	RC4	56
DES_SHA_EXPORT1024[3,4,5,6]	SHA	DES	56
TRIPLE_DES_SHA_US[4]	SHA	3DES	168
TLS_RSA_WITH_AES_128_CBC_SHA[7]	SHA	AES	128
TLS_RSA_WITH_AES_256_CBC_SHA[7]	SHA	AES	256
AES_SHA_US[8]	SHA	AES	128

Table 2. CipherSpecs that can be used with WebSphere MQ SSL support (continued)

CipherSpec name	Hash algorithm	Encryption algorithm	Encryption bits
Notes: 1. On OS/400, available when either AC2 or AC3 are installed 2. On OS/400, available only when AC3 is installed 3. Not available for z/OS 4. Not available for OS/400 5. Specifies a 1024–bit handshake key size 6. Not available for Windows 7. Available for AIX platforms only 8. Available for OS/400, AC3 only			

If the SSLCIPH parameter is blank, no attempt is made to use SSL on the channel.

SSLPeerName (MQCFST)
Peer name (parameter identifier: MQCACH_SSL_PEER_NAME).

The length of the string is MQ_SSL_PEER_NAME_LENGTH.

Specifies the filter to use to compare with the Distinguished Name of the certificate from the peer queue manager or client at the other end of the channel. (A Distinguished Name is the identifier of the SSL certificate.) If the Distinguished Name in the certificate received from the peer does not match the SSLPEER filter, the channel does not start.

This parameter is supported only on AIX, HP-UX, Linux, OS/400, Solaris, Windows, and z/OS.

This parameter is optional; if it is not specified, the Distinguished Name of the peer is not checked at channel start up. (The Distinguished Name from the certificate is still written into the SSLPEER definition held in memory, and passed to the security exit). If SSLCIPH is blank, the data is ignored and no error message is issued.

This parameter is valid for all channel types.

The SSLPEER value is specified in the standard form used to specify a Distinguished Name. For example: `SSLPEER('CN="xxx yyy zzz",O=xxx,C=xxx')`

You can use a semi-colon as a separator instead of a comma.

The possible attribute types supported are:

CN	common name
T	title
OU	organizational unit name
O	organization name
L	locality name
ST, SP or S	state or province name
C	country

WebSphere MQ only accepts upper case letters for the attribute types.

If any of the unsupported attribute types are specified in the SSLPEER string, an error is output either when the attribute is defined or at run time (depending on which platform you are running on), and the string is deemed not to have matched the flowed certificate's Distinguished Name.

If the flowed certificate's Distinguished Name contains multiple OU (organizational unit) attributes, and SSLPEER specifies these attributes to be compared, they must match in the order that they are found in the certificate's Distinguished Name, and must start with the first OU, or an asterisk. For example, if the flowed certificate's Distinguished Name contains the OUs `OU=One,OU=Two,OU=Three`, you can specify the following SSLPEER values:

```
('OU=One,OU=Two')
```

```
('OU=*,OU=Two,OU=Three')
```

```
('OU=*,OU=Two')
```

but not the following SSLPEER values:

```
('OU=Two,OU=Three')
```

```
('OU=One,OU=Three')
```

```
('OU=Two')
```

Any or all of the attribute values can be generic, either an asterisk (*) on its own, or a stem with initiating or trailing asterisks. This allows the SSLPEER to match any Distinguished Name value, or any value starting with the stem for that attribute.

If an asterisk is specified at the beginning or end of any attribute value in the Distinguished Name on the certificate, you can specify * to check for an exact match in SSLPEER. For example, if you have an attribute of CN=Test* in the Distinguished Name of the certificate, you can use the following command:

```
SSLPEER('CN=Test\*')
```

SSLClientAuth (MQCFIN)
Client authentication (parameter identifier: MQIACH_SSL_CLIENT_AUTH).

The value can be:

MQSCA_REQUIRED
Client authentication required

MQSCA_OPTIONAL
Client authentication optional.

Defines whether WebSphere MQ requires a certificate from the SSL client.

The initiating end of the channel acts as the SSL client, so this applies to the end of the channel that receives the initiation flow, which acts as the SSL server.

This parameter is supported only on AIX, HP-UX, Linux, OS/400, Solaris, Windows, and z/OS.

The parameter is used only for channels with SSLCIPH specified. If SSLCIPH is blank, the data is ignored and no error message is issued.

Error codes

This command might return the following in the response format header, in addition to the values shown on page18.

Reason (MQLONG)
The value can be:

MQRCCF_ATTR_VALUE_ERROR
Attribute value not valid.

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_CLUSTER_NAME_CONFLICT
Cluster name conflict.

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_NET_PRIORITY_ERROR
Network priority value error.

MQRCCF_NET_PRIORITY_WRONG_TYPE
Network priority attribute not allowed for this channel type.

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_SSL_CIPHER_SPEC_ERROR
SSL CipherSpec not valid.

MQRCCF_SSL_CLIENT_AUTH_ERROR
SSL client authentication not valid.

MQRCCF_SSL_PEER_NAME_ERROR
SSL peer name not valid.

MQRCCF_STRUCTURE_TYPE_ERROR
Structure type not valid.

MQRCCF_WRONG_CHANNEL_TYPE
Parameter not allowed for this channel type.

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, Copy, and Create Namelist

The Change Namelist (MQCMD_CHANGE_NAMELIST) command changes the specified attributes of an existing MQSeries namelist definition. For any optional parameters that are omitted, the value does not change.

The Copy Namelist (MQCMD_COPY_NAMELIST) command creates a new MQSeries namelist definition, using, for attributes not specified in the command, the attribute values of an existing namelist definition.

The Create Namelist (MQCMD_CREATE_NAMELIST) command creates a new MQSeries namelist definition. Any attributes that are not defined explicitly are set to the default values on the destination queue manager.

Required parameter (Change and Create Namelist):
NamelistName

Required parameters (Copy Namelist):
FromNamelistName, *ToNamelistName*

Optional parameters:
Replace, *NamelistDesc*, *Names*

Required parameter (Change and Create Namelist)

NamelistName (MQCFST)
The name of the namelist definition to be changed (parameter identifier: MQCA_NAMELIST_NAME).

The maximum length of the string is MQ_NAMELIST_NAME_LENGTH.

Required parameters (Copy Namelist)

FromNamelistName (MQCFST)
The name of the namelist definition to be copied from (parameter identifier: MQCACF_FROM_NAMELIST_NAME).

This specifies the name of the existing namelist definition that contains values for the attributes not specified in this command.

The maximum length of the string is MQ_NAMELIST_NAME_LENGTH.

ToNamelistName (MQCFST)
To namelist name (parameter identifier: MQCACF_TO_NAMELIST_NAME).

This specifies the name of the new namelist definition. If a namelist definition with this name already exists, *Replace* must be specified as MQRP_YES.

The maximum length of the string is MQ_NAMELIST_NAME_LENGTH.

Optional parameters

Replace (MQCFIN)
Replace attributes (parameter identifier: MQIACF_REPLACE).

If a namelist definition with the same name as *ToNamelistName* already exists, this specifies whether it is to be replaced. The value can be:

MQRP_YES
Replace existing definition.

MQRP_NO
Do not replace existing definition.

NamelistDesc (MQCFST)
Description of namelist definition (parameter identifier: MQCA_NAMELIST_DESC).

This is a plain-text comment that provides descriptive information about the namelist definition. It should contain only displayable characters.

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 might be translated incorrectly.

The maximum length of the string is MQ_NAMELIST_DESC_LENGTH.

Names (MQCFSL)
The names to be placed in the namelist (parameter identifier: MQCA_NAMES).

The number of names in the list is given by the *Count* field in the MQCFSL structure. The length of each name is given by the *StringLength* field in that structure. The maximum length of a name is MQ_OBJECT_NAME_LENGTH.

Error codes

| This command might return the following in the response format header, in
| addition to the values shown on page 18.

Reason (MQLONG)
The value can 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 not valid.

MQRCCF_CFSL_COUNT_ERROR
Name count not valid.

MQRCCF_CFSL_STRING_LENGTH_ERROR
String length value not valid.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length 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, Copy, and Create Process

The Change Process (MQCMD_CHANGE_PROCESS) command changes the specified attributes of an existing WebSphere MQ process definition. For any optional parameters that are omitted, the value does not change.

The Copy Process (MQCMD_COPY_PROCESS) command creates a new WebSphere MQ process definition, using, for attributes not specified in the command, the attribute values of an existing process definition.

The Create Process (MQCMD_CREATE_PROCESS) command creates a new WebSphere MQ process definition. Any attributes that are not defined explicitly are set to the default values on the destination queue manager.

Required parameter (Change and Create Process):
ProcessName

Required parameters (Copy Process):
FromProcessName, *ToProcessName*

Optional parameters:
Replace, *ProcessDesc*, *ApplType*, *ApplId*, *EnvData* *UserData*

Required parameters (Change and Create Process)

ProcessName (MQCFST)
The name of the process definition to be changed or created (parameter identifier: MQCA_PROCESS_NAME).

The maximum length of the string is MQ_PROCESS_NAME_LENGTH.

Required parameters (Copy Process)

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 can 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 must 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 might 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 or Windows 95, Windows 98 application.

MQAT_DOS
DOS client application.

MQAT_WINDOWS
Windows client application.

MQAT_UNIX
UNIX application.

MQAT_AIX
AIX application (same value as MQAT_UNIX).

MQAT_CICS
CICS® transaction.

MQAT_VMS
Compaq OpenVMS Alpha application.

MQAT_NSK
Compaq NonStop Kernel 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 Compaq OpenVMS Alpha:

  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 Compaq NonStop Kernel:

  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:

  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

| This command might return the following in the response format header, in
| addition to the values shown on page 18.

Reason (MQLONG)
The value can 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, Copy, and Create Queue

The Change Queue (MQCMD_CHANGE_Q) command changes the specified attributes of an existing WebSphere MQ queue. For any optional parameters that are omitted, the value does not change.

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.

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 (Change and Create Queue):
QName, *QType*

Required parameters (Copy Queue):
FromQName, *ToQName*, *QType*

Optional parameters (any QType):
Replace, *QDesc*, *InhibitPut*, *DefPriority*, *DefPersistence*

Optional parameters (alias QType):
Force, *InhibitGet*, *BaseQName*, *Scope*, *ClusterName*, *ClusterNamelist*, *DefBind*

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*, *ClusterName*, *ClusterNamelist*, *DefBind*

Optional parameters (remote QType):
Force, *RemoteQName*, *RemoteQMgrName*, *XmitQName*, *Scope*, *ClusterName*, *ClusterNamelist*, *DefBind*

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 (Change and Create Queue)

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.

Required parameters (Copy Queue)

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.

Required parameters (all commands)

QType (MQCFIN)
Queue type (parameter identifier: MQIA_Q_TYPE).

The value specified must match the type of the queue being changed.

The value can be:

MQQT_ALIAS
Alias queue definition.

MQQT_LOCAL
Local queue.

MQQT_REMOTE
Local definition of a remote queue.

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 can be:

MQRP_YES
Replace existing definition.

MQRP_NO
Do not replace existing definition.

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 can 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 can 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 can 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 can 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 WebSphere MQ process that identifies the application to be started when a trigger event occurs.

- On AIX, HP-UX, OS/2, OS/400, Solaris, Windows and Linux, if the queue is a transmission queue the process name can be left as all blanks.
- 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 greater than or equal to 0, and less than or equal to:

- 999 999 999 if the queue is on AIX, HP-UX, OS/400, Solaris, Linux, or Windows or
- 640 000 if the queue is on any other Websphere MQ platform.

MaxMsgLength (MQCFIN)
Maximum message length (parameter identifier: MQIA_MAX_MSG_LENGTH).

The maximum length for messages on the queue. Because applications might use the value of this attribute to determine the size of buffer they need to retrieve messages from the queue, change this value only if it is known that this will not cause an application to operate incorrectly.

Do not 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, Compaq OpenVMS Alpha, Compaq NonStop Kernel, HP-UX, OS/2, OS/400, Solaris, Linux, and Windows, the maximum message length is 100 MB (104 857 600 bytes).
- On UNIX systems not listed above, the maximum message length is 4 MB (4 194 304 bytes).

BackoutThreshold (MQCFIN)
Backout threshold (parameter identifier: MQIA_BACKOUT_THRESHOLD).

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 the queue can be shared (parameter identifier: MQIA_SHAREABILITY).

Specifies whether multiple instances of applications can open this queue for input.

The value can 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 can 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 is saved (hardened) across restarts of the message queue manager.

Note: WebSphere MQ for iSeries always hardens the count, regardless of the setting of this attribute.

The value can be:

MQQA_BACKOUT_HARDENED
Backout count remembered.

MQQA_BACKOUT_NOT_HARDENED
Backout count might not be remembered.

MsgDeliverySequence (MQCFIN)
Whether priority is relevant (parameter identifier: MQIA_MSG_DELIVERY_SEQUENCE).

The value can 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 might 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 can 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 is not normally 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, Solaris, Windows and Linux.

The value can 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 can 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.

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 can be:

MQTC_OFF
Trigger messages not required.

MQTC_ON
Trigger messages required.

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.

The value can 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 is supported (0 through 9).

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.

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.

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).

The value can 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).

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 can be:

MQSCO_Q_MGR
Queue-manager scope.

MQSCO_CELL
Cell scope.

This value is not supported on OS/400.

QDepthHighLimit (MQCFIN)
High limit for queue depth (parameter identifier: MQIA_Q_DEPTH_HIGH_LIMIT).

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).

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.

| Specify the value as a percentage of the maximum queue depth (*MaxQDepth*
| attribute), in the range 0 through 100.

QDepthMaxEvent (MQCFIN)
Controls whether Queue Full events are generated (parameter identifier: MQIA_Q_DEPTH_MAX_EVENT).

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, “Definitions of the Programmable Command Formats”, on page 17.

The value can 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).

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, "Definitions of the Programmable Command Formats", on page 17.

The value can 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).

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, "Definitions of the Programmable Command Formats", on page 17.

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

QServiceInterval (MQCFIN)
Target for queue service interval (parameter identifier: MQIA_Q_SERVICE_INTERVAL).

The service interval used for comparison to generate Queue Service Interval High and Queue Service Interval OK events. See the *QServiceIntervalEvent* parameter.

| Specify a value in the range 0 through 999 999 999 milliseconds.

QServiceIntervalEvent (MQCFIN)
Controls whether Service Interval High or Service Interval OK events are generated (parameter identifier: MQIA_Q_SERVICE_INTERVAL_EVENT).

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, "Definitions of the Programmable Command Formats", on page 17.

The value can 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**.

ClusterName (MQCFST)
Cluster name (parameter identifier: MQCA_CLUSTER_NAME).

The name of the cluster to which the queue belongs.

Changes to this parameter do not affect instances of the queue that are open.

The maximum length of the string is MQ_CLUSTER_NAME_LENGTH.

ClusterName and *ClusterNamelist* should not be specified together.

This parameter is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

ClusterNamelist (MQCFST)
Cluster namelist (parameter identifier: MQCA_CLUSTER_NAMELIST).

The name of the namelist, that specifies a list of clusters to which the queue belongs.

Changes to this parameter do not affect instances of the queue that are open.

ClusterName and *ClusterNamelist* should not be specified together.

This parameter is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

DefBind (MQCFIN)
Bind definition (parameter identifier: MQIA_DEF_BIND).

The parameter specifies the binding to be used when MQOO_BIND_AS_Q_DEF is specified on the MQOPEN call. The value can be:

MQBND_BIND_ON_OPEN
The binding is fixed by the MQOPEN call.

MQBND_BIND_NOT_FIXED
The binding is not fixed.

Changes to this parameter do not affect instances of the queue that are open.

This parameter is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

Error codes

| This command might return the following in the response format header, in
| addition to the values shown on page 18.

Reason (MQLONG)
The value can 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_CLUSTER_NAME_CONFLICT
Cluster name conflict.

MQRCCF_CLUSTER_Q_USAGE_ERROR
Cluster usage conflict.

MQRCCF_DYNAMIC_Q_SCOPE_ERROR
Dynamic queue scope error.

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_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.

This PCF is supported on all platforms.

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* *ClusterWorkloadExit*, *ClusterWorkloadData*, *ClusterWorkloadLength*,
| *RepositoryName*, *RepositoryNamelist*, *CodedCharSetId*, *ConfigurationEvent*,
| *SSLKeyRepository*, *SSLNamelist*, *SSLCryptoHardware*

Optional parameters

Force (MQCFIN)
Force changes (parameter identifier: MQIACF_FORCE).

Specifies whether the command will 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 will 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 generated only 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.

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.

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).

The value can 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).

The value can 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).

The value can 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).

The value can 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).

The value can 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).

The value can 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 must 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 might not work correctly.

The lower limit for this parameter is 32 KB (32 768 bytes). The upper limit depends on the environment:

- On AIX, Compaq OpenVMS Alpha, Compaq NonStop Kernel, HP-UX, OS/2, OS/400, Solaris, Linux, and Windows, the maximum message length is 100 MB (104 857 600 bytes).
- On UNIX systems not listed above, 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).

Auto-definition for cluster-sender channels is always enabled.

This parameter is supported in the following environments: AIX, HP-UX, OS/2, OS/400, Solaris, Windows and Linux.

The value can 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), when a receiver, server-connection, or cluster-sender channel is auto-defined.

This parameter is supported in the following environments: AIX, HP-UX, OS/2, OS/400, Solaris, Windows and Linux.

The value can 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).

This exit is invoked when an inbound request for an undefined channel is received, if:

1. The channel is a cluster-sender, or
2. Channel auto-definition is enabled (see *ChannelAutoDef*).

This exit is also invoked when a cluster-receiver channel is started.

The format of the name is the same as for the *SecurityExit* parameter described in "Change, Copy and Create Channel" on page 23.

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

This parameter is supported in the following environments: AIX, HP-UX, OS/2, OS/400, Solaris, Windows and Linux.

ClusterWorkLoadExit (MQCFST)
Cluster workload exit name (parameter identifier: MQCA_CLUSTER_WORKLOAD_EXIT).

If a nonblank name is defined this exit is invoked when a message is put to a cluster queue.

The format of the name is the same as for the *SecurityExit* parameter described in "Change, Copy and Create Channel" on page 23.

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

This parameter is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

ClusterWorkLoadData (MQCFST)
Cluster workload exit data (parameter identifier: MQCA_CLUSTER_WORKLOAD_DATA).

This is passed to the cluster workload exit when it is called.

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

This parameter is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

ClusterWorkLoadLength (MQCFIN)
Cluster workload length (parameter identifier: MQIA_CLUSTER_WORKLOAD_LENGTH).

The maximum length of the message passed to the cluster workload exit.

The value of this attribute must be in the range 0 through 999 999 999.

This parameter is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

RepositoryName (MQCFST)
Cluster name (parameter identifier: MQCA_REPOSITORY_NAME).

The name of a cluster for which this queue manager provides a repository manager service.

The maximum length of the string is MQ_OBJECT_NAME_LENGTH.

RepositoryName and *RepositoryNamelist* must not be specified together.

This parameter is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

RepositoryNamelist (MQCFST)
Repository namelist (parameter identifier: MQCA_REPOSITORY_NAMELIST).

The name, of a namelist of clusters, for which this queue manager provides a repository manager service.

This queue manager does not have a full repository, but can be a client of other repository services that are defined in the cluster, if

- Both *RepositoryName* and *RepositoryNamelist* are blank, or

- `RepositoryName` is blank and the namelist specified by `RepositoryNamelist` is empty.

`RepositoryName` and `RepositoryNamelist` must not be specified together.

This parameter is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

`CodedCharSetId` (MQCFIN)
Queue manager coded character set identifier (parameter identifier: MQIA_CODED_CHAR_SET_ID).

The coded character set identifier (CCSID) for the queue manager. The CCSID is the identifier used with all character string fields defined by the application programming interface (API). It does not apply to application data carried in the text of a message unless the CCSID in the message descriptor, when the message is put with an MQPUT or MQPUT1, is set to the value MQCCSI_Q_MGR.

Specify a value in the range 1 through 65 535.

The CCSID must specify a value that is defined for use on the platform and use an appropriate character set. The character set must be:
- EBCDIC on OS/400
- ASCII or ASCII-related on other platforms

Stop and restart the queue manager after execution of this command so that all processes reflect the changed CCSID of the queue manager.

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

`SSLKeyRepository` (MQCFST)
The SSL key repository (parameter identifier: MQCA_SSL_KEY_REPOSITORY).

The length of the string is MQ_SSL_KEY_REPOSITORY_LENGTH.

Indicates the name of the Secure Sockets Layer key repository.

This parameter is supported only on AIX, HP-UX, Linux, OS/400, Solaris, Windows, and z/OS.

The format of the name depends on the environment:
- On z/OS, it is the name of a key ring.
- On OS/400, it is of the form *pathname/keyfile*, where *keyfile* is specified without the suffix (.kdb), and identifies a GSKit key database file. The default value is `/QIBM/UserData/ICSS/Cert/Server/Default.`
- On UNIX platforms, it is of the form *pathname/keyfile*, where *keyfile* is specified without the suffix (.kdb), and identifies a GSKit key database file. The default value is `/var/mqm/qmgrs/QMGR/ssl/key`, where `QMGR` is replaced by the queue manager name.
- On Windows, the key database is held in a Microsoft Certificate Store file, which has a filename of the form `xxx.sto`, where `xxx` is your chosen name. The SSLKEYR attribute is the path to this file along with the filename stem, (that is, all characters in the filename up to but not including the .sto file extension). WebSphere MQ automatically appends the .sto suffix.

On OS/400, Windows, and UNIX systems, the syntax of this parameter is validated to ensure that it contains a valid, absolute, directory path.

If SSLKEYR is blank, or is set to a value that does not correspond to a key ring or key database file, channels using SSL fail to start.

Changes to SSLKEYR become effective:

- On OS/400, Windows, and UNIX platforms, when a new channel process is started.
- For channels that run as threads of the channel initiator on OS/400, Windows, and UNIX platforms, when the channel initiator is restarted.
- For channels that run as threads of the listener on OS/400, Windows, and UNIX platforms, when the listener is restarted.
- On z/OS, when the channel initiator is restarted.

SSLCRLNamelist (MQCFST)
The SSL namelist (parameter identifier: MQCA_SSL_CRL_NAMELIST).

The length of the string is MQ_NAMELIST_NAME_LENGTH.

Indicates the name of a namelist of authentication information objects to be used for CRL checking by the queue manager.

This parameter is supported only on AIX, HP-UX, Linux, OS/400, Solaris, Windows, and z/OS.

If SSLCRLNamelist is blank, CRL checking is not invoked.

Changes to SSLCRLNamelist, or to the names in a previously specified namelist, or to previously referenced authentication information objects become effective:

- On OS/400, Windows, and UNIX platforms, when a new channel process is started.
- For channels that run as threads of the channel initiator on OS/400, Windows, and UNIX platforms, when the channel initiator is restarted.
- For channels that run as threads of the listener on OS/400, Windows, and UNIX platforms, when the listener is restarted.
- On z/OS, when the channel initiator is restarted.

SSLCryptoHardware (MQCFST)
The SSL cryptographic hardware (parameter identifier: MQCA_SSL_CRYPTO_HARDWARE).

The length of the string is MQ_SSL_CRYPTO_HARDWARE_LENGTH.

Sets the name of the parameter string required to configure the cryptographic hardware present on the system.

This parameter is supported on AIX, HP-UX, Solaris, and Linux only.

The string can have one of the following values:

- GSK_ACCELERATOR_RAINBOW_CS_OFF
- GSK_ACCELERATOR_RAINBOW_CS_ON
- GSK_ACCELERATOR_NCIPHER_NF_OFF
- GSK_ACCELERATOR_NCIPHER_NF_ON
- GSK_PKCS11=*<the PKCS #11 driver path and filename>;<the PKCS #11 token label>;<the PKCS #11 token password>;*

The strings containing RAINBOW enable or disable the Rainbow cryptographic hardware. If the Rainbow cryptographic hardware is present, it is enabled by default.

The strings containing NCIPHER enable or disable the nCipher cryptographic hardware. If the nCipher cryptographic hardware is present, it is enabled by default.

To use cryptographic hardware which is accessed using the PKCS #11 interface, you must specify the string containing PKCS11. The PKCS #11 driver path is an absolute path to the shared library providing support for the PKCS #11 card. The PKCS #11 driver filename is the name of the shared library. An example of the value required for the PKCS #11 driver path and filename is `/usr/lib/pkcs11/PKCS11_API.so`

The maximum length of the string is 256 characters. The default value is blank.

If you specify a string that does not begin with one of the cryptographic strings listed above, you get an error. If you specify the GSK_PKCS11 string, the syntax of the other parameters is also checked.

When the SSLCRYP value is changed, the cryptographic hardware parameters specified become the ones used for new SSL connection environments. The new information becomes effective:

- When a new channel process is started.
- For channels that run as threads of the channel initiator, when the channel initiator is restarted.
- For channels that run as threads of the listener, when the listener is restarted.

Error codes

This command might return the following in the response format header, in addition to the values shown on page 18.

Reason (MQLONG)
The value can 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_SYNTAX_ERROR**
| Syntax error found in parameter.

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_PATH_NOT_VALID**
| Path not valid.

| **MQRCCF_PWD_LENGTH_ERROR**
| Password length error.

MQRCCF_Q_MGR_CCSID_ERROR
Coded character set value not valid.

MQRCCF_REPOS_NAME_CONFLICT
Repository names 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 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

This command might return the following in the response format header, in addition to the values shown on page 18.

Reason (MQLONG)
The value can 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 occurs only 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.

Delete Authentication Information Object

Note: This command is supported only on the WebSphere MQ Version 5.3 platforms: AIX, HP-UX, Linux, z/OS, Solaris, Windows, and iSeries.

The Delete authentication information(MQCMD_DELETE_AUTH_INFO) command deletes the specified AuthInfo object.

Required parameters :
AuthInfoName

Required parameters

AuthInfoName (MQCFST)
authentication information object name (parameter identifier: MQCA_AUTH_INFO_NAME).

The maximum length of the string is MQ_AUTH_INFO_NAME_LENGTH.

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 can 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.

Error codes

This command might return the following in the response format header, in addition to the values shown on page 18.

Reason (MQLONG)
The value can 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 Namelist

The Delete Namelist (MQCMD_DELETE_NAMELIST) command deletes an existing WebSphere MQ namelist definition.

This PCF is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

Required parameters:
NamelistName

Optional parameters:
None

Required parameters

NamelistName (MQCFST)
Namelist name (parameter identifier: MQCA_NAMELIST_NAME).

This is the name of the namelist definition to be deleted. The maximum length of the string is MQ_NAMELIST_NAME_LENGTH.

Error codes

This command might return the following in the response format header, in addition to the values shown on page 18.

Reason (MQLONG)
The value can 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 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 Process

The Delete Process (MQCMD_DELETE_PROCESS) command deletes an existing WebSphere MQ process definition.

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

| This command might return the following in the response format header, in
| addition to the values shown on page 18.

Reason (MQLONG)
The value can 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 WebSphere MQ 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 can be:

MQQT_ALIAS
Alias queue definition.

MQQT_LOCAL
Local queue.

MQQT_REMOTE
Local definition of a remote queue.

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 can be:

MQPO_YES
Purge the queue.

MQPO_NO
Do not purge the queue.

Error codes

This command might return the following in the response format header, in addition to the values shown on page 18.

Reason (MQLONG)
The value can 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 WebSphere MQ 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 WebSphere MQ 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 recognized at the receiving queue manager.

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
WebSphere MQ 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

| This command might return the following in the response format header, in
| addition to the values shown on page 18.

Reason (MQLONG)
The value can 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 might be issued, depending upon the command contained in the Escape request.

Escape (Response)

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.

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
WebSphere MQ command.

EscapeText (MQCFST)
Escape text (parameter identifier: MQCACF_ESCAPE_TEXT).

A string holding the response to the original command.

Inquire Authentication Information Object

Note: This command is supported only on the WebSphere MQ Version 5.3 platforms: AIX, HP-UX, Linux, z/OS, Solaris, Windows, and iSeries.

The Inquire authentication information object (MQCMD_INQUIRE_AUTH_INFO) command inquires about the attributes of authentication information objects.

Required parameters :
AuthInfoName

Optional parameters:
AuthInfoAttrs

Required parameters

AuthInfoName (MQCFST)
Authentication information object name (parameter identifier: MQCA_AUTH_INFO_NAME).

The maximum length of the string is MQ_AUTH_INFO_NAME_LENGTH.

Optional parameters

AuthInfoAttrs (MQCFIL)
: Authentication information object attributes (parameter identifier: MQIACF_AUTH_INFO_ATTRS).

  The attribute list can specify the following on its own (this is the default value if the parameter is not specified) :

  MQIACF_ALL
  : All attributes.

  or a combination of the following :

  MQCA_AUTH_INFO_NAME
  : Name of the authentication information object.

  MQIA_AUTH_INFO_TYPE
  : Type of authentication information object.

  MQCA_AUTH_INFO_CONN_NAME
  : Connection name of the authentication information object.

  MQCA_LDAP_USER_NAME
  : LDAP user name in the authentication information object.

  MQCA_LDAP_PASSWORD
  : LDAP password in the authentication information object.

  MQCA_AUTH_INFO_DESC
  : Description of the authentication information object.

Inquire Authentication Information Object (Response)

Note: This command is supported only on the WebSphere MQ Version 5.3 platforms: AIX, HP-UX, Linux, z/OS, Solaris, Windows, and iSeries.

The response of the Inquire authentication information (MQCMD_INQUIRE_AUTH_INFO) command consists of the response header followed by the *AuthInfoName* structure and the requested combination of attribute parameter structures (where applicable).

Always returned:
: *AuthInfoName*

Returned if requested:
: *AuthInfoType*, *AlterationDate*, *LDAPUserName*, *LDAPPassword*, *AuthInfoDesc*, *AlterationTime*, *AuthInfoConnName*

Response data

AuthInfoName (MQCFST)
: authentication information object name (parameter identifier: MQCA_AUTH_INFO_NAME).

  The maximum length of the string is MQ_AUTH_INFO_NAME_LENGTH.

AuthInfoType (MQCFIN)
: The type of authentication information object (parameter identifier: MQIA_AUTH_INFO_TYPE).

  The value can be:

MQAIT_CRL_LDAP
This defines this authentication information object as specifying Certificate Revocation Lists that are held on the LDAP. See the *WebSphere MQ Security* book for more information.

AlterationDate (MQCFST)
Alteration date of the authentication information object (parameter identifier: MQCA_ALTERATION_DATE).

LDAPUserName (MQCFST)
The LDAP user name (parameter identifier: MQCA_LDAP_USER_NAME).

The maximum length is MQ_DISTINGUISHED_NAME_LENGTH.

The Distinguished Name of the user who is binding to the directory.

You cannot use asterisks in the user name.

LDAPPassword (MQCFST)
The LDAP password (parameter identifier: MQCA_LDAP_PASSWORD).

The maximum length is MQ_LDAP_PASSWORD_LENGTH.

AuthInfoDesc (MQCFST)
The description of the authentication information object (parameter identifier: MQCA_AUTH_INFO_DESC).

The maximum length is MQ_AUTH_INFO_DESC_LENGTH.

AlterationTime (MQCFST)
Alteration time of the authentication information object (parameter identifier: MQCA_ALTERATION_TIME).

AuthInfoConnName (MQCFST)
The connection name of the authentication information object (parameter identifier: MQCA_AUTH_INFO_CONN_NAME).

The maximum length of the string is MQ_AUTH_INFO_CONN_NAME_LENGTH.

Inquire Authentication Information Object Names

Note: This command is supported only on the WebSphere MQ Version 5.3 platforms: AIX, HP-UX, Linux, z/OS, Solaris, Windows, and iSeries.

The Inquire authentication information names (MQCMD_INQUIRE_AUTH_INFO_NAMES) command asks for a list of authentication information names that match the generic authentication information name specified.

Required parameters:
AuthInfoName

Optional parameters:
None

Required parameters

AuthInfoName (MQCFST)
Authentication information object name (parameter identifier: MQCA_AUTH_INFO_NAME).

The maximum length of the string is MQ_AUTH_INFO_NAME_LENGTH.

Error codes

This command might return the following in the response format header, in addition to the values shown on page18.

Reason (MQLONG)
The value can 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 Authentication Information Object Names (Response)

Note: This command is supported only on the WebSphere MQ Version 5.3 platforms: AIX, HP-UX, Linux, z/OS, Solaris, Windows, and iSeries.

The response to the inquire authentication information names (MQCMD_INQUIRE_AUTH_INFO_NAMES) command consists of the response header followed by a single parameter structure giving zero or more names that match the specified authentication information name.

Always returned:
AuthInfoNames

Returned if requested:
None

Response data

AuthInfoNames (MQCFSL)
authentication information object names (parameter identifier: MQCACF_AUTH_INFO_NAMES).

Inquire Channel

The Inquire Channel (MQCMD_INQUIRE_CHANNEL) command inquires about the attributes of WebSphere MQ 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 might 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 can 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.

MQCHT_CLUSRCVR
Cluster-receiver.

This value is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

MQCHT_CLUSSDR
Cluster-sender.

This value is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

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 can 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:

MQIACH_CHANNEL_TYPE
: Channel type.

MQIACH_XMIT_PROTOCOL_TYPE
: Transport (transmission protocol) type.

MQCACH_CHANNEL_NAME
: Channel name.

MQCACH_DESC
: Description.

MQCACH_SEC_EXIT_NAME
: Security exit name.

| **MQCACH_SSL_CIPHER_SPEC**
| : SSL cipher spec.

| **MQCACH_SSL_PEER_NAME**
| : SSL peer 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.

| The following are supported on AIX, HP-UX, OS/2, OS/400, Solaris, Linux,
| Windows:

| **MQCA_ALTERATION_DATE**
| Date on which the definition was last altered.

| **MQCA_ALTERATION_TIME**
| Time at which the definition was last altered.

Relevant for sender or server channel types:

MQCACH_XMIT_Q_NAME
Transmission queue name.

| **MQCACH_LOCAL_ADDRESS**
| Local communications address for the channel.

MQCACH_MCA_NAME
Message channel agent name.

MQCACH_MODE_NAME
Mode name.

MQCACH_TP_NAME
Transaction program name.

| **MQIACH_BATCH_HB**
| The value to use for the batch heartbeating.

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.

| **MQCACH_LOCAL_ADDRESS**
| Local communications address for the channel.

The following is supported on Compaq OpenVMS Alpha, OS/2, OS/400, Compaq NonStop Kernel, UNIX systems, and Windows:

MQCACH_CONNECTION_NAME
Connection name.

The following are supported on Compaq OpenVMS Alpha,OS/2, OS/400, Compaq NonStop Kernel, UNIX systems, Windows:

MQCACH_USER_ID
User identifier.

MQCACH_PASSWORD
Password.

The following are supported on AIX, HP-UX, OS/2, OS/400, Solaris, Linux, Windows:

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, Solaris, Linux, Windows:

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_SSL_CLIENT_AUTH**
| SSL client authentication.

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.

| **MQCACH_LOCAL_ADDRESS**
| Local communications address for the channel.

The following is supported on Compaq OpenVMS Alpha, OS/2, OS/400, Compaq NonStop Kernel, UNIX systems, Windows:

MQCACH_CONNECTION_NAME
Connection name.

The following are supported on Compaq OpenVMS Alpha, Compaq NonStop Kernel, OS/2, UNIX systems, Windows:

MQCACH_USER_ID
User identifier.

MQCACH_PASSWORD
Password.

The following is supported on AIX, HP-UX, OS/2, OS/400, Solaris, Linux, Windows:

MQIACH_HB_INTERVAL
Heartbeat interval (seconds).

The following is supported on AIX, HP-UX, OS/2, OS/400, Solaris, Linux, Windows:

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.

| **MQCACH_SSL_CLIENT_AUTH**
| SSL client authentication.

The following is supported on AIX, HP-UX, OS/2, OS/400, Solaris, Linux, Windows:

MQIACH_HB_INTERVAL
Heartbeat interval (seconds).

The following is supported on AIX, HP-UX, OS/2, OS/400, Solaris, Linux, Windows:

MQIACH_NPM_SPEED
Speed of nonpersistent messages.

Relevant for server-connection channel type

The following is supported on Compaq OpenVMS Alpha, OS/2, OS/400, Compaq NonStop Kernel, UNIX systems, Windows:

MQCACH_MCA_USER_ID
MCA user identifier.

| **MQCACH_SSL_CLIENT_AUTH**
| SSL client authentication.

Relevant for client-connection channel type

The following are supported on Compaq OpenVMS Alpha, OS/2, OS/400, Compaq NonStop Kernel, UNIX systems, Windows:

| **MQCACH_LOCAL_ADDRESS**
| Local communications address for the channel.

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 Compaq OpenVMS Alpha, OS/2, OS/400, Compaq NonStop Kernel, and UNIX systems:

MQCACH_USER_ID
User identifier.

MQCACH_PASSWORD
Password.

Relevant for cluster-receiver channel type

The following are supported on AIX, HP-UX, OS/2, OS/400, Solaris, Linux, Windows:

| **MQCACH_LOCAL_ADDRESS**
| Local communications address for the channel.

MQCACH_MODE_NAME
Mode name.

| **MQCACH_SSL_CLIENT_AUTH**
| SSL client authentication.

MQCACH_TP_NAME
Transaction program name.

MQCACH_CONNECTION_NAME
Connection name.

| **MQIACH_BATCH_HB**
| The value to use for the batch heartbeating.

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_DATA_CONVERSION
Whether sender should convert application data.

MQIACH_BATCH_SIZE
Batch size.

MQIACH_PUT_AUTHORITY
Put authority.

MQIACH_SEQUENCE_NUMBER_WRAP
Sequence number wrap.

MQCACH_MCA_USER_ID
MCA user identifier.

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_HB_INTERVAL
Heartbeat interval (seconds).

MQIACH_NPM_SPEED
Speed of nonpersistent messages.

MQIACH_BATCH_INTERVAL
Batch wait interval (seconds).

MQCA_CLUSTER_NAME
Cluster name.

MQCA_CLUSTER_NAMELIST
Cluster namelist.

MQIACH_NETWORK_PRIORITY
Network priority.

| **MQCACH_LOCAL_ADDRESS**
| Local communications address for the channel.

Relevant for cluster-sender channel type

The following are supported on AIX, HP-UX, OS/2, OS/400, Solaris, Linux, Windows:.

MQCACH_CONNECTION_NAME
Connection name.

| **MQCACH_LOCAL_ADDRESS**
| Local communications address for the channel.

MQCACH_MODE_NAME
Mode name.

MQCACH_TP_NAME
Transaction program name.

| **MQIACH_BATCH_HB**
| The value to use for the batch heartbeating.

MQIACH_BATCH_SIZE
Batch size.

MQIACH_DATA_CONVERSION
Whether sender should convert application data.

MQIACH_DISC_INTERVAL
Disconnection interval.

MQIACH_LONG_RETRY
Long retry count.

MQIACH_LONG_TIMER
Long timer.

MQIACH_MCA_TYPE
MCA type.

MQIACH_SEQUENCE_NUMBER_WRAP
Sequence number wrap.

MQIACH_SHORT_RETRY
Short retry count.

MQIACH_SHORT_TIMER
Short timer.

MQCACH_MCA_NAME
Message channel agent name.

MQCACH_MCA_USER_ID
MCA user identifier.

MQCACH_USER_ID
User identifier.

MQCACH_PASSWORD
Password.

MQIACH_HB_INTERVAL
Heartbeat interval (seconds).

MQIACH_NPM_SPEED
Speed of nonpersistent messages.

MQIACH_BATCH_INTERVAL
Batch wait interval (seconds).

MQCA_CLUSTER_NAME
Cluster name.

MQCA_CLUSTER_NAMELIST
Cluster namelist.

| **MQCACH_LOCAL_ADDRESS**
| Local communications address for the channel.

Error codes

| This command might return the following in the response format header, in
| addition to the values shown on page 18.

Reason (MQLONG)
The value can 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.

This response is supported on all platforms.

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*, *AlterationDate*, *AlterationTime*, *ClusterName*,
| *ClusterNamelist*, *NetworkPriority*, *LocalAddress*, *BatchHeartbeat*
| *SSLCipherSpec*, *SSLPeerName*, *SSLClientAuth*

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 can 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.

MQCHT_CLUSRCVR
Cluster-receiver.

MQCHT_CLUSSDR
Cluster-sender.

TransportType (MQCFIN)
Transmission protocol type (parameter identifier: MQIACH_XMIT_PROTOCOL_TYPE).

The value may be:

MQXPT_LU62
LU 6.2.

MQXPT_TCP
TCP.

MQXPT_NETBIOS
NetBIOS.

MQXPT_SPX
SPX.

MQXPT_DECNET
DECnet.

MQXPT_UDP
UDP.

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 can 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 exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

MsgExit (MQCFSL)
Message exit name (parameter identifier: MQCACH_MSG_EXIT_NAME).

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

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, Solaris, Linux, and Windows.

SendExit (MQCFSL)
Send exit name (parameter identifier: MQCACH_SEND_EXIT_NAME).

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

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, Solaris, Linux, and Windows.

ReceiveExit (MQCFSL)
Receive exit name (parameter identifier: MQCACH_RCV_EXIT_NAME).

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

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, Solaris, Linux, and Windows.

PutAuthority (MQCFIN)
Put authority (parameter identifier: MQIACH_PUT_AUTHORITY).

The value can 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 (MQCFSL)
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, Solaris, Linux, and Windows.

SendUserData (MQCFSL)
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, Solaris, Linux, and Windows.

`ReceiveUserData` (MQCFSL)
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, Solaris, Linux, and Windows.

`MCAType` (MQCFIN)
Message channel agent type (parameter identifier: MQIACH_MCA_TYPE).

The value can be:

MQMCAT_PROCESS
Process.

MQMCAT_THREAD
Thread (OS/2, Windows only).

`MCAUserIdentifier` (MQCFST)
Message channel agent user identifier (parameter identifier: MQCACH_MCA_USER_ID).

The maximum length of the MCA user identifier depends on the environment in which the MCA is running. MQ_MCA_USER_ID_LENGTH gives the maximum length for the environment for which your application is running. MQ_MAX_MCA_USER_ID_LENGTH gives the maximum for all supported environments.

On Windows, the user identifier may be qualified with the domain name in the following format:

```
user@domain
```

`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 exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

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 can be:

MQNPMS_NORMAL
Normal speed.

MQNPMS_FAST
Fast speed.

AlterationDate (MQCFST)
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date when the information was last altered.

AlterationTime (MQCFST)
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time when the information was last altered.

ClusterName (MQCFST)
Cluster name (parameter identifier: MQCA_CLUSTER_NAME).

ClusterNamelist (MQCFSL)
Cluster namelist (parameter identifier: MQCA_CLUSTER_NAMELIST).

NetworkPriority (MQCFIN)
Network priority (parameter identifier: MQIACH_NETWORK_PRIORITY).

| *LocalAddress* (MQCFST)
| Local communications address for the channel (parameter identifier:
| MQCACH_LOCAL_ADDRESS).

| The maximum length of the string is MQ_LOCAL_ADDRESS_LENGTH.

| *BatchHeartbeat* (MQCFIN)
| The value being used for the batch heartbeating (parameter identifier:
| MQIACH_BATCH_HB).

| The value can be between 0 and 999 999. A value of 0 indicates that
| heartbeating is not in use.

| *SSLCipherSpec* (MQCFST)
| CipherSpec (parameter identifier: MQCACH_SSL_CIPHER_SPEC).

| The length of the string is MQ_SSL_CIPHER_SPEC_LENGTH.

This parameter is supported only on AIX, HP-UX, Linux, OS/400, Solaris, Windows, and z/OS.

This parameter is valid only for channels with a transport type (TRPTYPE) of TCP. If the TRPTYPE is not TCP, the data is ignored and no error message is issued.

The SSLCIPH values must specify the same cipher specification on both ends of the channel.

Specify the name of the cipher specification you are using. Alternatively, on OS/400 and z/OS, you can specify the two-digit hexadecimal code.

The following table shows the CipherSpecs that can be used with WebSphere MQ SSL.

Table 3. CipherSpecs that can be used with WebSphere MQ SSL support

CipherSpec name	Hash algorithm	Encryption algorithm	Encryption bits
NULL_MD5[1]	MD5	None	0
NULL_SHA[1]	SHA	None	0
RC4_MD5_EXPORT[1]	MD5	RC4	40
RC4_MD5_US[2]	MD5	RC4	128
RC4_SHA_US[2]	SHA	RC4	128
RC2_MD5_EXPORT[1]	MD5	RC2	40
DES_SHA_EXPORT[1]	SHA	DES	56
RC4_56_SHA_EXPORT1024[3,4,5]	SHA	RC4	56
DES_SHA_EXPORT1024[3,4,5,6]	SHA	DES	56
TRIPLE_DES_SHA_US[4]	SHA	3DES	168
TLS_RSA_WITH_AES_128_CBC_SHA[7]	SHA	AES	128
TLS_RSA_WITH_AES_256_CBC_SHA[7]	SHA	AES	256
AES_SHA_US[8]	SHA	AES	128

Notes:
1. On OS/400, available when either AC2 or AC3 are installed
2. On OS/400, available only when AC3 is installed
3. Not available for z/OS
4. Not available for OS/400
5. Specifies a 1024–bit handshake key size
6. Not available for Windows
7. Available for AIX platforms only
8. Available for OS/400, AC3 only

If the SSLCIPH parameter is blank, no attempt is made to use SSL on the channel.

SSLPeerName (MQCFST)
Peer name (parameter identifier: MQCACH_SSL_PEER_NAME).

The length of the string is MQ_SSL_PEER_NAME_LENGTH.

Specifies the filter to use to compare with the Distinguished Name of the certificate from the peer queue manager or client at the other end of the channel. (A Distinguished Name is the identifier of the SSL certificate.) If the Distinguished Name in the certificate received from the peer does not match the SSLPEER filter, the channel does not start.

This parameter is supported only on AIX, HP-UX, Linux, OS/400, Solaris, Windows, and z/OS.

This parameter is optional; if it is not specified, the Distinguished Name of the peer is not checked at channel start up. (The Distinguished Name from the certificate is still written into the SSLPEER definition held in memory and passed to the security exit). If SSLCIPH is blank, the data is ignored and no error message is issued.

This parameter is valid for all channel types.

The SSLPEER value is specified in the standard form used to specify a Distinguished Name. For example: `SSLPEER('CN="xxx yyy zzz",O=xxx,C=xxx')`

You can use a semi-colon as a separator instead of a comma.

The possible attribute types supported are:

CN	common name
T	title
OU	organizational unit name
O	organization name
L	locality name
ST, SP or S	state or province name
C	country

WebSphere MQ accepts only upper case letters for the attribute types.

If any of the unsupported attribute types are specified in the SSLPEER string, an error is output either when the attribute is defined or at run time (depending on which platform you are running on), and the string is deemed not to have matched the flowed certificate's Distinguished Name.

If the flowed certificate's Distinguished Name contains multiple OU (organizational unit) attributes, and SSLPEER specifies these attributes to be compared, they must match in the order that they are found in the certificate's Distinguished Name, and must start with the first OU, or an asterisk. For example, if the flowed certificate's Distinguished Name contains the OUs `OU=One,OU=Two,OU=Three`, specifying the following SSLPEER values will work:

```
('OU=One,OU=Two')
```

```
('OU=*,OU=Two,OU=Three')
```

```
('OU=*,OU=Two')
```

but specifying the following SSLPEER values will fail:

```
('OU=Two,OU=Three')
```

```
('OU=One,OU=Three')
```

```
('OU=Two')
```

Any or all of the attribute values can be generic, either an asterisk (*) on its own, or a stem with initiating or trailing asterisks. This allows the SSLPEER to match any Distinguished Name value, or any value starting with the stem for that attribute.

If an asterisk is specified at the beginning or end of an attribute value in the Distinguished Name on the certificate, you can specify * to check for an exact match in SSLPEER. For example, if you have an attribute of CN=Test* in the Distinguished Name of the certificate, you can use the following command:

```
SSLPEER('CN=Test\*')
```

SSLClientAuth (MQCFIN)
Client authentication (parameter identifier: MQCACH_SSL_CLIENT_AUTH).

The value can be

MQSCA_REQUIRED
Client authentication required

MQSCA_OPTIONAL
Client authentication is optional.

Defines whether WebSphere MQ requires a certificate from the SSL client.

The initiating end of the channel acts as the SSL client, so this applies to the end of the channel that receives the initiation flow, which acts as the SSL server.

This parameter is supported only on AIX, HP-UX, Linux, OS/400, Solaris, Windows, and z/OS.

The parameter is used only for channels with SSLCIPH specified. If SSLCIPH is blank, the data is ignored and no error message is issued.

Inquire Channel Names

The Inquire Channel Names (MQCMD_INQUIRE_CHANNEL_NAMES) command inquires a list of WebSphere MQ 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 can 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.

MQCHT_CLUSRCVR
Cluster-receiver.

MQCHT_CLUSSDR
Cluster-sender.

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

| This command might return the following in the response format header, in
| addition to the values shown on 18.

Reason (MQLONG)
The value can 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.

This response is supported on all platforms.

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 WebSphere MQ 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.

  Status for all channels that meet the selection criteria is given, whether the channels were defined manually or automatically.

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 can 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* might 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 might 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 might have different values for saved and current status.
- A current channel always has current status and might have saved status.

Channels can 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 can 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, cluster-sender, cluster-receiver, 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.

This PCF is supported on all platforms.

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 must not include a part number. It must specify only the TCP 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 can 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 might specify the following on its own:

MQIACF_ALL
All attributes.

This is the default value used if the parameter is not specified or it can specify 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.

MQCA_REMOTE_Q_MGR_NAME
: Queue manager name, or queue-sharing group name of the remote system. The remote queue manager name is always returned regardless of the instance attributes requested.

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_LOCAL_ADDRESS
: Local communications address for the channel.

MQCACH_MCA_JOB_NAME
Name of MCA job.

MQCACH_SSL_SHORT_PEER_NAME
SSL short peer name.

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 is supported on Compaq OpenVMS Alpha, OS/2, OS/400, Compaq NonStop Kernel, UNIX systems, and Windows:

MQIACH_BATCH_SIZE
Batch size.

The following is supported on Compaq OpenVMS Alpha, Compaq NonStop Kernel, OS/2, OS/400, UNIX systems, and Windows:

MQIACH_HB_INTERVAL
Heartbeat interval (seconds).

The following is supported on Compaq OpenVMS Alpha, Compaq NonStop Kernel, OS/2, OS/400, UNIX systems, and Windows:

MQIACH_NPM_SPEED
Speed of nonpersistent messages.

MQIACH_BATCHES, MQIACH_LONG_RETRIES_LEFT, MQIACH_SHORT_RETRIES_LEFT, MQIACH_BATCH_SIZE, MQIACH_HB_INTERVAL, MQIACH_NPM_SPEED, and MQCA_REMOTE_Q_MGR_NAME do not apply to server-connection channels, and no values are returned. If specified on the command they are ignored.

Error codes

This command might return the following in the response format header, in addition to the values shown on page 18.

Reason (MQLONG)
The value can 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,
- The *ChannelStatus* structure, and
- | The *RemoteQMgrName* structure

which are followed by the requested combination of status attribute parameter structures. One such message is generated for each channel instance found that matches the criteria specified on the command.

This response is supported on all platforms.

Always returned:
ChannelName, *XmitQName*, *ConnectionName*, *ChannelInstanceType*, | *ChannelType*, *ChannelStatus*, *RemoteQMgrName*

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*, *SSLShortPeerName*, *LocalAddress*

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 can be:

MQOT_CURRENT_CHANNEL
Current channel status.

MQOT_SAVED_CHANNEL
Saved channel status.

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

The value can 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.

MQCHT_CLUSRCVR
Cluster-receiver.

MQCHT_CLUSSDR
Cluster-sender.

ChannelStatus (MQCFIN)
Channel status (parameter identifier: MQIACH_CHANNEL_STATUS).

The value can 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 can 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 can 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 can 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 can be:

MQNPMS_NORMAL
Normal speed.

MQNPMS_FAST
Fast speed.

RemoteQMgrName (MQCFST)
Name of the remote queue manager, or queue-sharing group (parameter identifier: MQCA_REMOTE_Q_MGR_NAME).

LocalAddress (MQCFST)
Local communications address for the channel (parameter identifier: MQCACH_LOCAL_ADDRESS).

The maximum length of the string is MQ_LOCAL_ADDRESS_LENGTH.

The value shown depends on the transport type *(TransportType)* of the channel shown:

TCP/IP
The format for this information is as follows:

```
[ip-addr][(port)]
```

Inquire Cluster Queue Manager

The Inquire Cluster Queue Manager (MQCMD_INQUIRE_CLUSTER_Q_MGR) command inquires about the attributes of WebSphere MQ queue managers in a cluster.

This PCF is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

Required parameters:
ClusterQMgrName

Optional parameters:
Channel, *ClusterName*, *ClusterQMgrAttrs*

Required parameters

ClusterQMgrName (MQCFST)
Queue manager name (parameter identifier: MQCA_CLUSTER_Q_MGR_NAME).

Generic queue manager names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all queue managers having names that start with the selected character string. An asterisk on its own matches all possible names.

The queue manager name is always returned, regardless of the attributes requested.

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

Optional parameters

Channel (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 maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

If you do not specify a value for this parameter, channel information about *all* queue managers in the cluster is automatically returned.

ClusterName (MQCFST)
Cluster name (parameter identifier: MQCA_CLUSTER_NAME).

Generic cluster names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all clusters 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_CLUSTER_NAME_LENGTH.

If you do not specify a value for this parameter, cluster information about *all* queue managers inquired is automatically returned.

ClusterQMgrAttrs (MQCFIL)
Attributes (parameter identifier: MQIACF_CLUSTER_Q_MGR_ATTRS).

The attribute list might 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_ALTERATION_DATE
The date on which the information was last altered, in the form `yyyy-mm-dd`.

MQCA_ALTERATION_TIME
The time at which the information was last altered, in the form `hh.mm.ss`.

MQCA_CLUSTER_DATE
The date on which the information became available to the local queue manager.

MQCA_CLUSTER_NAME
The name of the cluster to which the channel belongs.

MQCA_CLUSTER_TIME
The time at which the information became available to the local queue manager.

MQCA_Q_MGR_IDENTIFIER
The unique identifier of the queue manager.

MQCACH_CONNECTION_NAME
Connection name.

MQCACH_DESCRIPTION
Description.

| **MQCACH_LOCAL_ADDRESS**
| Local communications address for the channel.

MQCACH_MCA_NAME
Message channel agent name.

MQCACH_MCA_USER_ID
MCA user identifier.

MQCACH_MODE_NAME
Mode name.

MQCACH_MR_EXIT_NAME
Message-retry exit name.

MQCACH_MR_EXIT_USER_DATA
Message-retry exit user data.

MQCACH_MSG_EXIT_NAME
Message exit name.

MQCACH_MSG_EXIT_USER_DATA
Message exit user data.

MQCACH_PASSWORD
Password.

MQCACH_RCV_EXIT_NAME
Receive exit name.

MQCACH_RCV_EXIT_USER_DATA
Receive exit user data.

MQCACH_SEC_EXIT_NAME
Security exit name.

MQCACH_SEC_EXIT_USER_DATA
Security exit user data.

MQCACH_SEND_EXIT_NAME
Send exit name.

MQCACH_SEND_EXIT_USER_DATA
Send exit user data.

MQCACH_SSL_CIPHER_SPEC
SSL cipher spec.

MQCACH_SSL_CLIENT_AUTH
SSL client authentication.

MQCACH_SSL_PEER_NAME
SSL peer name.

MQCACH_TP_NAME
Transaction program name.

MQCACH_USER_ID
User identifier.

MQIACF_Q_MGR_DEFINITION_TYPE
How the cluster queue manager was defined.

MQIACF_Q_MGR_TYPE
The function of the queue manager in the cluster.

MQIACF_SUSPEND
Whether the queue manager is suspended from the cluster.

MQIACH_BATCH_HB
The value being used for batch heartbeating.

MQIACH_BATCH_INTERVAL
Batch wait interval (seconds).

MQIACH_BATCH_SIZE
Batch size.

MQIACH_CHANNEL_STATUS
Channel status.

MQIACH_DATA_CONVERSION
Whether sender must convert application data.

MQIACH_DISC_INTERVAL
Disconnection interval.

MQIACH_HB_INTERVAL
Heartbeat interval (seconds).

MQIACH_LONG_RETRY
Long retry count.

MQIACH_LONG_TIMER
Long timer.

MQIACH_MAX_MSG_LENGTH
Maximum message length.

MQIACH_MCA_TYPE
MCA type.

MQIACH_MR_COUNT
Message retry count.

MQIACH_MR_INTERVAL
Message retry interval (milliseconds).

MQIACH_NETWORK_PRIORITY
Network priority.

MQIACH_NPM_SPEED
Speed of nonpersistent messages.

MQIACH_PUT_AUTHORITY
Put authority.

MQIACH_SEQUENCE_NUMBER_WRAP
Sequence number wrap.

MQIACH_SHORT_RETRY
Short retry count.

MQIACH_SHORT_TIMER
Short timer.

MQIACH_XMIT_PROTOCOL_TYPE
Transmission protocol type.

Error codes

This command might return the following in the response format header, in addition to the values shown on page 18.

Reason (MQLONG)
The value can 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 Cluster Queue Manager (Response)

The response to the Inquire Cluster Queue Manager (MQCMD_INQUIRE_CLUSTER_Q_MGR) command consists of the response header followed by the *QMgrName* structure and the requested combination of attribute parameter structures.

This response is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

Always returned:
QMgrName, *ChannelName*, *ClusterName*

Returned if requested:

TransportType, *ModeName*, *TpName*, *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*, *AlterationDate*, *AlterationTime*, *ClusterInfo*, *QMgrDefinitionType*, *QMgrType*, *QMgrIdentifier*, *ClusterDate*, *ClusterTime*, *ChannelStatus*,
| *Suspend*, *NetworkPriority*, *BatchHeartbeat*, *LocalAddress*, *SSLCipherSpec*,
| *SSLPeerName*, *SSLClientAuth*

Response data

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

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

TransportType (MQCFIN)
Transmission protocol type (parameter identifier: MQIACH_XMIT_PROTOCOL_TYPE).

The value can be:

MQXPT_LU62
LU 6.2.

MQXPT_TCP
TCP.

MQXPT_NETBIOS
NetBIOS.

MQXPT_SPX
SPX.

MQXPT_DECNET
DECnet.

MQXPT_UDP
UDP.

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.

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 must convert application data (parameter identifier: MQIACH_DATA_CONVERSION).

The value can 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 (MQCFSL)
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/400, Solaris, Linux, and Windows.

SendExit (MQCFSL)
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/400, Solaris, Linux, and Windows.

ReceiveExit (MQCFSL)
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/400, Solaris, Linux, and Windows.

PutAuthority (MQCFIN)
Put authority (parameter identifier: MQIACH_PUT_AUTHORITY).

The value can 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 (MQCFSL)
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/400, Solaris, Linux, and Windows.

SendUserData (MQCFSL)
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/400, Solaris, Linux, and Windows.

ReceiveUserData (MQCFSL)
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/400, Solaris, Linux, and Windows.

MCAType (MQCFIN)
Message channel agent type (parameter identifier: MQIACH_MCA_TYPE).

The value can be:

MQMCAT_PROCESS
Process.

MQMCAT_THREAD
Thread (Windows 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).

AlterationDate (MQCFST)
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date at which the information was last altered.

AlterationTime (MQCFST)
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time at which the information was last altered.

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 can be:

MQNPMS_NORMAL
Normal speed.

MQNPMS_FAST
Fast speed.

ClusterName (MQCFST)
Cluster name (parameter identifier: MQCA_CLUSTER_NAME).

QMgrDefinitionType (MQCFIN)
Queue manager definition type (parameter identifier: MQIACF_Q_MGR_DEFINITION_TYPE).

The value can be:

MQQMDT_EXPLICIT_CLUSTER_SENDER
A cluster-sender channel from an explicit definition.

MQQMDT_AUTO_CLUSTER_SENDER
A cluster-sender channel by auto-definition.

MQQMDT_CLUSTER_RECEIVER
A cluster-receiver channel.

MQQMDT_AUTO_EXP_CLUSTER_SENDER
A cluster-sender channel, both from an explicit definition and by auto-definition.

QMgrType (MQCFIN)
Queue manager type (parameter identifier: MQIACF_Q_MGR_TYPE).

The value can be:

MQQMT_NORMAL
A normal queue manager.

MQQMT_REPOSITORY
A repository queue manager.

QMgrIdentifier (MQCFST)
Queue manager identifier (parameter identifier: MQCA_Q_MGR_IDENTIFIER).

The unique identifier of the queue manager.

ClusterDate (MQCFST)
Cluster date (parameter identifier: MQCA_CLUSTER_DATE).

The date at which the information became available to the local queue manager.

ClusterInfo (MQCFIN)
Cluster information (parameter identifier: MQIACF_CLUSTER_INFO).

The cluster information available to the local queue manager.

ClusterTime (MQCFST)
Cluster time (parameter identifier: MQCA_CLUSTER_TIME).

The time at which the information became available to the local queue manager.

ChannelStatus (MQCFIN)
Channel status (parameter identifier: MQIACH_CHANNEL_STATUS).

The value can be:

MQCHS_BINDING
Channel is negotiating with the partner.

MQCHS_INACTIVE
Channel is not active.

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.

This parameter is returned if the channel is a cluster-sender channel (CLUSSDR) only.

Suspend (MQCFIN)
Whether the queue manager is suspended (parameter identifier: MQIACF_SUSPEND).

The value can be:

MQSUS_NO
The queue manager is not suspended from the cluster.

MQSUS_YES
The queue manager is suspended from the cluster.

NetworkPriority (MQCFIN)
Network priority (parameter identifier: MQIACF_NETWORK_PRIORITY).

| *BatchHeartbeat* (MQCFIN)
| The value being used for batch heartbeating (parameter identifier:
| MQIACH_BATCH_HB).

| The value can be between 0 and 999 999. A value of 0 indicates that batch
| heartbeating is not being used.

| *LocalAddress* (MQCFST)
| Local communications address for the channel (parameter identifier:
| MQCACH_LOCAL_ADDRESS).

| The maximum length of the string is MQ_LOCAL_ADDRESS_LENGTH.

| *SSLCipherSpec* (MQCFST)
| CipherSpec (parameter identifier: MQCACH_SSL_CIPHER_SPEC).

| The length of the string is MQ_SSL_CIPHER_SPEC_LENGTH.

| This parameter is supported only on AIX, HP-UX, Linux, OS/400, Solaris,
| Windows, and z/OS.

This parameter is valid only for channels with a transport type (TRPTYPE) of TCP. If the TRPTYPE is not TCP, the data is ignored and no error message is issued.

The SSLCIPH values must specify the same CipherSpec on both ends of the channel.

Specify the name of the CipherSpec you are using. Alternatively, on OS/400 and z/OS, you can specify the two-digit hexadecimal code.

The following table shows the CipherSpecs that can be used with WebSphere MQ SSL.

Table 4. CipherSpecs that can be used with WebSphere MQ SSL support

CipherSpec name	Hash algorithm	Encryption algorithm	Encryption bits
NULL_MD5[1]	MD5	None	0
NULL_SHA[1]	SHA	None	0
RC4_MD5_EXPORT[1]	MD5	RC4	40
RC4_MD5_US[2]	MD5	RC4	128
RC4_SHA_US[2]	SHA	RC4	128
RC2_MD5_EXPORT[1]	MD5	RC2	40
DES_SHA_EXPORT[1]	SHA	DES	56
RC4_56_SHA_EXPORT1024[3,4,5]	SHA	RC4	56
DES_SHA_EXPORT1024[3,4,5,6]	SHA	DES	56
TRIPLE_DES_SHA_US[4]	SHA	3DES	168
TLS_RSA_WITH_AES_128_CBC_SHA[7]	SHA	AES	128
TLS_RSA_WITH_AES_256_CBC_SHA[7]	SHA	AES	256
AES_SHA_US[8]	SHA	AES	128

Notes:
1. On OS/400, available when either AC2 or AC3 are installed
2. On OS/400, available only when AC3 is installed
3. Not available for z/OS
4. Not available for OS/400
5. Specifies a 1024–bit handshake key size
6. Not available for Windows
7. Available for AIX platforms only
8. Available for OS/400, AC3 only

If the SSLCIPH parameter is blank, no attempt is made to use SSL on the channel.

SSLPeerName (MQCFST)
Peer name (parameter identifier: MQCACH_SSL_PEER_NAME).

The length of the string is MQ_SSL_PEER_NAME_LENGTH.

Specifies the filter to use to compare with the Distinguished Name of the certificate from the peer queue manager or client at the other end of the channel. (A Distinguished Name is the identifier of the SSL certificate.) If the Distinguished Name in the certificate received from the peer does not match the SSLPEER filter, the channel does not start.

This parameter is supported only on AIX, HP-UX, Linux, OS/400, Solaris, Windows, and z/OS.

This parameter is optional; if it is not specified, the Distinguished Name of the peer is not checked at channel start up. (The Distinguished Name from the certificate is still written into the SSLPEER definition held in memory, and passed to the security exit). If SSLCIPH is blank, the data is ignored and no error message is issued.

This parameter is valid for all channel types.

The SSLPEER value is specified in the standard form used to specify a Distinguished Name. For example: `SSLPEER('CN="xxx yyy zzz",O=xxx,C=xxx')`

You can use a semi-colon as a separator instead of a comma.

The possible attribute types supported are:

CN	common name
T	title
OU	organizational unit name
O	organization name
L	locality name
ST, SP or S	state or province name
C	country

WebSphere MQ accepts only upper case letters for the attribute types.

If any of the unsupported attribute types are specified in the SSLPEER string, an error is output either when the attribute is defined or at run time (depending on which platform you are running on), and the string is deemed not to have matched the flowed certificate's Distinguished Name.

If the flowed certificate's Distinguished Name contains multiple OU (organizational unit) attributes, and SSLPEER specifies these attributes to be compared, they must match in the order that they are found in the certificate's Distinguished Name, and must start with the first OU, or an asterisk. For example, if the flowed certificate's Distinguished Name contains the OUs `OU=One,OU=Two,OU=Three`, specifying the following SSLPEER values will work:

```
('OU=One,OU=Two')
```

```
('OU=*,OU=Two,OU=Three')
```

```
('OU=*,OU=Two')
```

but specifying the following SSLPEER values will fail:

```
('OU=Two,OU=Three')
```

```
('OU=One,OU=Three')
```

```
('OU=Two')
```

Any or all of the attribute values can be generic, either an asterisk (*) on its own, or a stem with initiating or trailing asterisks. This allows the SSLPEER to match any Distinguished Name value, or any value starting with the stem for that attribute.

If an asterisk is specified at the beginning or end of an attribute value in the Distinguished Name on the certificate, you can specify * to check for an exact match in SSLPEER. For example, if you have an attribute of CN=Test* in the Distinguished Name of the certificate, you can use the following command:

```
SSLPEER('CN=Test\*')
```

SSLClientAuth (MQCFIN)
Client authentication (parameter identifier: MQCACH_SSL_CLIENT_AUTH).

The value can be:

MQSCA_REQUIRED
Client authentication required

MQSCA_OPTIONAL
Client authentication is optional.

Defines whether WebSphere MQ requires a certificate from the SSL client.

The initiating end of the channel acts as the SSL client, so this applies to the end of the channel that receives the initiation flow, which acts as the SSL server.

This parameter is supported only on AIX, HP-UX, Linux, OS/400, Solaris, Windows, and z/OS.

The parameter is used only for channels with SSLCIPH specified. If SSLCIPH is blank, the data is ignored and no error message is issued.

Inquire Namelist

The Inquire Namelist (MQCMD_INQUIRE_NAMELIST) command inquires about the attributes of existing WebSphere MQ namelists.

This PCF is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

Required parameters:
NamelistName

Optional parameters:
NamelistAttrs

Required parameters

NamelistName (MQCFST)
Namelist name (parameter identifier: MQCA_NAMELIST_NAME).

This is the name of the namelist whose attributes are required. Generic namelist names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all namelists having names that start with the selected character string. An asterisk on its own matches all possible names.

The namelist name is always returned regardless of the attributes requested.

The maximum length of the string is MQ_NAMELIST_NAME_LENGTH.

Optional parameters

NamelistAttrs (MQCFIL)
Namelist attributes (parameter identifier: MQIACF_NAMELIST_ATTRS).

The attribute list might specify the following on its own (this is the default value if the parameter is not specified):

MQIACF_ALL
All attributes.

or a combination of the following:

MQCA_NAMELIST_NAME
Name of namelist object.

MQCA_NAMELIST_DESC
Namelist description.

MQCA_NAMES
Names in the namelist.

MQCA_ALTERATION_DATE
The date on which the information was last altered, in the form `yyyy-mm-dd`.

MQCA_ALTERATION_TIME
The time at which the information was last altered, in the form `hh.mm.ss`.

| **MQIA_NAME_COUNT**
| Number of names in the namelist

Error codes

| This command might return the following in the response format header, in
| addition to the values shown on page 18.

Reason (MQLONG)
The value can 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 not valid.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier 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 Namelist (Response)

The response to the Inquire Namelist (MQCMD_INQUIRE_NAMELIST) command consists of the response header followed by the *NamelistName* structure and the requested combination of attribute parameter structures. If a generic namelist name was specified, one such message is generated for each namelist found.

This response is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

Always returned:
NamelistName

Returned if requested:
NamelistDesc, *Names*, *AlterationDate*, *AlterationTime*, *NameCount*

Response data

NamelistName (MQCFST)
The name of the namelist definition (parameter identifier: MQCA_NAMELIST_NAME).

The maximum length of the string is MQ_NAMELIST_NAME_LENGTH.

NamelistDesc (MQCFST)
Description of namelist definition (parameter identifier: MQCA_NAMELIST_DESC).

The maximum length of the string is MQ_NAMELIST_DESC_LENGTH.

Names (MQCFSL)
The names contained in the namelist (parameter identifier: MQCA_NAMES).

The number of names in the list is given by the *Count* field in the MQCFSL structure. The length of each name is given by the *StringLength* field in that structure. The maximum length of a name is MQ_OBJECT_NAME_LENGTH.

AlterationDate (MQCFST)
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date when the information was last altered.

AlterationTime (MQCFST)
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time when the information was last altered.

NameCount (MQCFIN)
Number of names in the namelist (parameter identifier: MQIA_NAME_COUNT).

The number of names contained in the namelist.

Inquire Namelist Names

The Inquire Namelist Names (MQCMD_INQUIRE_NAMELIST_NAMES) command inquires for a list of namelist names that match the generic namelist name specified.

This PCF is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

Required parameters:
NamelistName

Optional parameters:
None

Required parameters

NamelistName (MQCFST)
Name of namelist (parameter identifier: MQCA_NAMELIST_NAME).

Generic namelist 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

This command might return the following in the response format header, in addition to the values shown on page 18.

Reason (MQLONG)
The value can be:

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier 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 Namelist Names (Response)

The response to the Inquire Namelist Names (MQCMD_INQUIRE_NAMELIST_NAMES) command consists of the response header followed by a single parameter structure giving zero or more names that match the specified namelist name.

Always returned:
NamelistNames

Returned if requested:
None

Response data

NamelistNames (MQCFSL)
Namelist Names (parameter identifier: MQCACF_NAMELIST_NAMES).

Inquire Process

The Inquire Process (MQCMD_INQUIRE_PROCESS) command inquires about the attributes of existing WebSphere MQ processes.

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 might 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.

MQCA_ALTERATION_DATE
The date at which the information was last altered, in the form `yyyy-mm-dd`.

This attribute is supported on AIX, HP-UX, OS/2, OS/400, Solaris, Linux, Windows only.

MQCA_ALTERATION_TIME
The time at which the information was last altered, in the form `hh.mm.ss`.

This attribute is supported on AIX, HP-UX, OS/2, OS/400, Solaris, Linux, Windows only.

Error codes

| This command might return the following in the response format header, in addition to the values shown on page 18.

Reason (MQLONG)
The value can 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.

Always returned:
ProcessName

Returned if requested:
ProcessDesc, *ApplType*, *ApplId*, *EnvData*, *UserData*, *AlterationDate*, *AlterationTime*

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 can be:

MQAT_OS400
OS/400 application.

MQAT_OS2
OS/2 or Presentation Manager application.

MQAT_DOS
DOS client application.

MQAT_WINDOWS
Windows client or Windows 3.1 application.

MQAT_WINDOWS_NT
Windows or Windows 95, Windows 98 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.

AlterationDate (MQCFST)
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date when the information was last altered.

AlterationTime (MQCFST)
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time when the information was last altered.

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.

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

| This command might return the following in the response format header, in
| addition to the values shown on page 18.

Reason (MQLONG)
The value can 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 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 WebSphere MQ queues.

Required parameters:
QName

Optional parameters:
QType, *ClusterName*, *ClusterNamelist*, *ClusterInfo*, *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 valid only 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 need 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 can be:

MQQT_ALL
All queue types.

MQQT_LOCAL
Local queue.

MQQT_ALIAS
Alias queue definition.

MQQT_REMOTE
Local definition of a remote queue.

MQQT_CLUSTER
Cluster queue.

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.

ClusterName (MQCFST)
Cluster name (parameter identifier: MQCA_CLUSTER_NAME).

The name of the cluster to which the channel belongs.

Generic cluster names are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all clusters 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_CLUSTER_NAME_LENGTH.

This parameter is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

ClusterNamelist (MQCFST)
Cluster namelist (parameter identifier: MQCA_CLUSTER_NAMELIST).

The name, of the namelist, that specifies a list of clusters to which the channel belongs.

Generic cluster namelists are supported. A generic name is a character string followed by an asterisk (*), for example ABC*, and it selects all cluster namelists having names that start with the selected character string. An asterisk on its own matches all possible names.

This parameter is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

`ClusterInfo` (MQCFIN)
Cluster information (parameter identifier: MQIACF_CLUSTER_INFO).

This parameter requests that, in addition to information about attributes of queues defined on this queue manager, cluster information about these and other queues in the repository that match the selection criteria will be displayed.

In this case, there might be multiple queues with the same name displayed. The cluster information is shown with a queue type of MQQT_CLUSTER.

The cluster information is obtained locally from the queue manager.

This parameter is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

`QAttrs` (MQCFIL)
Queue attributes (parameter identifier: MQIACF_Q_ATTRS).

The attribute list might 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.

The following are supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

MQCA_ALTERATION_DATE
The date on which the information was last altered, in the form `yyyy-mm-dd`.

MQCA_ALTERATION_TIME
The time at which the information was last altered, in the form `hh.mm.ss`.

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.

The following are supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

MQCA_CLUSTER_NAME
Cluster name.

MQCA_CLUSTER_NAMELIST
Cluster namelist.

MQIA_DEF_BIND
Default binding.

Relevant for cluster QType:

The following are supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

MQCA_CLUSTER_NAME
Cluster name.

MQCA_CLUSTER_Q_MGR_NAME
Queue manager name that hosts the queue.

MQCA_Q_MGR_IDENTIFIER
Internally generated queue manager name.

MQCA_CLUSTER_DATE
Date when the definition became available to the local queue manager.

MQCA_CLUSTER_TIME
Time when the definition became available to the local queue manager.

MQIA_CLUSTER_Q_TYPE
Cluster queue type.

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 are supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

MQIA_DIST_LISTS
Distribution list support.

MQCA_CLUSTER_NAME
Cluster name.

MQCA_CLUSTER_NAMELIST
Cluster name.

MQIA_DEF_BIND
Default binding.

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.

The following are supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

MQCA_CLUSTER_NAME
Cluster name.

MQCA_CLUSTER_NAMELIST
Cluster name.

MQIA_DEF_BIND
Default binding.

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 only in the environments in which an MQSeries Version 5.1 product, or later, is available.

MQIA_DIST_LISTS
Distribution list support.

Error codes

| This command might return the following in the response format header, in
| addition to the values shown on page 18.

Reason (MQLONG)
The value can 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, or cluster queues requested (either by using MQQT_CLUSTER or MQIACF_CLUSTER_INFO), 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*, *AlterationDate*, *AlterationTime*, *ClusterDate*, *ClusterTime*, *ClusterName*, *ClusterNamelist*, *ClusterQType*, *DefBind*, *QMgrName*, *QMgrIdentifier*

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 can be:

MQQT_ALIAS
Alias queue definition.

MQQT_CLUSTER
Cluster 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 can 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 can 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 can 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 can 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 can 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 can 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 can 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 can 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 only in the environments in which an MQSeries Version 5.1 product, or later, is available.

The value can be:

MQDL_SUPPORTED
Distribution lists supported.

MQDL_NOT_SUPPORTED
Distribution lists not supported.

Usage (MQCFIN)
Usage (parameter identifier: MQIA_USAGE).

The value can 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 can be:

MQTC_OFF
Trigger messages not required.

MQTC_ON
Trigger messages required.

TriggerType (MQCFIN)
Trigger type (parameter identifier: MQIA_TRIGGER_TYPE).

The value can 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).

The value can 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).

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).

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).

The value can 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).

The value can 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).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

MQEVR_ENABLED
Event reporting enabled.

QServiceInterval (MQCFIN)
Target for queue service interval (parameter identifier: MQIA_Q_SERVICE_INTERVAL).

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).

The value can 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.

AlterationDate (MQCFST)
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date when the information was last altered.

AlterationTime (MQCFST)
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time when the information was last altered.

ClusterDate (MQCFST)
Cluster date (parameter identifier: MQCA_CLUSTER_DATE).

The date on which the information became available to the local queue manager.

ClusterName (MQCFST)
Cluster name (parameter identifier: MQCA_CLUSTER_NAME).

ClusterNamelist (MQCFST)
Cluster namelist (parameter identifier: MQCA_CLUSTER_NAMELIST).

ClusterTime (MQCFST)
Cluster time (parameter identifier: MQCA_CLUSTER_TIME).

The time at which the information became available to the local queue manager.

ClusterQType (MQCFIN)
Cluster queue type (parameter identifier: MQIA_CLUSTER_Q_TYPE).

The value can be:

MQCQT_LOCAL_Q
The cluster queue represents a local queue.

MQCQT_ALIAS_Q
The cluster queue represents an alias queue.

MQCQT_REMOTE_Q
The cluster queue represents a remote queue.

MQCQT_Q_MGR_ALIAS
The cluster queue represents a queue manager alias.

DefBind (MQCFIN)
Default binding (parameter identifier: MQIA_DEF_BIND).

The value can be:

MQBND_BIND_ON_OPEN
Binding fixed by MQOPEN call.

MQBND_BIND_NOT_FIXED
Binding not fixed.

QMgrName (MQCFST)
Name of local queue manager (parameter identifier: MQCA_CLUSTER_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

QMgrIdentifier (MQCFST)
Queue manager identifier (parameter identifier: MQCA_Q_MGR_IDENTIFIER).

The unique identifier of the queue manager.

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 might 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.

MQCACH_LOCAL_ADDRESS
Local communications address for the channel.

The following attributes are supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

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.

MQCA_CLUSTER_WORKLOAD_DATA
Data passed to the cluster workload exit.

MQCA_CLUSTER_WORKLOAD_EXIT
Name of the cluster workload exit.

MQIA_CLUSTER_WORKLOAD_LENGTH
Maximum length of the message passed to the cluster workload exit.

MQCA_REPOSITORY_NAME
Cluster name for the queue manager repository.

MQIA_REPOSITORY_NAMELIST
Name of the list of clusters for which the queue manager is providing a repository manager service.

MQCA_Q_MGR_IDENTIFIER
Internally generated unique queue manager name.

MQCA_ALTERATION_DATE
Date at which the definition was last altered.

MQCA_ALTERATION_TIME
Time at which the definition was last altered.

MQCA_SSL_KEY_REPOSITORY
Location and name of the SSL key repository.

MQCA_SSL_CRL_NAMELIST
SSL Certification Revocation List (CRL) namelist.

MQCA_SSL_CRYPTO_HARDWARE
Parameters to configure the SSL cryptographic hardware. This parameter is supported on UNIX platforms only.

Error codes

This command might return the following in the response format header, in addition to the values shown on page 18.

Reason (MQLONG)
The value can 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.

This response is supported on all platforms.

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*, *AlterationDate*, *AlterationTime*, *ClusterWorkloadExit*, *ClusterWorkloadData*, *ClusterWorkloadLength*, *QMgrIdentifier*, *RepositoryName*, *RepositoryNamelist*, *SSLKeyRepository*, *SSLNamelist*, *SSLCryptoHardware*

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 can be:

MQPL_OS400
OS/400.

MQPL_UNIX
UNIX systems.

MQPL_AIX
AIX (same value as MQPL_UNIX).

MQPL_WINDOWS_NT
Windows.

MQPL_NSK
Compaq NonStop Kernel.

MQPL_VMS
Compaq OpenVMS Alpha.

CommandLevel (MQCFIN)
Command level supported by queue manager (parameter identifier: MQIA_COMMAND_LEVEL).

The value can be:

MQCMDL_LEVEL_1
Level 1 of system control commands.

This value is returned by the following:
- MQSeries for AIX V2.2
- MQSeries for MVS/ESA™:
  - V1.1.1
  - V1.1.2
  - V1.1.3
- MQSeries for OS/2 V2.0
- MQSeries for OS/400:
  - V2R3
  - V3R1
  - V3R6
- MQSeries for Windows V2.0

MQCMDL_LEVEL_101
MQSeries for Windows V2.0.1

MQCMDL_LEVEL_110
MQSeries for Windows V2.1

MQCMDL_LEVEL_200
MQSeries for Windows NT V2.0

MQCMDL_LEVEL_201
MQSeries for OS/2 V2.0.1

MQCMDL_LEVEL_210
MQSeries for OS/390® V2.1

MQCMDL_LEVEL_220
Level 220 of system control commands.

This value is returned by the following:
- MQSeries for AT®&T GIS UNIX V2.2
- MQSeries for SINIX and DC/OSx V2.2
- MQSeries for Compaq NonStop Kernel V2.2.0.1

MQCMDL_LEVEL_221
Level 221 of system control commands.

This value is returned by the following:
- MQSeries for AIX Version 2.2.1
- MQSeries for DIGITAL UNIX (Compaq Tru64 UNIX) V2.2.1

MQCMDL_LEVEL_320
MQSeries for OS/400 V3R2 and V3R7

MQCMDL_LEVEL_420
MQSeries for AS/400® V4R2 and R2.1

MQCMDL_LEVEL_500
Level 500 of system control commands.

This value is returned by the following:
- MQSeries for AIX V5.0
- MQSeries for HP-UX V5.0
- MQSeries for OS/2 Warp V5.0
- MQSeries for Solaris V5.0
- MQSeries for Windows NT V5.0

MQCMDL_LEVEL_510
Level 510 of system control commands.

This value is returned by the following:
- MQSeries for AIX V5.1
- MQSeries for AS/400 V5.1
- MQSeries for HP-UX V5.1
- MQSeries for OS/2 Warp V5.1
- MQSeries for Compaq Tru64 UNIX, V5.1
- MQSeries for Compaq OpenVMS Alpha, V5.1
- MQSeries for Compaq NonStop Kernel, V5.1
- MQSeries for Solaris V5.1
- MQSeries for Windows NT V5.1

MQCMDL_LEVEL_520
Level 520 of system control commands.

This value is returned by the following:
- MQSeries for AIX V5.2
- MQSeries for AS/400 V5.2
- MQSeries for HP-UX V5.2
- MQSeries for Linux V5.2
- MQSeries for OS/390 V5.2
- MQSeries for Solaris V5.2
- MQSeries for Windows NT V5.2
- MQSeries for Windows 2000 V5.2

MQCMDL_LEVEL_530
Level 530 of system control commands.

This value is returned by the following:
- WebSphere MQ for AIX, V5.3
- WebSphere MQ for iSeries, V5.3
- WebSphere MQ for HP-UX, V5.3

- WebSphere MQ for Linux for Intel and Linux for zSeries, V5.3
- WebSphere MQ for Solaris, V5.3
- WebSphere MQ for Windows, V5.3

MQCMDL_LEVEL_531
Level 531 of system control commands.

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 must 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 must 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 must 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 must 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 only in the environments in which an MQSeries Version 5.1 product, or later, is available.

The value can be:

MQDL_SUPPORTED
Distribution lists supported.

MQDL_NOT_SUPPORTED
Distribution lists not supported.

SyncPoint (MQCFIN)
Syncpoint availability (parameter identifier: MQIA_SYNCPOINT).

The value can 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).

The value can 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).

The value can 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).

The value can 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).

The value can 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).

The value can 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).

The value can 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 only in the environments in which an MQSeries Version 5.1 product, or later, is available.

The value can 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), when a receiver, server-connection, or cluster-sender channel is auto-defined.

This parameter is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

The value can 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).

This exit is invoked when an inbound request for an undefined channel is received, if:

1. The channel is a cluster-sender, or
2. Channel auto-definition is enabled (see *ChannelAutoDef*).

This exit is also invoked when a cluster-receiver channel is started. If a nonblank name is defined, this exit is invoked when an inbound request for an undefined cluster-sender channel is received or channel auto-definition is enabled (see *ChannelAutoDef*),

The format of the name is the same as for the *SecurityExit* parameter described in "Change, Copy and Create Channel" on page 23.

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

This parameter is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

AlterationDate (MQCFST)
Alteration date (parameter identifier: MQCA_ALTERATION_DATE).

The date when the information was last altered.

This parameter is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

AlterationTime (MQCFST)
Alteration time (parameter identifier: MQCA_ALTERATION_TIME).

The time when the information was last altered.

This parameter is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

ClusterWorkLoadExit (MQCFST)
Name of the cluster workload exit (parameter identifier: MQCA_CLUSTER_WORKLOAD_EXIT).

The maximum length of the exit name depends on the environment in which the exit is running. MQ_EXIT_NAME_LENGTH gives the maximum length for the environment in which your application is running. MQ_MAX_EXIT_NAME_LENGTH gives the maximum for all supported environments.

This parameter is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

ClusterWorkLoadData (MQCFST)
Data passed to the cluster workload exit (parameter identifier: MQCA_CLUSTER_WORKLOAD_DATA).

This parameter is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

ClusterWorkLoadLength (MQCFIN)
Cluster workload length (parameter identifier: MQIA_CLUSTER_WORKLOAD_LENGTH).

The maximum length of the message passed to the cluster workload exit.

The value of this attribute must be in the range zero through 999 999 999.

This parameter is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

QMgrIdentifier (MQCFST)
Queue manager identifier (parameter identifier: MQCA_Q_MGR_IDENTIFIER).

The unique identifier of the queue manager.

RepositoryName (MQCFST)
Repository name (parameter identifier: MQCA_REPOSITORY_NAME).

The name of a cluster for which this queue manager is to provide a repository service.

This parameter is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

RepositoryNamelist (MQCFST)
Repository name list (parameter identifier: MQCA_REPOSITORY_NAMELIST).

The name of a list of clusters for which this queue manager is to provide a repository service.

This parameter is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

| *SSLKeyRepository* (MQCFST)
| Location and name of the SSL key repository (parameter identifier:
| MQCA_SSL_KEY_REPOSITORY).

| The length of the string is MQ_SSL_KEY_REPOSITORY_LENGTH.

| Indicates the name of the Secure Sockets Layer key repository.

| This parameter is supported only on AIX, HP-UX, Linux, OS/400, Solaris,
| Windows, and z/OS.

| The format of the name depends on the environment:

- On z/OS, it is the name of a key ring.
- On OS/400, it is of the form *pathname/keyfile*, where *keyfile* is specified without the suffix (.kdb), and identifies a GSKit key database file. The default value is `/QIBM/UserData/ICSS/Cert/Server/Default.`
- On UNIX platforms, it is of the form *pathname/keyfile*, where *keyfile* is specified without the suffix (.kdb), and identifies a GSKit key database file. The default value is `/var/mqm/qmgrs/QMGR/ssl/key`, where `QMGR` is replaced by the queue manager name.
- On Windows systems, the key database is held in a Microsoft Certificate Store file, which has a filename of the form `xxx.sto`, where `xxx` is your chosen name. The SSLKEYR attribute is the path to this file along with the filename stem, (that is, all characters in the filename up to but not including the .sto file extension). WebSphere MQ automatically appends the .sto suffix.

On OS/400, Windows, and UNIX platforms, the syntax of this parameter is validated to ensure that it contains a valid, absolute, directory path.

If SSLKEYR is blank, or is set to a value that does not correspond to a key ring or key database file, channels using SSL fail to start.

Changes to SSLKEYR become effective either:

- On OS/400, Windows, and UNIX platforms, when a new channel process is started.
- For channels that run as threads of the channel initiator on OS/400, Windows, and UNIX platforms, when the channel initiator is restarted.
- For channels that run as threads of the listener on OS/400, Windows, and UNIX platforms, when the listener is restarted.
- On z/OS, when the channel initiator is restarted.

SSLCRLNamelist (MQCFST)
The SSL Certification Revocation List (CRL) namelist (parameter identifier: MQCA_SSL_CRL_NAMELIST).

The length of the string is MQ_NAMELIST_NAME_LENGTH.

Indicates the name of a namelist of authentication information objects to be used for CRL checking by the queue manager.

This parameter is supported only on AIX, HP-UX, Linux, OS/400, Solaris, Windows, and z/OS.

If SSLCRLNamelist is blank, CRL checking is not invoked.

Changes to SSLCRLNamelist, or to the names in a previously specified namelist, or to previously referenced authentication information objects, become effective either:

- On OS/400, Windows, and UNIX platforms, when a new channel process is started.
- For channels that run as threads of the channel initiator on OS/400, Windows, and UNIX platforms, when the channel initiator is restarted.
- For channels that run as threads of the listener on OS/400, Windows, and UNIX platforms, when the listener is restarted.
- On z/OS, when the channel initiator is restarted.

SSLCryptoHardware (MQCFST)
Parameters to configure the SSL cryptographic hardware (parameter identifier: MQCA_SSL_CRYPTO_HARDWARE).

The length of the string is MQ_SSL_CRYPTO_HARDWARE_LENGTH.

Sets the name of the parameter string required to configure the cryptographic hardware present on the system.

This parameter is supported on AIX, HP-UX, Solaris, and Linux only.

The string can have one of the following values:

- GSK_ACCELERATOR_RAINBOW_CS_OFF
- GSK_ACCELERATOR_RAINBOW_CS_ON
- GSK_ACCELERATOR_NCIPHER_NF_OFF
- GSK_ACCELERATOR_NCIPHER_NF_ON
- GSK_PKCS11=*<the PKCS #11 driver path and filename>;<the PKCS #11 token label>;<the PKCS #11 token password>;*

The strings containing RAINBOW enable or disable the Rainbow cryptographic hardware. If the Rainbow cryptographic hardware present, it is enabled by default.

The strings containing NCIPHER enable or disable the nCipher cryptographic hardware. If the nCipher cryptographic hardware is present, it is enabled by default.

To use cryptographic hardware using the PKCS #11 interface, you must specify the string containing PKCS11. The PKCS #11 driver path is an absolute path to the shared library providing support for the PKCS #11 card. The PKCS #11 driver filename is the name of the shared library. An example of the value required for the PKCS #11 driver path and filename is `/usr/lib/pkcs11/PKCS11_API.so`

The maximum length of the string is 256 characters. The default value is blank.

If you specify a string that does not begin with one of the cryptographic strings listed above, you get an error. If you specify the GSK_PKCS11 string, the syntax of the other parameters is also checked.

When the SSLCRYP value is changed, the cryptographic hardware parameters specified become the ones used for new SSL connection environments. The new information becomes effective:

- When a new channel process is started.
- For channels that run as threads of the channel initiator, when the channel initiator is restarted.
- For channels that run as threads of the listener, when the listener is restarted.

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 can be:

MQQT_ALL
All queue types.

MQQT_LOCAL
Local queue.

MQQT_ALIAS
Alias queue definition.

MQQT_REMOTE
Local definition of a remote queue.

MQQT_MODEL
Model queue definition.

The default value if this parameter is not specified is MQQT_ALL.

Error codes

| This command might return the following in the response format header, in
| addition to the values shown on page 18.

Reason (MQLONG)
The value can 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).

Inquire Queue Status

The Inquire Queue Status (MQCMD_INQUIRE_Q_STATUS) command inquires about the status of a local WebSphere MQ queue. You must specify the name of a local queue for which you want to receive status information.

Required parameters:
QName

Optional parameters:
StatusType, *OpenType*, *QStatusAttrs*

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

StatusType (MQCFIN)
Queue status type (parameter identifier: MQIACF_Q_STATUS_TYPE).

Specifies the type of status information required.

The value can be:

MQIACF_Q_STATUS
: Selects status information relating to queues.

MQIACF_Q_HANDLE
: Selects status information relating to the handles that are accessing the queues.

The default value, if this parameter is not specified, is MQIACF_Q_STATUS.

OpenType (MQCFIN)
: Queue status open type (parameter identifier: MQIACF_OPEN_TYPE).

It is always returned, regardless of the channel instance attributes requested.

The value can be:

MQQSOT_ALL
: Selects status for queues that are open with any type of access.

MQQSOT_INPUT
: Selects status for queues that are open for input.

MQQSOT_OUTPUT
: Selects status for queues that are open for output.

The default value if this parameter is not specified is MQQSOT_ALL.

QStatusAttrs (MQCFIL)
: Queue status attributes (parameter identifier: MQIACF_Q_STATUS_ATTRS).

The attribute list can 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_NAME
: Queue name.

MQIA_OPEN_INPUT_COUNT
: The number of handles that are currently open for input for the queue. This does not include handles that are open for browse.

MQIA_OPEN_OUTPUT_COUNT
: The number of handles that are currently open for output for the queue.

MQIA_CURRENT_Q_DEPTH
: The current number of messages on the queue.

MQIACF_UNCOMMITED_MSGS
: Whether there are uncommitted messages on the queue.

MQIACF_PROCESS_ID
: The process identifier of the application that has opened the specified queue.

MQIACF_THREAD_ID
: The thread identifier of the application that has opened the specified queue.

MQCACF_APPL_TAG
: This is a string containing the tag of the application connected to the queue manager.

MQIA_APPL_TYPE
: The type of application that has the queue open.

MQIACF_OPEN_OPTIONS
: The options used to open the queue.

MQCACF_USER_IDENTIFIER
: The username of the application that has opened the specified queue.

MQCACH_CHANNEL_NAME
: The name of the channel that has the queue open, if any.

MQCACH_CONNECTION_NAME
: The connection name of the channel that has the queue open, if any.

Error codes

This command might return the following in the response format header, in addition to the values shown on page 18.

Reason (MQLONG)
The value can be:

MQRC_UNKNOWN_OBJECT
: Unknown object.

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 not valid.

MQRCCF_CFIN_DUPLICATE_PARM
: Duplicate parameter.

MQRCCF_CFIN_LENGTH_ERROR
: Structure length not valid.

MQRCCF_CFIN_PARM_ID_ERROR
: Parameter identifier not valid.

MQRCCF_CFST_DUPLICATE_PARM
: Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
: Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
: Parameter identifier 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.

MQRCCF_OBJECT_NAME_ERROR
Object name not valid.

MQRCCF_STRING_LENGTH_ERROR
String length not valid.

Inquire Queue Status (Response)

The response to the Inquire Queue Status (MQCMD_INQUIRE_Q_STATUS) command consists of the response header followed by the *QName* structure. This response is supported on all platforms

Always returned:
QName

Returned if StatusType is MQIACF_Q_STATUS:
OpenInputCount, *OpenOutputCount*, *CurrentQDepth*, *UncommittedMsgs*

Returned if StatusType is MQIACF_Q_HANDLE:
ProcessId, *ApplTag*, *ThreadId*, *ApplType*, *OpenOptions*, *UserIdentifier*, *ChannelName*, *ConnectionName*, *OpenType*

Response data

QName (MQCFST)
Queue name (parameter identifier: MQCA_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

OpenType (MQCFIN)
Queue status open type (parameter identifier: MQIACF_OPEN_TYPE).

It is always returned, regardless of the queue instance attributes requested.

The value can be:

MQOTCF_ALL
Selects status for queues that are open with any type of access.

MQOTCF_INPUT
Selects status for queues that are open for input.

MQOTCF_OUTPUT
Selects status for queues that are open for output.

The default value if this parameter is not specified is MQOTCF_ALL.

OpenInputCount (MQCFIN)
Open input count (parameter identifier: MQIA_OPEN_INPUT_COUNT).

OpenOutputCount (MQCFIN)
Open output count (parameter identifier: MQIA_OPEN_OUTPUT_COUNT).

CurrentQDepth (MQCFIN)
Current queue depth (parameter identifier: MQIA_CURRENT_Q_DEPTH).

UncommittedMsgs (MQCFIN)
Uncommitted messages (parameter identifier: MQIACF_UNCOMMITTED_MSGS).

The value can be:

MQUMCF_YES
There are uncommitted messages.

MQUMCF_NO
There are no uncommitted messages.

ProcessId (MQCFIN)
Open application process ID (parameter identifier: MQIACF_PROCESS_ID).

ApplTag (MQCFST)
Open application tag (parameter identifier: MQCACF_APPL_TAG).

The maximum length of the string is MQ_APPL_TAG_LENGTH.

ThreadId (MQCFIN)
Open application thread ID (parameter identifier: MQIACF_THREAD_ID).

The maximum length of the string is MQ_PROCESS_NAME_LENGTH.

ApplType (MQCFIN)
Open application type (parameter identifier: MQIA_APPL_TYPE).

The value can be:

MQAT_QMGR
A queue manager process.

MQAT_CHANNEL_INITIATOR
The channel initiator.

MQAT_USER
A user application.

OpenBrowse (MQCFIN)
Open browse (parameter identifier: MQIACF_OPEN_BROWSE).

The value can be:

MQQSO_YES
The queue is open for browsing.

MQQSO_NO
The queue is not open for browsing.

OpenInputType (MQCFIN)
Open input type (parameter identifier: MQIACF_OPEN_INPUT_TYPE).

The value can be:

MQQSO_NO
The queue is not open for inputing.

MQQSO_SHARED
The queue is open for shared input.

MQQSO_EXCLUSIVE
The queue is open for exclusive input.

OpenInquire (MQCFIN)
Open inquire (parameter identifier: MQIACF_OPEN_INQUIRE).

The value can be:

MQQSO_YES
The queue is open for inquiring.

MQQSO_NO
The queue is not open for inquiring.

OpenOutput (MQCFIN)
Open output (parameter identifier: MQIACF_OPEN_OUTPUT).

The value can be:

MQQSO_YES
The queue is open for outputting.

MQQSO_NO
The queue is not open for outputting.

OpenSet (MQCFIN)
Open set (parameter identifier: MQIACF_OPEN_SET).

The value can be:

MQQSO_YES
The queue is open for setting.

MQQSO_NO
The queue is not open for setting.

UserIdentifier (MQCFST)
Open application username (parameter identifier: MQCACF_USER_IDENTIFIER).

The maximum length of the string is MQ_MAX_USER_ID_LENGTH.

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

The maximum length of the string is MQ_CHANNEL_NAME_LENGTH.

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

The maximum length of the string is MQ_CONN_NAME_LENGTH.

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, MQCHT_SERVER, or MQCHT_CLUSSDR.

Where there is both a locally defined channel and an auto-defined cluster-sender channel of the same name, the command applies to the locally defined channel.

If there is no locally defined channel but more than one auto-defined cluster-sender channel, the command applies to the last channel added to the repository on the local queue manager.

The command is not valid if the channel is running; however it is valid if the channel is stopped or in retry mode.

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

| This command might return the following in the response format header, in
| addition to the values shown on page 18.

Reason (MQLONG)
: The value can 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_CLOSED**
| : Channel closed.

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
Connection name not valid.

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

| This command might return the following in the response format header, in
| addition to the values shown on page 18.

Reason (MQLONG)
The value can be:

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

MQRCCF_PARM_COUNT_TOO_SMALL
Parameter count too small.

Refresh Cluster

The Refresh Cluster (MQCMD_REFRESH_CLUSTER) command discards all locally held cluster information, including any auto-defined channels that are not in doubt, and forces the repository to be rebuilt.

This command is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

Required parameters:
ClusterName

| **Optional parameters:**
| *RefreshRepository*

Required parameters

ClusterName (MQCFST)
Cluster name (parameter identifier: MQCA_CLUSTER_NAME).

The name of the cluster to be refreshed.

The maximum length of the string is MQ_CLUSTER_NAME_LENGTH.

This is the name of the cluster to be refreshed. If an asterisk (*) is specified for the name, the queue manager is refreshed in all the clusters to which it belongs.

If an asterisk (*) is specified with *RefreshRepository* set to MQCFO_REFRESH_REPOSITORY_YES, the queue manager restarts its search for repository queue managers, using information in the local cluster-sender channel definitions.

Optional parameters

RefreshRepository (MQCFIN)
Whether repository information should be refreshed (parameter identifier: MQIACF_REFRESH_REPOSITORY).

This indicates whether the information about repository queue managers should be refreshed.

The value can be:

MQCFO_REFRESH_REPOSITORY_YES (REPOS(YES)):
Refresh repository information.

This value cannot be specified if the queue manager is itself a repository queue manager.

REPOS(YES) specifies that in addition to REPOS(NO) behavior, objects representing full repository cluster queue managers are also refreshed. Do not use this option if the queue manager is itself a full repository.

If it is a full repository, you must first alter it so that it is not a full repository for the cluster in question.

The full repository location is recovered from the manually defined CLUSSDR definitions. After the refresh with REPOS(YES) has been issued the queue manager can be altered so that it is once again a full repository.

MQCFO_REFRESH_REPOSITORY_NO (REPOS(NO)):
Do not refresh repository information. This is the default.

If you select REPOS(YES), check that all CLUSSDR channels in the relevant cluster are inactive or stopped before you issue the REFRESH CLUSTER command. If there are CLUSSDR channels running at the time when the REFRESH is processed, and they are used exclusively by the cluster or clusters being refreshed and REPOS(YES) is used, the channels are stopped, by using STOP(channelname) MODE(FORCE) if necessary.

This ensures that the REFRESH can remove the channel state and that the channel will run with the refreshed version after the REFRESH has completed. If a channel's state cannot be deleted, for example because it is in doubt, or because it is also running as part of another cluster, it is state is not new after the refresh and it does not automatically restart if it was stopped.

Error codes

This command might return the following in the response format header, in addition to the values shown on page 18.

Reason (MQLONG)
The value can be:

MQRC_SELECTOR_ERROR
(2067, X'813') Attribute selector not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier 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.

Refresh Security

The Refresh Security (MQCMD_REFRESH_SECURITY) command refreshes the list of authorizations held internally by the authorization service component.

This PCF is supported if you are using the V5.2 or later products only.

Required parameters:
None

Optional parameters:
None

Error codes

| This command might return the following in the response format header, in
| addition to the values shown on page 18.

Reason (MQLONG)
The value can be:

MQRCCF_PARM_COUNT_TOO_BIG
Parameter count too big.

Reset Channel

The Reset Channel (MQCMD_RESET_CHANNEL) command resets the message sequence number for a WebSphere MQ channel with, optionally, a specified 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), server (MQCHT_SERVER), or cluster-sender (MQCHT_CLUSSDR) channel, the value at both ends (issuing end and receiver or requester end), is reset when the channel is next initiated or resynchronized. The value at both ends is reset to be equal.

If the command is issued to a receiver (MQCHT_RECEIVER), requester (MQCHT_REQUESTER), or cluster-receiver (MQCHT_CLUSRCVR) channel, the value at the other end is *not* reset as well; this must be done separately if necessary.

Where there is both a locally defined channel and an auto-defined cluster-sender channel of the same name, the command applies to the locally defined channel.

If there is no locally defined channel but more than one auto-defined cluster-sender channel, the command applies to the last channel added to the repository on the local queue manager.

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 must be in the range 1 through 999 999 999. The default value is one.

Error codes

| This command might return the following in the response format header, in
| addition to the values shown on page 18.

Reason (MQLONG)
The value can 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 Cluster

The Reset Cluster (MQCMD_RESET_CLUSTER) command forces a queue manager to leave a cluster.

This command is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available. If you use any character other than the standard ones listed, you will need to put quotes around those characters.

Required parameters:
ClusterName, *QMgrIdentifier* or *QMgrName*, *Action*

Optional parameters:
RemoveQueues

Required parameters

ClusterName (MQCFST)
Cluster name (parameter identifier: MQCA_CLUSTER_NAME).

The name of the cluster to be reset.

The maximum length of the string is MQ_CLUSTER_NAME_LENGTH.

QMgrIdentifier (MQCFST)
Queue manager identifier (parameter identifier: MQCA_Q_MGR_IDENTIFIER).

This is the unique identifier of the queue manager to be forcibly removed from the cluster. Only one of `QMgrIdentifier` and `QMgrName` can be specified. Use `QMgrIdentifier` in preference to `QmgrName`, because `QmgrName` might not be unique.

QMgrName (MQCFST)
Queue manager name (parameter identifier: MQCA_Q_MGR_NAME).

This is the name of the queue manager to be forcibly removed from the cluster. Only one of `QMgrIdentifier` and `QMgrName` can be specified. Use `QMgrIdentifier` in preference to `QmgrName`, because `QmgrName` might not be unique.

Action (MQCFIN)
Action (parameter identifier: MQIACF_ACTION).

Specifies the action to take place. This can be requested only by a repository queue manager.

The value can be:

MQACT_FORCE_REMOVE
Requests that a queue manager is forcibly removed from a cluster.

Optional parameters

RemoveQueues (MQCFIN)
Whether cluster queues should be removed from the cluster (parameter identifier: MQIACF_REMOVE_QUEUES).

This indicates whether the cluster queues that belong to the queue manager being removed from the cluster should be removed from the cluster. This parameter can be specified even if the queue manager identified by the *QMgrName* parameter is not currently in the cluster.

The value can be:

MQCFO_REMOVE_QUEUES_YES
Remove queues belonging to the queue manager being removed from the cluster.

MQCFO_REMOVE_QUEUES_NO
Do not remove queues belonging to the queue manager being removed. This is the default.

Error codes

This command might return the following in the response format header, in addition to the values shown on page 18.

Reason (MQLONG)
The value can be:

MQRC_SELECTOR_ERROR
(2067, X'813') Attribute selector not valid.

MQRCCF_ACTION_VALUE_ERROR
Value not valid.

MQRCCF_CFIN_DUPLICATE_VALUE
Duplicate parameter.

MQRCCF_CFIN_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFIN_PARM_ID_ERROR
Parameter identifier is not valid.

MQRCCF_CFST_CONFLICTING_PARM
Conflicting parameters.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_PARM_ID_ERROR
Parameter identifier 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.

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.

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

This command might return the following in the response format header, in addition to the values shown on page 18.

Reason (MQLONG)
The value can 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.

This response is supported on all platforms.

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, MQCHT_SERVER, or MQCHT_CLUSSDR.

Where there is both a locally defined channel and an auto-defined cluster-sender channel of the same name, the command applies to the locally defined channel.

If there is no locally defined channel but more than one auto-defined cluster-sender channel, the command applies to the last channel added to the repository on the local queue manager.

This PCF is supported on all platforms.

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.

InDoubt (MQCFIN)
Indoubt resolution (parameter identifier: MQIACH_IN_DOUBT).

Specifies whether to commit or back out the in-doubt messages.

The value can be:

MQIDO_COMMIT
Commit.

MQIDO_BACKOUT
Backout.

Error codes

| This command might return the following in the response format header, in
| addition to the values shown on page 18.

Reason (MQLONG)
The value can 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.

Resume Queue Manager Cluster

The Resume Queue Manager Cluster (MQCMD_RESUME_Q_MGR_CLUSTER) command informs other queue managers in a cluster that the local queue manager is again available for processing, and can be sent messages.

It reverses the action of the Suspend Queue Manager Cluster (MQCMD_SUSPEND_Q_MGR_CLUSTER) command.

This command is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

Required parameters:
ClusterName, or *ClusterNamelist*

Optional parameters:
None

Required parameters

ClusterName (MQCFST)
Cluster name (parameter identifier: MQCA_CLUSTER_NAME).

The name of the cluster for which availability is to be resumed.

The maximum length of the string is MQ_CLUSTER_NAME_LENGTH.

ClusterNamelist (MQCFST)
Cluster Namelist (parameter identifier: MQCA_CLUSTER_NAMELIST).

The name of the namelist specifying a list of clusters for which availability is to be resumed.

Error codes

| This command might return the following in the response format header, in
| addition to the values shown on page 18.

Reason (MQLONG)
The value can be:

MQRC_SELECTOR_ERROR
(2067, X'813') Attribute selector not valid.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

| **MQRCCF_CFST_CONFLICTING_PARM**
| Parameter identifier not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_CLUSTER_NAME_CONFLICT
Cluster name conflict.

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

The Start Channel (MQCMD_START_CHANNEL) command starts a WebSphere MQ channel.

Client connections on MQSeries Version 5, or later, products cannot initiate this command.

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, MQCHT_SVRCONN, or MQCHT_CLUSRCVR, the only action is to enable the channel, not start it.

Where there is both a locally defined channel and an auto-defined cluster-sender channel of the same name, the command applies to the locally defined channel.

If there is no locally defined channel but more than one auto-defined cluster-sender channel, the command applies to the last channel added to the repository on the local queue manager.

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

This command might return the following in the response format header, in addition to the values shown on page 18.

Reason (MQLONG)
The value can 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

| **Note:** This command is not supported on MQSeries for Compaq NonStop Kernel.

The Start Channel Initiator (MQCMD_START_CHANNEL_INIT) command starts a WebSphere MQ channel initiator.

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

| This command might return the following in the response format header, in
| addition to the values shown on page 18.

Reason (MQLONG)
The value can 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

| **Note:** This command is not supported on MQSeries for Compaq NonStop Kernel.

The Start Channel Listener (MQCMD_START_CHANNEL_LISTENER) command starts a WebSphere MQ TCP listener.

This command is valid only for TCP transmission protocols.

Required parameters:
None

Optional parameters:
TransportType

Optional parameters

TransportType (MQCFIN)
Transmission protocol type (parameter identifier: MQIACH_XMIT_PROTOCOL_TYPE).

The value can be:

MQXPT_LU62
LU 6.2.

MQXPT_TCP
TCP.

MQXPT_NETBIOS
NetBIOS.

MQXPT_SPX
SPX.

MQXPT_UDP
UDP.

Error codes

| This command might return the following in the response format header, in
| addition to the values shown on page 18.

Reason (MQLONG)
The value can be:

MQRCCF_COMMS_LIBRARY_ERROR
Communications protocol library error.

MQRCCF_LISTENER_NOT_STARTED
Listener not started.

MQRCCF_NETBIOS_NAME_ERROR
NetBIOS listener name error.

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 a WebSphere MQ channel.

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

Where there is both a locally defined channel and an auto-defined cluster-sender channel of the same name, the command applies to the locally defined channel.

If there is no locally defined channel but more than one auto-defined cluster-sender channel, the command applies to the last channel added to the repository on the local queue manager.

Required parameters:
ChannelName

Optional parameters:
Mode, *ConnectionName*, *QMgrName*, *ChannelStatus*

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

Mode (MQCFIN)
How the channel should be stopped (parameter identifier: MQIACF_MODE).

The value can be:

MQMODE_QUIESCE
Quiesce the channel. This is the default.

MQMODE_FORCE
Stop the channel immediately; the channel's thread or process is not terminated.

MQMODE_TERMINATE
Stop the channel immediately; the channel's thread or process is terminated.

Note: This parameter was previously called *Quiesce* (MQIACF_QUIESCE), with values MQQO_YES and MQQO_NO. The old names can still be used.

ChannelStatus (MQCFIN)
The new state of the channel after the command is executed (parameter identifier: MQIACH_CHANNEL_STATUS).

The value can be:

MQCHS_INACTIVE
Channel is inactive.

MQCHS_STOPPED
Channel is stopped. This is the default if nothing is specified.

ConnectionName (MQCFST)
Connection name of channel to be stopped (parameter identifier: MQCACH_CONNECTION_NAME).

This is the connection name of the channel to be stopped. If this parameter is omitted, all channels with the specified channel name and remote queue manager name are stopped. The maximum length of the string is MQ_CONN_NAME_LENGTH.

If this parameter is specified, ChannelStatus must be MQCHS_INACTIVE.

QMgrName (MQCFST)
Name of remote queue manager (parameter identifier: MQCA_Q_MGR_NAME).

This is the name of the remote queue manager to which the channel is connected. If this parameter is omitted, all channels with the specified channel name and connection name are stopped. The maximum length of the string is MQ_Q_MGR_NAME_LENGTH.

If this parameter is specified, ChannelStatus must be MQCHS_INACTIVE.

Error codes

This command might return the following in the response format header, in addition to the values shown on page 18.

Reason (MQLONG)
The value can 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_CONFLICTING_PARM
Conflicting parameter or if you have specified both STATUS (STOPPED) and either CONNAME or QMNAME parameters.

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_MODE_VALUE_ERROR
Mode value not valid.

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_STRUCTURE_TYPE_ERROR
Structure type not valid.

Suspend Queue Manager Cluster

The Suspend Queue Manager Cluster (MQCMD_SUSPEND_Q_MGR_CLUSTER) command informs other queue managers in a cluster that the local queue manager is not available for processing, and cannot be sent messages.

Its action can be reversed by the Resume Queue Manager Cluster (MQCMD_RESUME_Q_MGR_CLUSTER) command.

This command is supported only in the environments in which an MQSeries Version 5.1 product, or later, is available.

Required parameters:
ClusterName or *ClusterNamelist*

Optional parameters:
Mode

Required parameters

ClusterName (MQCFST)
Cluster name (parameter identifier: MQCA_CLUSTER_NAME).

The name of the cluster for which availability is to be suspended.

The maximum length of the string is MQ_CLUSTER_NAME_LENGTH.

ClusterNamelist (MQCFST)
Cluster Namelist (parameter identifier: MQCA_CLUSTER_NAMELIST).

The name of the namelist specifying a list of clusters for which availability is to be suspended.

Optional parameters

| *Mode* (MQCFIN)
| How the local queue manager should be suspended from the cluster
| (parameter identifier: MQIACF_MODE).

| The value can be:

| **MQMODE_QUIESCE**
| Other queue managers in the cluster are advised that the local queue
| manager should not be sent further messages.

| **MQMODE_FORCE**
| All inbound and outbound channels to other queue managers in the
| cluster are stopped forcibly.

Note: This parameter was previously called *Quiesce* (MQIACF_QUIESCE), with values MQQO_YES and MQQO_NO. The old names can still be used.

Error codes

This command might return the following in the response format header, in addition to the values shown on page 18.

Reason (MQLONG)
The value can be:

MQRC_SELECTOR_ERROR
(2067, X'813') Attribute selector not valid.

MQRCCF_CFST_DUPLICATE_PARM
Duplicate parameter.

MQRCCF_CFST_LENGTH_ERROR
Structure length not valid.

MQRCCF_CFST_CONFLICTING_PARM
Parameter identifier not valid.

MQRCCF_CFST_STRING_LENGTH_ERR
String length not valid.

MQRCCF_CLUSTER_NAME_CONFLICT
Cluster name conflict.

MQRCCF_MODE_VALUE_ERROR
Mode 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.

Chapter 5. Structures used for commands and responses

Commands and responses have the form:
- PCF header (MQCFH) structure (described on page 193), followed by
- Zero or more parameter structures. Each of these is one of the following:
  - PCF integer parameter (MQCFIN, page 201)
  - PCF string parameter (MQCFST, page 205)
  - PCF integer list parameter (MQCFIL, page 209)
  - PCF string list parameter (MQCFSL, page 213)

This chapter defines these parameter structures, and includes:
- “How the structures are shown”
- “Usage notes” on page 192
- Chapter 6, “MQCFH - PCF header”, on page 193
- Chapter 7, “MQCFIN - PCF integer parameter”, on page 201
- Chapter 8, “MQCFST - PCF string parameter”, on page 205
- Chapter 9, “MQCFIL - PCF integer list parameter”, on page 209
- Chapter 10, “MQCFSL - PCF string list parameter”, on page 213

See also:
- Chapter 6, “MQCFH - PCF header”, on page 193
- Chapter 7, “MQCFIN - PCF integer parameter”, on page 201
- Chapter 8, “MQCFST - PCF string parameter”, on page 205
- Chapter 9, “MQCFIL - PCF integer list parameter”, on page 209
- Chapter 10, “MQCFSL - PCF string list parameter”, on page 213

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
- Visual Basic

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 *WebSphere MQ Application Programming Reference* manual.

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
Assembler	CMQCFA CMQCFINA CMQCFILA CMQCFSTA CMQCFSLA CMQCFHA

Structures

COBOL	CMQCFV CMQCFHL CMQCFHV CMQCFINL CMQCFINV CMQCFSLL CMQCFSLV CMQCFSTL CMQCFSTV CMQCFILL CMQCFILV
PL/I	CMQCFP
Visual Basic	CMQB CMQFB CMQXB

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.

This enables conversions of the strings within the message, to the *CodedCharSetId* value in the MQMD specified on the MQGET call, if the MQGMO_CONVERT option is also specified.

Note: If you request conversion of the internal strings in the message, the conversion will occur only if the value of the *CodedCharSetId* field in the MQMD of the message is different from the *CodedCharSetId* field of the MQMD specified on the MQGET call.

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.

Chapter 6. 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 9). 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.

Fields

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.

MQCFT_USER
User-defined PCF message.

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_INQUIRE_Q_STATUS**
| Inquire queue status.

MQCMD_REFRESH_Q_MGR
Refresh queue manager.

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_CHANGE_NAMELIST
: Change namelist.

MQCMD_COPY_NAMELIST
: Copy namelist.

MQCMD_CREATE_NAMELIST
: Create namelist.

MQCMD_DELETE_NAMELIST
: Delete namelist.

MQCMD_INQUIRE_NAMELIST
: Inquire namelist.

MQCMD_INQUIRE_NAMELIST_NAMES
: Inquire namelist names.

MQCMD_ESCAPE
: Escape.

MQCMD_RESOLVE_CHANNEL
: Resolve channel.

MQCMD_PING_Q_MGR
: Ping queue manager.

MQCMD_INQUIRE_CHANNEL_STATUS
: Inquire channel status.

MQCMD_CONFIG_EVENT
: Configuration event.

MQCMD_Q_MGR_EVENT
Queue manager event.

MQCMD_PERFM_EVENT
Performance event.

MQCMD_CHANNEL_EVENT
Channel event.

MQCMD_INQUIRE_CLUSTER_Q_MGR
Inquire cluster queue manager.

MQCMD_RESUME_Q_MGR_CLUSTER
Resume cluster queue manager.

MQCMD_SUSPEND_Q_MGR_CLUSTER
Suspend cluster queue manager.

MQCMD_REFRESH_CLUSTER
Refresh cluster.

MQCMD_RESET_CLUSTER
Reset cluster.

MQCMD_REFRESH_SECURITY
Refresh security.

The initial value of this field is the following special value:

MQCMD_NONE
No command.

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 `one`.

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 in Chapter 3, “Definitions of the Programmable Command Formats”, on page 17, and in the description of each command. The reason codes are listed in alphabetic order, with complete descriptions in Appendix A, “Error codes”, on page 341.

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 `zero`.

Table 5. 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	MQCMD_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};
```

Language declarations

This structure is available in the following languages:

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 (z/OS, OS/2 and Windows)

```
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 (z/OS 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)
```

Visual Basic language declaration (Windows only)

```
Type MQCFH
  Type As Long            'Structure type
  StrucLength As Long     'Structure length
  Version As Long         'Structure version number
  Command As Long         'Command identifier
  MsgSeqNumber As Long    'Message sequence number
  Control As Long         'Control options
  CompCode As Long        'Completion code
  Reason As Long          'Reason code qualifying completion code
  ParameterCount As Long  'Count of parameter structures
End Type

Global MQCFH_DEFAULT As MQCFH
```

RPG language declaration (iSeries only)

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQCFH Structure
D*
D* Structure type
D  FHTYP                  1      4I 0 INZ(1)
D* Structure length
D  FHLEN                  5      8I 0 INZ(36)
D* Structure version number
D  FHVER                  9     12I 0 INZ(1)
D* Command identifier
D  FHCMD                 13     16I 0 INZ(0)
D* Message sequence number
D  FHSEQ                 17     20I 0 INZ(1)
D* Control options
D  FHCTL                 21     24I 0 INZ(1)
D* Completion code
D  FHCMP                 25     28I 0 INZ(0)
D* Reason code qualifying completion code
D  FHREA                 29     32I 0 INZ(0)
D* Count of parameter structures
D  FHCNT                 33     36I 0 INZ(0)
D*
```

Chapter 7. 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 9). 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.

Fields

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 Chapter 6, “MQCFH - PCF header”, on page 193 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 6. 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};
```

Language declarations

This structure is available in the following languages:

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 (OS/2, z/OS, and Windows)

```
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 (z/OS 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)
```

Visual Basic language declaration (Windows only)

```
Type MQCFIN
  Type As Long         ' Structure type
  StrucLength As Long  ' Structure length
  Parameter As Long    ' Parameter identifier
  Value As Long        ' Parameter value
End Type

Global MQCFIN_DEFAULT As MQCFIN
```

RPG language declaration (iSeries only)

```
D* MQCFIN Structure
D*
D* Structure type
D  INTYP                 1      4I 0 INZ(3)
D* Structure length
D  INLEN                 5      8I 0 INZ(16)
D* Parameter identifier
D  INPRM                 9     12I 0 INZ(0)
D* Parameter value
D  INVAL                13     16I 0 INZ(0)
D*
```

Chapter 8. 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 9). 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 192 for further information on how to use the structure.

Fields

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 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 Chapter 6, "MQCFH - PCF header", on page 193 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 character set identifier.

The string data is in the character set defined by the *`CodedCharSetId`* field in the MQ header structure that *precedes* the MQCFH structure, or by the *`CodedCharSetId`* field in the MQMD if the MQCFH structure is at the start of the message.

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 might 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 must 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 must 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 7. 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};
```

Language declarations

This structure is available in the following languages:

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 (OS/2, z/OS, and Windows)

```
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 (z/OS 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)
```

Visual Basic language declaration (Windows only)

```
Type MQCFST
  Type As Long            ' Structure type
  StrucLength As Long     ' Structure length
  Parameter As Long       ' Parameter identifier
  CodedCharSetId As Long  ' Coded character set identifier
  StringLength As Long    ' Length of string
End Type

Global MQCFST_DEFAULT As MQCFST
```

RPG language declaration (iSeries only)

```
D* MQCFST Structure
D*
D* Structure type
D  STTYP                 1      4I 0 INZ(4)
D* Structure length
D  STLEN                 5      8I 0 INZ(20)
D* Parameter identifier
D  STPRM                 9     12I 0 INZ(0)
D* Coded character set identifier
D  STCSI                13     16I 0 INZ(0)
D* Length of string
D  STSTL                17     20I 0 INZ(0)
D*
```

Chapter 9. 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 9). 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.

Fields

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 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 Chapter 6, "MQCFH - PCF header", on page 193 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 must be allocated dynamically, and pointers used to address the fields within it.
- For the COBOL, PL/I, RPG, and System/390 assembler programming languages, the field is omitted from the structure declaration. When an instance of the structure is declared, you must include MQCFIN in a larger structure, and declare additional fields following MQCFIN, to represent the *Values* field as required.

In C, the initial value of this field is a single `0`.

Table 8. 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};
```

Language declarations

This structure is available in the following languages:

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 (OS/2, z/OS, and Windows)

```
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 (z/OS 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)
```

Visual Basic language declaration (Windows only)

```
Type MQCFIL
  Type As Long         ' Structure type
  StrucLength As Long  ' Structure length
  Parameter As Long    ' Parameter identifier
  Count As Long        ' Count of parameter values
End Type

Global MQCFIL_DEFAULT As MQCFIL
```

RPG language declaration (iSeries only)

```
D* MQCFIL Structure
D*
D* Structure type
D  ILTYP                 1      4I 0 INZ(5)
D* Structure length
D  ILLEN                 5      8I 0 INZ(16)
D* Parameter identifier
D  ILPRM                 9     12I 0 INZ(0)
D* Count of parameter values
D  ILCNT                13     16I 0 INZ(0)
D*
```

Chapter 10. 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 9). 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 192 for further information on how to use the structure.

Fields

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 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 Chapter 6, "MQCFH - PCF header", on page 193 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 character set identifier.

The string data is in the character set defined by the *CodedCharSetId* field in the MQ header structure that *precedes* the MQCFH structure, or by the *CodedCharSetId* field in the MQMD if the MQCFH structure is at the start of the message.

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 might 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 must be allocated dynamically, and pointers used to address the fields within it.
- For the COBOL, PL/I, RPG, and System/390 assembler programming languages, the field is omitted from the structure declaration. When an instance of the structure is declared, you must include MQCFSL in a larger structure, and declare additional fields following MQCFSL, to represent the *Strings* field as required.

In C, the initial value of this field is the null string.

Table 9. 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};
```

Language declarations

The declarations available for this structure are:

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 (OS/2, z/OS and Windows)

```
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 (z/OS 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)
```

Visual Basic language declaration (Windows only)

```
Type MQCFSL
  Type As Long            ' Structure type
  StrucLength As Long     ' Structure length
  Parameter As Long       ' Parameter identifier
  CodedCharSetId As Long  ' Coded character set identifier
  Count As Long           ' Count of parameter values
  StringLength As Long    ' Length of one string
End Type

Global MQCFSL_DEFAULT As MQCFSL
```

RPG language declaration (iSeries only)

```
D* MQCFSL Structure
D*
D* Structure type
D  SLTYP                 1      4I 0 INZ(6)
D* Structure length
D  SLLEN                 5      8I 0 INZ(24
D* Parameter identifier
D  SLPRM                 9     12I 0 INZ(0)
D* Coded character set identifier
D  SLCSI                13     16I 0 INZ(0)
D* Count of parameter values
D  SLCNT                17     20I 0 INZ(0)
D* Length of one string
D  SLSTL                21     24I 0 INZ(0)
```

Chapter 11. MQCFBS — PCF byte string parameter

The MQCFBS structure describes a byte-string parameter in a PCF message. This can occur in the following types of message:

- Command message (MQCFT_COMMAND); the format name is MQFMT_ADMIN.
- Response message (MQCFT_RESPONSE); the format name is MQFMT_ADMIN.
- Event message (MQCFT_EVENT); the format name is MQFMT_PCF.

When an MQCFBS structure is present, the *Version* field in the MQCFH structure at the start of the PCF must be MQCFH_VERSION_2.

In a user PCF message, the *Parameter* field has no significance, and can be used by the application for its own purposes.

The structure ends with a variable-length byte string; see the *String* field below for further details.

Fields

Type (MQLONG)
Structure type.

This indicates that the structure is an MQCFBS structure describing byte string parameter. The value must be:

MQCFT_BYTE_STRING
Structure defining a byte string.

The initial value of this field is MQCFT_BYTE_STRING.

StrucLength (MQLONG)
Structure length.

This is the length in bytes of the MQCFBS 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:

MQCFBS_STRUC_LENGTH_FIXED
Length of fixed part of MQCFBS structure.

The initial value of this field is MQCFBS_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 Chapter 6, "MQCFH - PCF header", on page 193 for details. In user PCF messages (MQCFT_USER), this field has no significance.

The initial value of this field is 0.

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 (MQBYTE×*StringLength*)
String value.

This is the value of the parameter identified by the *parameter* field. The string is a byte string, and so is not subject to character-set conversion when sent between different systems.

Note: A null character in the string is treated as normal data, and does not act as a delimiter for the string

For MQFMT_ADMIN messages, if the specified string is shorter than the standard length of the *parameter,* the omitted characters are assumed to be nulls. If the specified string is longer than the standard length, those characters in excess of the standard length must be nulls.

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 must be allocated dynamically, and pointers used to address the fields within it.
- For other programming languages, the field is omitted from the structure declaration. When an instance of the structure is declared, you must include MQCFBS in a larger structure, and declare additional fields following MQCFBS, to represent the *String* field as required.

In C, the initial value of this field is the null string.

Chapter 12. Example of using PCFs

This is an example of how Programmable Command Formats can be used in a program for administration of WebSphere MQ queues.

Enquire local queue attributes

A C language program is listed here that uses WebSphere MQ for Windows, V5.3. 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 WebSphere MQ 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

```
/*===========================================================================*/
/*                                                                           */
/* 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 immediately 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 all 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->MsgDeliverySequence = 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    ParmBuffer[120];   /* 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 nonblank 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 */
}
```

PCF example

Part 2. Message Queuing Administration Interface

Chapter 13. Introduction to the WebSphere MQ Administration Interface (MQAI)

This chapter describes:
- The main WebSphere MQ Administration Interface (MQAI) concepts and terminology
- When the MQAI can be used
- How to use the MQAI

MQAI concepts and terminology

The MQAI is a programming interface to WebSphere MQ, using the C language and also Visual Basic for Windows. It performs administration tasks on a WebSphere MQ queue manager using *data bags*. Data bags allow you to handle properties (or parameters) of objects in a way that is easier than using the other administration interface, Programmable Command Formats (PCFs). The MQAI offers easier manipulation of PCFs than using the MQGET and MQPUT calls. For more information about data bags, see Chapter 14, "Using data bags", on page 239. For more information about PCFs, see part 1 of this book.

The data bag contains zero or more *data items*. These are ordered within the bag as they are placed into the bag. This is called the *insertion order*. Each data item contains a *selector* that identifies the data item and a *value* of that data item that can be either an integer, a string, or a handle of another bag.

There are two types of selector; *user selectors* and *system selectors*. These are described in Appendix G, "MQAI Selectors", on page 389. The selectors are usually unique, but it is possible to have multiple values for the same selector. In this case, an *index* identifies the particular occurrence of selector that is required. Indexes are described in "Indexing" on page 335.

A hierarchy of the above concepts is shown in Figure 1.

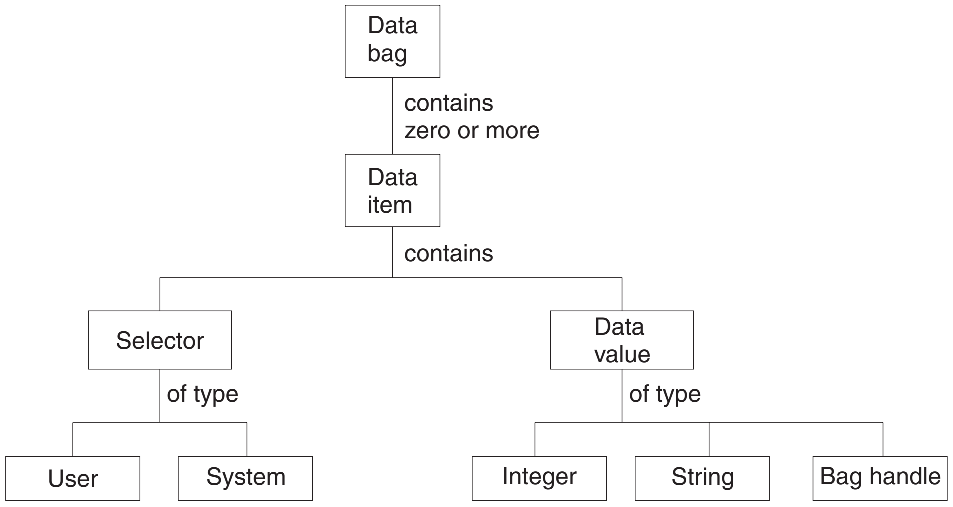


Figure 1. Hierarchy of MQAI concepts

Use of the MQAI

You can use the MQAI to:.

- Implement self-administering applications and administration tools. For example, the Active Directory Services provided on Windows uses the MQAI. For more information about the Active Directory Service Interface, see the *WebSphere MQ for Windows, V5.3 Using the Component Object Model Interface* book.
- Simplify the use of PCF messages. The MQAI is an easy way to administer WebSphere MQ; you do not have to write your own PCF messages and thus avoid the problems associated with complex data structures.
- Handle error conditions more easily. It is difficult to get return codes back from the WebSphere MQ script (MQSC) commands, but the MQAI makes it easier for the program to handle error conditions.

How do I use the MQAI?

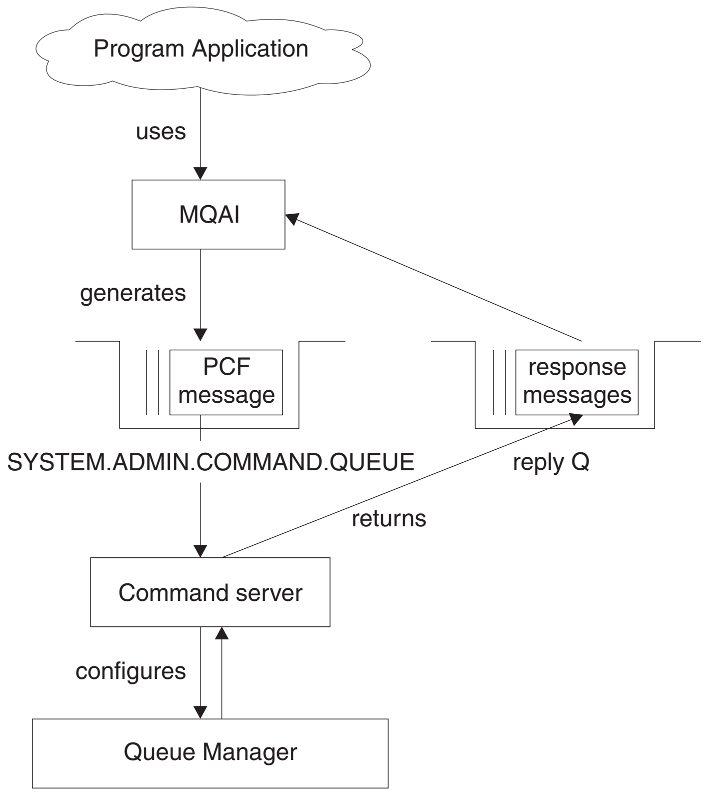


Figure 2. How the MQAI administers WebSphere MQ

The MQAI provides easier programming access to PCF messages. To pass parameters in programs that are written using MQI calls, the PCF message must contain the command and details of the string or integer data. To do this, several statements are needed in your program for every structure, and memory space must be allocated.

On the other hand, programs written using the MQAI pass parameters into the data bag and only one statement is required for each structure. The data bag removes the need for the programmer to handle arrays and allocate storage, and provides some isolation from the details of PCF.

The MQAI administers WebSphere MQ by sending PCF messages to the command server and waiting for a response as shown in Figure 2 on page 236.

Overview

The following instructions give a brief overview of 1) what you do with the MQAI, and 2) how you use the MQAI. Further details are contained in the rest of this book.

To use the MQAI to administer WebSphere MQ:

1. Decide on the task you want to carry out (for example, Change Queue).
2. Use part 1 of this book as a reference to the commands and responses sent between a WebSphere MQ systems management application program and a WebSphere MQ queue manager. For example, look up the Change, Create and Copy Queues command in this book.
3. Choose the values of the selectors for the required parameters and any optional parameters that you want to set.
4. Create a data bag using the mqCreateBag call and enter values for each of these selectors using the mqAddInteger, mqAddString, and mqAddInquiry calls. This is described in Chapter 14, “Using data bags”, on page 239.
5. Ensure the command server is running.
6. Using the mqExecute call, send the message to the command server and wait for a response. This is described in Chapter 15, “Configuring WebSphere MQ using mqExecute”, on page 247.

To use the MQAI to exchange data between applications:

- The sender must:
  1. Create a data bag intended to send the data using mqCreateBag. See “Creating and deleting data bags” on page 239.
  2. Add the data to be sent in the bag using mqAddInteger or mqAddString. See “Adding data items to bags” on page 240.
  3. Use the mqPutBag call to convert the data in the bag into a PCF message and put the message onto the required queue. See “Putting and receiving data bags” on page 252.
- The receiver must:
  1. Create a data bag intended to receive the data using mqCreateBag. See “Creating and deleting data bags” on page 239.
  2. Use the mqGetBag call to get the PCF message from the queue and recreate a bag from the PCF message. See “Putting and receiving data bags” on page 252.

Using the MQAI is discussed in more detail in the chapters that follow.

Building your MQAI application

To build your application using the MQAI, you link to the same libraries as you do for WebSphere MQ. For information on how to build your WebSphere MQ applications, see the *WebSphere MQ Application Programming Guide*.

Chapter 14. Using data bags

A data bag is a means of handling properties (or parameters) of objects using the MQAI. This chapter discusses the configuration of data bags. It describes:
- The different types of bag and their uses
- How to create and delete data bags
- Types of data item
- How to add data items to data bags
- How to change information within a data bag
- How to count data items within a data bag
- How to delete data items
- How to inquire within data bags
- System items

Types of data bag

You can choose the type of data bag that you want to create depending on the task that you wish to perform:

user bag
A simple bag used for user data.

administration bag
A bag created for data used to administer WebSphere MQ objects by sending administration messages to a command server. The administration bag automatically implies certain options as described in "Creating and deleting data bags".

command bag
A bag also created for commands for administering WebSphere MQ objects. However, unlike the administration bag, the command bag does not automatically imply certain options although these options are available. Again, these options are discussed in "Creating and deleting data bags".

In addition, the **system bag** is created by the MQAI when a reply message is returned from the command server and placed into a user's output bag. A system bag cannot be modified by the user.

Creating and deleting data bags

To use the MQAI, you first create a data bag using the mqCreateBag call. As input to this call, you supply one or more options to control the creation of the bag.

The `Options` parameter of the MQCreateBag call lets you choose whether to create a user bag, a command bag, or an administration bag.

To create a user bag or a command bag, you can choose one or more further options to:
- Use the list form when there are two or more adjacent occurrences of the same selector in a bag.
- Reorder the data items as they are added to a PCF message to ensure that the parameters are in their correct order.

- Check the values of user selectors for items that you add to the bag.

Administration bags automatically imply these options.

A data bag is identified by its handle. The bag handle is returned from mqCreateBag and must be supplied on all other calls that use the data bag.

For a full description of the mqCreateBag call, see "mqCreateBag" on page 270.

Deleting data bags

Any data bag that is created by the user must also be deleted using the mqDeleteBag call. For example, if a bag is created in the user code, it must also be deleted in the user code.

System bags are created and deleted automatically by the MQAI. For more information about this, see "Sending administration commands to the command server" on page 247. User code cannot delete a system bag.

For a full description of the mqDeleteBag call, see "mqDeleteBag" on page 274.

Types of data item

Here are the types of data item available within the MQAI:
- Integer
- Character-string
- Bag handle

When you have created a data bag, you can populate it with integer or character-string items. You can inquire about all three types of item.

Note: You cannot insert bag handles.

These data items can be user or system items. User items contain user data such as attributes of objects that are being administered. System items should be used for more control over the messages generated: for example, the generation of message headers. For more information about system items, see "System items" on page 244.

Adding data items to bags

The MQAI lets you add integer items and character-string items to bags and this is shown in Figure 3. The items are identified by a selector. Usually one selector identifies one item only, but this is not always the case. If a data item with the specified selector is already present in the bag, an additional instance of that selector is added to the end of the bag.

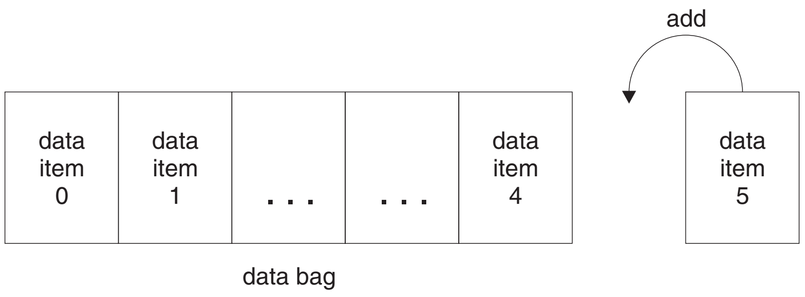


Figure 3. Adding data items

Add data items to a bag using the mqAdd* calls. To add integer items, use the mqAddInteger call as described in "mqAddInteger" on page 258. To add character-string items, use the mqAddString call as described in "mqAddString" on page 260.

Adding an inquiry command to a bag

The mqAddInquiry call is used to add an inquiry command to a bag. The call is specifically for administration purposes, so it can be used with administration bags only. It lets you specify the selectors of attributes on which you want to inquire from WebSphere MQ.

For a full description of the mqAddInquiry call, see "mqAddInquiry" on page 256.

Filtering and querying data items

When using the MQAI to inquire about the attributes of WebSphere MQ objects, you can control the data that is returned to your program in two ways.

1. You can ***filter*** the data that is returned using the mqAddInteger and mqAddString calls. This approach lets you specify a *Selector* and *ItemValue* pair, for example:

   ```
   mqAddInteger(inputbag, MQIA_Q_TYPE, MQQT_LOCAL)
   ```

   This example specifies that the queue type (*Selector*) must be local (*ItemValue*) and this specification must match the attributes of the object (in this case, a queue) about which you are inquiring.

   Other attributes that can be filtered correspond to the PCF Inquire* commands that can be found in part 1 of this book. For example, to inquire about the attributes of a channel, see the Inquire Channel command in this book. The "Required parameters" and "Optional parameters" of the Inquire Channel command identify the selectors that you can use for filtering.
2. You can ***query*** particular attributes of an object using the mqAddInquiry call. This specifies the selector in which you are interested. If you do not specify the selector, all attributes of the object are returned.

Here is an example of filtering and querying the attributes of a queue:

```
/* Request information about all queues */
mqAddString(adminbag, MQCA_Q_NAME, "*")

/* Filter attributes so that local queues only are returned */
mqAddInteger(adminbag, MQIA_Q_TYPE, MQQT_LOCAL)

/* Query the names and current depths of the local queues */
mqAddInquiry(adminbag, MQCA_Q_NAME)
```

Adding data items

```
mqAddInquiry(adminbag, MQIA_CURRENT_Q_DEPTH)

/* Send inquiry to the command server and wait for reply */
mqExecute(MQCMD_INQUIRE_Q, ...)
```

For more examples of filtering and querying data items, see Chapter 18, "Examples of using the MQAI", on page 315.

Changing information within a bag

The MQAI lets you change information within a bag using the mqSet* calls. You can:

1. Modify data items within a bag. The index allows an individual instance of a parameter to be replaced by identifying the occurrence of the item to be modified (see Figure 4).

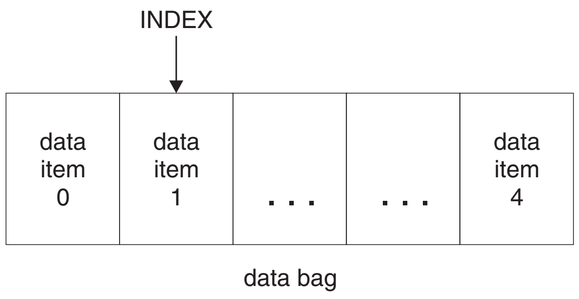


Figure 4. Modifying a single data item

2. Delete all existing occurrences of the specified selector and add a new occurrence to the end of the bag. (See Figure 5.) A special index value allows ***all*** instances of a parameter to be replaced.

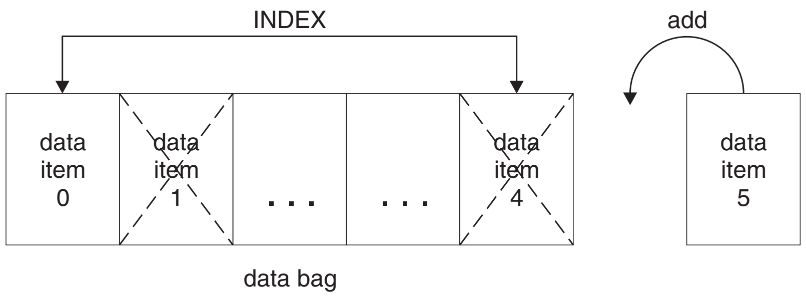


Figure 5. Modifying all data items

Note: The index preserves the insertion order within the bag but can affect the indices of other data items.

The mqSetInteger call lets you modify integer items within a bag and the mqSetString call lets you modify character-string items. Alternatively, you can use these calls to delete all existing occurrences of the specified selector and add a new occurrence at the end of the bag. The data item can be a user item or a system item.

For a full description of these calls, see "mqSetInteger" on page 304 and "mqSetString" on page 307.

Counting data items

The mqCountItems call counts the number of user items, system items, or both, that are stored in a data bag, and returns this number. For example, `mqCountItems(Bag, 7, ...)`, returns the number of items in the bag with a selector of 7. It can count items by individual selector, by user selectors, by system selectors, or by all selectors.

Note: This call counts the number of data items, not the number of unique selectors in the bag. A selector can occur multiple times, so there may be fewer unique selectors in the bag than data items.

For a full description of the mqCountItems call, see "mqCountItems" on page 268.

Deleting data items

You can delete items from bags in a number of ways. You can:
- Remove one or more user items from a bag,
- Delete ***all*** user items from a bag, that is, *clear* a bag,
- Delete user items from the end of a bag, that is, *truncate* a bag.

Deleting data items from a bag using the mqDeleteItem call

The mqDeleteItem call removes one or more user items from a bag. The index is used to delete either:

1. A single occurrence of the specified selector. (See Figure 6.)

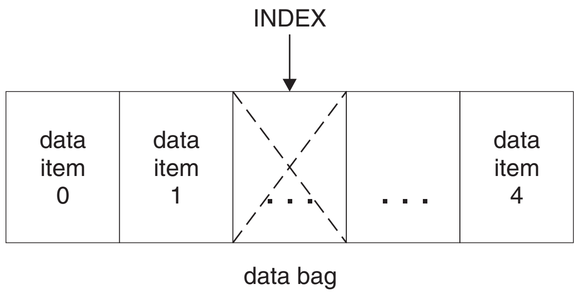


Figure 6. Deleting a single data item

or

2. All occurrences of the specified selector. (See Figure 7.)

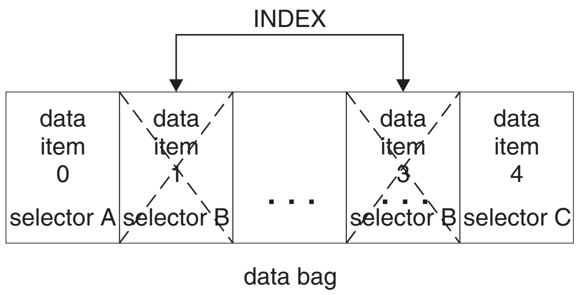


Figure 7. Deleting all data items

Note: The index preserves the insertion order within the bag but can affect the indices of other data items. For example, the mqDeleteItem call does not preserve the index values of the data items that follow the deleted item because the indices are reorganized to fill the gap that remains from the deleted item.

For a full description of the mqDeleteItem call, see “mqDeleteItem” on page 276.

Clearing a bag using the mqClearBag call

The mqClearBag call removes all user items from a user bag and resets system items to their initial values. System bags contained within the bag are also deleted.

For a full description of the mqClearBag call, see “mqClearBag” on page 267.

Truncating a bag using the mqTruncateBag call

The mqTruncateBag call reduces the number of user items in a user bag by deleting the items from the end of the bag, starting with the most recently added item. For example, it can be used when using the same header information to generate more than one message.

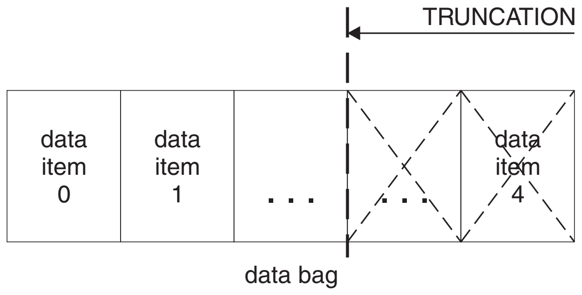


Figure 8. Truncating a bag

For a full description of the mqTruncateBag call, see “mqTruncateBag” on page 312.

Inquiring within data bags

You can inquire about:

- The value of an integer item using the mqInquireInteger call. See “mqInquireInteger” on page 289.
- The value of a character-string item using the mqInquireString call. See “mqInquireString” on page 295.
- The value of a bag handle using the mqInquireBag call. See “mqInquireBag” on page 286.

You can also inquire about the type (integer, character string, or bag handle) of a specific item using the mqInquireItemInfo call. See “mqInquireItemInfo” on page 292.

System items

System items can be used for:

- The generation of PCF headers. System items can control the PCF command identifier, control options, message sequence number, and command type.
- Data conversion. System items handle the character-set identifier for the character-string items in the bag.

Like all data items, system items consist of a selector and a value. For information about these selectors and what they are for, see Appendix G, “MQAI Selectors”, on page 389.

System items are unique. One or more system items can be identified by a system selector. There is only one occurrence of each system selector.

Most system items can be modified (see “Changing information within a bag” on page 242), but the bag-creation options cannot be changed by the user. You cannot delete system items. (See “Deleting data items” on page 243.)

Chapter 15. Configuring WebSphere MQ using mqExecute

After you have created and populated your data bag, you can send an administration command message to the command server of a queue manager and wait for any response messages. The easiest way to do this is by using the mqExecute call. This handles the exchange with the command server and returns responses in a bag.

Sending administration commands to the command server

The mqExecute call sends an administration command message as a nonpersistent message and waits for any responses. Responses are returned in a response bag. These might contain information about attributes relating to several WebSphere MQ objects or a series of PCF error response messages, for example. Therefore, the response bag could contain a return code only or it could contain *nested bags*.

Response messages are placed into system bags that are created by the system. For example, for inquiries about the names of objects, a system bag is created to hold those object names and the bag is inserted into the user bag. Handles to these bags are then inserted into the response bag and the nested bag can be accessed by the selector MQHA_BAG_HANDLE. The system bag stays in storage, if it is not deleted, until the response bag is deleted.

The concept of *nesting* is shown in Figure 9.

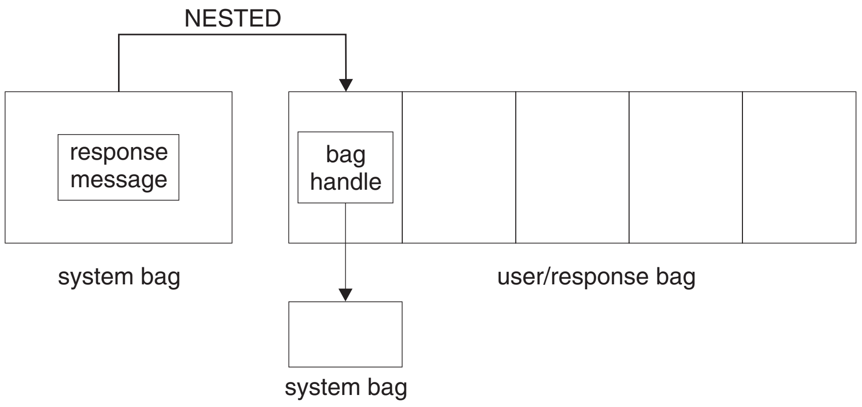


Figure 9. Nesting

As input to the mqExecute call, you must supply:

- An MQI connection handle.
- The command to be executed. This should be one of the MQCMD_* values.

  Note: If this value is not recognized by the MQAI, the value is still accepted. However, if the mqAddInquiry call was used to insert values into the bag, this parameter must be an INQUIRE command recognized by the MQAI. That is, the parameter should be of the form MQCMD_INQUIRE_*.
- Optionally, a handle of the bag containing options that control the processing of the call. This is also where you can specify the maximum time in milliseconds that the MQAI should wait for each reply message.

- A handle of the administration bag that contains details of the administration command to be issued.
- A handle of the response bag that receives the reply messages.

The following are optional:
- An object handle of the queue where the administration command is to be placed.

  If no object handle is specified, the administration command is placed on the SYSTEM.ADMIN.COMMAND.QUEUE belonging to the currently connected queue manager. This is the default.
- An object handle of the queue where reply messages are to be placed.

  You can choose to place the reply messages on a dynamic queue that is created automatically by the MQAI. The queue created exists for the duration of the call only, and is deleted by the MQAI on exit from the mqExecute call.

Example code

Here are some example uses of the mqExecute call.

The example shown in figure 10 creates a local queue (with a maximum message length of 100 bytes) on a queue manager:

```
/* Create a bag for the data you want in your PCF message */
mqCreateBag(MQCBO_ADMIN_BAG, &hbagRequest)

/* Create a bag to be filled with the response from the command server */
mqCreateBag(MQCBO_ADMIN_BAG, &hbagResponse)

/* Create a queue    */
/* Supply queue name */
mqAddString(hbagRequest, MQCA_Q_NAME, "QBERT")

/* Supply queue type */
mqAddString(hbagRequest, MQIA_Q_TYPE, MQQT_LOCAL)

/* Maximum message length is an optional parameter */
mqAddString(hbagRequest, MQIA_MAX_MSG_LENGTH, 100)

/* Ask the command server to create the queue */
mqExecute(MQCMD_CREATE_Q, hbagRequest, hbagResponse)

/* Tidy up memory allocated */
mqDeleteBag(hbagRequest)
mqDeleteBag(hbagResponse)
```

Figure 10. Using mqExecute to create a local queue

The example shown in figure 11 inquires about all attributes of a particular queue. The mqAddInquiry call identifies all WebSphere MQ object attributes of a queue to be returned by the Inquire parameter on mqExecute.

```
/* Create a bag for the data you want in your PCF message */
mqCreateBag(MQCBO_ADMIN_BAG, &hbagRequest)

/* Create a bag to be filled with the response from the command server */
mqCreateBag(MQCBO_ADMIN_BAG, &hbagResponse)

/* Inquire about a queue by supplying its name */
/* (other parameters are optional) */
mqAddString(hbagRequest, MQCA_Q_NAME, "QBERT")

/* Request the command server to inquire about the queue */
mqExecute(MQCMD_INQUIRE_Q, hbagRequest, hbagResponse)

/* If it worked, the attributes of the queue are returned */
/* in a system bag within the response bag */
mqInquireBag(hbagResponse, MQHA_BAG_HANDLE, 0, &hbagAttributes)

/* Inquire the name of the queue and its current depth */
mqInquireString(hbagAttributes, MQCA_Q_NAME, &stringAttribute)
mqInquireString(hbagAttributes, MQIA_CURRENT_Q_DEPTH, &integerAttribute)

/* Tidy up memory allocated */
mqDeleteBag(hbagRequest)
mqDeleteBag(hbagResponse)
```

Figure 11. Using mqExecute to inquire about queue attributes

Using mqExecute is the simplest way of administering WebSphere MQ, but lower-level calls, mqBagToBuffer and mqBufferToBag, can be used. For more information about the use of these calls, see Chapter 16, “Exchanging data between applications”, on page 251.

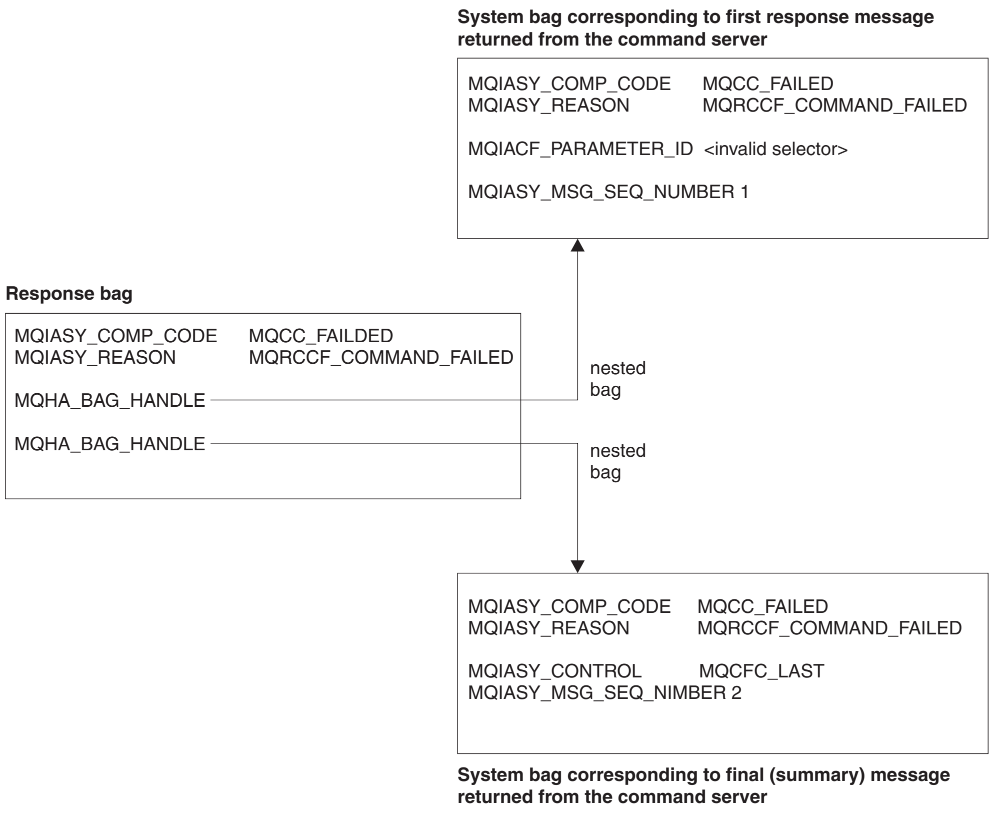
For sample programs, see Chapter 18, “Examples of using the MQAI”, on page 315.

Hints and tips for configuring WebSphere MQ

The MQAI uses PCF messages to send administration commands to the command server rather than dealing directly with the command server itself. Here are some tips for configuring WebSphere MQ using the MQAI:

- Character strings in WebSphere MQ are blank padded to a fixed length. Using C, null-terminated strings can normally be supplied as input parameters to WebSphere MQ programming interfaces.
- To clear the value of a string attribute, set it to a single blank rather than an empty string.
- It is recommended that you know in advance the attributes that you want to change and that you inquire on just those attributes. This is because the number of attributes that can be returned by the Inquire Queue (Response) command is higher than the number of attributes that can be changed using the Change Queue command. (See part 1 of this book for details of these commands.) Therefore, you are not recommended to attempt to modify all the attributes that you inquire.
- If an MQAI call fails, some detail of the failure is returned to the response bag. Further detail can then be found in a nested bag that can be accessed by the selector MQHA_BAG_HANDLE. For example, if an mqExecute call fails with a reason code of MQRCCF_COMMAND_FAILED, this information is returned in the response bag. However, a possible reason for this reason code is that a selector specified was not valid for the type of command message and this detail of information is found in a nested bag that can be accessed via a bag handle.

The following diagram shows this:


System bag corresponding to first response message returned from the command server
MQIASY_COMP_CODE MQCC_FAILED
MQIASY_REASON MQRCCF_COMMAND_FAILED
MQIACF_PARAMETER_ID <invalid selector>
MQIASY_MSG_SEQ_NUMBER 1
Response bag
MQIASY_COMP_CODE MQCC_FAILDED
MQIASY_REASON MQRCCF_COMMAND_FAILED
MQHA_BAG_HANDLE
MQHA_BAG_HANDLE
nested bag
nested bag
MQIASY_COMP_CODE MQCC_FAILED
MQIASY_REASON MQRCCF_COMMAND_FAILED
MQIASY_CONTROL MQCFC_LAST
MQIASY_MSG_SEQ_NIMBER 2
System bag corresponding to final (summary) message returned from the command server

Chapter 16. Exchanging data between applications

The MQAI can also be used to exchange data between applications. The application data is sent in PCF format and packed and unpacked by the MQAI. If your message data consists of integers and character strings, you can use the MQAI to take advantage of WebSphere MQ built-in data conversion for PCF data. This avoids the need to write data-conversion exits. To exchange data, the sender must first create the message and send it to the receiving application. Then, the receiver must read the message and extract the data. This can be done in two ways:

1. Converting bags and buffers, that is, using the mqBagToBuffer and mqBufferToBag calls.
2. Putting and getting bags, that is, using the mqPutBag and mqGetBag calls to send and receive PCF messages.

Both of these options are described in this chapter.

Note: You cannot convert a bag containing nested bags into a message.

Converting bags and buffers

To send data between applications, firstly the message data is placed in a bag. Then, the data in the bag is converted into a PCF message using the mqBagToBuffer call. The PCF message is sent to the required queue using the MQPUT call. This is shown in Figure 12. For a full description of the mqBagToBuffer call, see "mqBagToBuffer" on page 262.

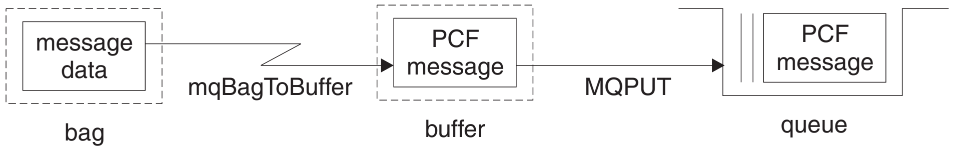


Figure 12. Converting bags to PCF messages

To receive data, the message is received into a buffer using the MQGET call. The data in the buffer is then converted into a bag using the mqBufferToBag call, providing the buffer contains a valid PCF message. This is shown in Figure 13. For a full description of the mqBufferToBag call, see "mqBufferToBag" on page 265.

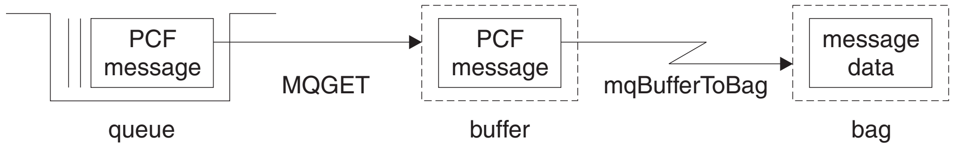


Figure 13. Converting PCF messages to bag form

Putting and receiving data bags

Data can also be sent between applications by putting and getting data bags using the mqPutBag and mqGetBag calls. This lets the MQAI handle the buffer rather than the application. The mqPutBag call converts the contents of the specified bag into a PCF message and sends the message to the specified queue and the mqGetBag call removes the message from the specified queue and converts it back into a data bag. Therefore, the mqPutBag call is the equivalent of the mqBagToBuffer call followed by MQPUT, and the mqGetBag is the equivalent of the MQGET call followed by mqBufferToBag.

Note: If you choose to use the mqGetBag call, the PCF details within the message must be correct; if they are not, an appropriate error results and the PCF message is not returned.

Sending PCF messages to a specified queue

To send a message to a specified queue, the mqPutBag call converts the contents of the specified bag into a PCF message and sends the message to the specified queue. The contents of the bag are left unchanged after the call.

As input to this call, you must supply:
- An MQI connection handle.
- An object handle for the queue on which the message is to be placed.
- A message descriptor. For more information about the message descriptor, see the *WebSphere MQ Application Programming Reference*.
- Put Message Options using the MQPMO structure. For more information about the MQPMO structure, see the *WebSphere MQ Application Programming Reference*.
- The handle of the bag to be converted to a message.

  Note: If the bag contains an administration message and the mqAddInquiry call was used to insert values into the bag, the value of the MQIASY_COMMAND data item must be an INQUIRE command recognized by the MQAI.

For a full description of the mqPutBag call, see "mqPutBag" on page 301.

Receiving PCF messages from a specified queue

To receive a message from a specified queue, the mqGetBag call gets a PCF message from a specified queue and converts the message data into a data bag.

As input to this call, you must supply:
- An MQI connection handle.
- An object handle of the queue from which the message is to be read.
- A message descriptor. Within the MQMD structure, the `Format` parameter must be MQFMT_ADMIN, MQFMT_EVENT, or MQFMT_PCF.

  Note: If the message is received within a unit of work (that is, with the MQGMO_SYNCPOINT option) and the message has an unsupported format, the unit of work can be backed out. The message is then reinstated on the queue and can be retrieved using the MQGET call instead of the mqGetBag call. For more information about the message descriptor, see the *WebSphere MQ Application Programming Reference*.

- Get Message Options using the MQGMO structure. For more information about the MQGMO structure, see the *WebSphere MQ Application Programming Reference*.
- The handle of the bag to contain the converted message.

For a full description of the mqGetBag call, see “mqGetBag” on page 283.

Chapter 17. MQAI reference

This chapter contains reference information for the MQAI. There are three types of call:

- Data-bag manipulation calls for configuring data bags:
  - "mqAddInquiry" on page 256
  - "mqAddInteger" on page 258
  - "mqAddString" on page 260
  - "mqClearBag" on page 267
  - "mqCountItems" on page 268
  - "mqCreateBag" on page 270
  - "mqDeleteBag" on page 274
  - "mqDeleteItem" on page 276
  - "mqInquireBag" on page 286
  - "mqInquireInteger" on page 289
  - "mqInquireItemInfo" on page 292
  - "mqInquireString" on page 295
  - "mqSetInteger" on page 304
  - "mqSetString" on page 307
  - "mqTruncateBag" on page 312
- Command calls for sending and receiving administration commands and PCF messages:
  - "mqBagToBuffer" on page 262
  - "mqBufferToBag" on page 265
  - "mqExecute" on page 279
  - "mqGetBag" on page 283
  - "mqPutBag" on page 301
- Utility calls for handling blank-padded and null-terminated strings:
  - "mqPad" on page 299
  - "mqTrim" on page 310

These calls are described in alphabetical order in the following sections.

mqAddInquiry

Note: The mqAddInquiry call can be used with administration bags only; it is specifically for administration purposes.

The mqAddInquiry call adds a selector to an administration bag. The selector refers to a WebSphere MQ object attribute that is to be returned by a PCF INQUIRE command. The value of the `Selector` parameter specified on this call is added to the end of the bag, as the value of a data item that has the selector value MQIACF_INQUIRY.

Syntax

mqAddInquiry *(Bag, Selector, CompCode, Reason)*

Parameters

Bag (MQHBAG) – input
: Bag handle.

  The bag must be an administration bag; that is, it must have been created with the MQCBO_ADMIN_BAG option on the mqCreateBag call. If the bag was not created this way, MQRC_BAG_WRONG_TYPE results.

Selector (MQLONG) – input
: Selector of the WebSphere MQ object attribute that is to be returned by the appropriate INQUIRE administration command.

CompCode (MQLONG) – output
: Completion code.

Reason (MQLONG) – output
: Reason code qualifying *CompCode*.

  The following reason codes indicate error conditions that can be returned from the mqAddInquiry call:

  MQRC_BAG_WRONG_TYPE
  Wrong type of bag for intended use.

  MQRC_HBAG_ERROR
  Bag handle not valid.

  MQRC_SELECTOR_OUT_OF_RANGE
  Selector not within valid range for call.

  MQRC_STORAGE_NOT_AVAILABLE
  Insufficient storage available.

  MQRC_SYSTEM_BAG_NOT_ALTERABLE
  System bag cannot be altered or deleted.

Usage notes

1. When the administration message is generated, the MQAI constructs an integer list with the MQIACF_*_ATTRS or MQIACH_*_ATTRS selector that is appropriate to the `Command` value specified on the mqExecute, mqPutBag, or mqBagToBuffer call. It then adds the values of the attribute selectors specified by the mqAddInquiry call.

2. If the `Command` value specified on the mqExecute, mqPutBag, or mqBagToBuffer call is not recognized by the MQAI, MQRC_INQUIRY_COMMAND_ERROR results. Instead of using the mqAddInquiry call, this can be overcome by using the mqAddInteger call with the appropriate MQIACF_*_ATTRS or MQIACH_*_ATTRS selector and the `ItemValue` parameter of the selector being inquired.

C language invocation

```
mqAddInquiry (Bag, Selector, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;           /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows only.)

```
mqAddInquiry Bag, Selector, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag      As Long 'Bag handle'
Dim Selector As Long 'Selector'
Dim CompCode As Long 'Completion code'
Dim Reason   As Long 'Reason code qualifying CompCode'
```

Supported INQUIRE command codes

- MQCMD_INQUIRE_Q_MGR
- MQCMD_INQUIRE_PROCESS
- MQCMD_INQUIRE_Q
- MQCMD_INQUIRE_Q_STATUS
- MQCMD_INQUIRE_CHANNEL
- MQCMD_INQUIRE_CHANNEL_STATUS
- MQCMD_INQUIRE_NAMELIST
- MQCMD_INQUIRE_NAMELIST_NAMES
- MQCMD_INQUIRE_CLUSTER_Q_MGR
- MQCMD_INQUIRE_AUTH_INFO
- MQCMD_INQUIRE_Q_STATUS

For an example that demonstrates the use of supported INQUIRE command codes, see “Inquiring about queues and printing information (amqsailq.c)” on page 321.

mqAddInteger

The mqAddInteger call adds an integer item identified by a user selector to the end of a specified bag.

Syntax

mqAddInteger *(Bag, Selector, ItemValue, CompCode, Reason)*

Parameters

Bag (MQHBAG) – input
Handle of the bag to be modified.

This must be the handle of a bag created by the user, not the handle of a system bag. MQRC_SYSTEM_BAG_NOT_ALTERABLE results if the value you specify identifies a system bag.

Selector (MQLONG) – input
Selector identifying the item to be added to the bag.

If the selector is less than zero (that is, a system selector), MQRC_SELECTOR_OUT_OF_RANGE results.

If the selector is zero or greater (that is, a user selector) and the bag was created with the MQCBO_CHECK_SELECTORS option or as an administration bag (MQCBO_ADMIN_BAG), the selector must be in the range MQIA_FIRST through MQIA_LAST; if not, again MQRC_SELECTOR_OUT_OF_RANGE results.

If MQCBO_CHECK_SELECTORS was not specified, the selector can be any value of zero or greater.

If the call is creating a second or later occurrence of a selector that is already in the bag, the datatype of this occurrence must be the same as the datatype of the first occurrence; MQRC_INCONSISTENT_ITEM_TYPE results if it is not.

ItemValue (MQLONG) – input
The integer value to be placed in the bag.

CompCode (MQLONG) – output
Completion code.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

The following reason codes indicate error conditions that can be returned from the mqAddInteger call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INCONSISTENT_ITEM_TYPE
Datatype of this occurrence of selector differs from datatype of first occurrence.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

Usage notes

1. If a data item with the specified selector is already present in the bag, an additional instance of that selector is added to the end of the bag. The new instance is not necessarily adjacent to the existing instance.
2. This call cannot be used to add a system selector to a bag.

C language invocation

```
mqAddInteger (Bag, Selector, ItemValue, &CompCode, &Reason)
```

Declare the parameters as follows:

```
MQHBAG   Bag;       /* Bag handle */
MQLONG   Selector;  /* Selector */
MQLONG   Item Value;/* Integer value */
MQLONG   CompCode;  /* Completion code */
MQLONG   Reason;    /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows only.)

```
mqAddInteger Bag, Selector, ItemValue, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag       As Long 'Bag handle'
Dim Selector  As Long 'Selector'
Dim ItemValue As Long 'Integer value'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

mqAddString

The mqAddString call adds a character data item identified by a user selector to the end of a specified bag.

Syntax

mqAddString *(Bag, Selector, BufferLength, Buffer, CompCode, Reason)*

Parameters

Bag (MQHBAG) – input
: Handle of the bag to be modified.

  This value must be the handle of a bag created by the user, not the handle of a system bag. MQRC_SYSTEM_BAG_NOT_ALTERABLE results if the value you specify relates to a system bag.

Selector (MQLONG) – input
: Selector identifying the item to be added to the bag.

  If the selector is less than zero (that is, a system selector), MQRC_SELECTOR_OUT_OF_RANGE results.

  If the selector is zero or greater (that is, a user selector), and the bag was created with the MQCBO_CHECK_SELECTORS option or as an administration bag (MQCBO_ADMIN_BAG), the selector must be in the range MQCA_FIRST through MQCA_LAST. MQRC_SELECTOR_OUT_OF_RANGE results if it is not in the correct range.

  If MQCBO_CHECK_SELECTORS was not specified, the selector can be any value zero or greater.

  If the call is creating a second or later occurrence of a selector that is already in the bag, the datatype of this occurrence must be the same as the datatype of the first occurrence; MQRC_INCONSISTENT_ITEM_TYPE results if it is not.

BufferLength (MQLONG) – input
: The length in bytes of the string contained in the *Buffer* parameter. The value must be zero or greater, or the special value MQBL_NULL_TERMINATED:

  - If MQBL_NULL_TERMINATED is specified, the string is delimited by the first null encountered in the string. The null is not added to the bag as part of the string.
  - If MQBL_NULL_TERMINATED is not specified, *BufferLength* characters are inserted into the bag, even if null characters are present. Nulls do not delimit the string.

Buffer (MQCHAR × *BufferLength*) – input
: Buffer containing the character string.

  The length is given by the *BufferLength* parameter. If zero is specified for *BufferLength*, the null pointer can be specified for the address of the *Buffer* parameter. In all other cases, a valid (nonnull) address must be specified for the *Buffer* parameter.

CompCode (MQLONG) – output
: Completion code.

Reason (MQLONG) – output
: Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqAddString call:

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not completely accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_CODED_CHAR_SET_ID_ERROR
Bag CCSID is MQCCSI_EMBEDDED.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INCONSISTENT_ITEM_TYPE
Datatype of this occurrence of selector differs from datatype of first occurrence.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

Usage notes

1. If a data item with the specified selector is already present in the bag, an additional instance of that selector is added to the end of the bag. The new instance is not necessarily adjacent to the existing instance.
2. This call cannot be used to add a system selector to a bag.
3. The Coded Character Set ID associated with this string is copied from the current CCSID of the bag.

C language invocation

```
mqAddString (hBag, Selector, BufferLength, Buffer, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   hBag;          /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   BufferLength;  /* Buffer length */
PMQCHAR  Buffer         /* Buffer containing item value */
MQLONG   CompCode;       /* Completion code */
MQLONG   Reason;         /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows only.)

```
mqAddString Bag, Selector, BufferLength, Buffer, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag          As Long 'Bag handle'
Dim Selector     As Long 'Selector'
Dim BufferLength As Long 'Buffer length'
Dim Buffer       As Long 'Buffer containing item value'
Dim CompCode     As Long 'Completion code'
Dim Reason       As Long 'Reason code qualifying CompCode'
```

mqBagToBuffer

The mqBagToBuffer call converts the bag into a PCF message in the supplied buffer.

Syntax

mqBagToBuffer *(OptionsBag, DataBag, BufferLength, Buffer, DataLength, CompCode, Reason)*

Parameters

OptionsBag (MQHBAG) – input
: Handle of the bag containing options that control the processing of the call. This is a reserved parameter; the value must be MQHB_NONE.

DataBag (MQHBAG) – input
: The handle of the bag to convert.

  If the bag contains an administration message and mqAddInquiry was used to insert values into the bag, the value of the MQIASY_COMMAND data item must be an INQUIRE command that is recognized by the MQAI; MQRC_INQUIRY_COMMAND_ERROR results if it is not.

  If the bag contains nested bags, MQRC_NESTED_BAG_NOT_SUPPORTED results.

BufferLength (MQLONG) – input
: Length in bytes of the buffer supplied.

  If the buffer is too small to accommodate the message generated, MQRC_BUFFER_LENGTH_ERROR results.

Buffer (MQBYTE × *BufferLength*) – output
: The buffer to hold the message.

DataLength (MQLONG) – output
: The length in bytes of the buffer required to hold the entire bag. If the buffer is not long enough, the contents of the buffer are undefined but the `DataLength` is returned.

CompCode (MQLONG) – output
: Completion code.

Reason (MQLONG) – output
: Reason code qualifying *CompCode*.

  The following reason codes indicating error conditions can be returned from the mqBagToBuffer call:

  MQRC_BUFFER_ERROR
  : Buffer parameter not valid (invalid parameter address or buffer not accessible).

  MQRC_BUFFER_LENGTH_ERROR
  : Buffer length not valid or buffer too small. (Required length returned in *DataLength*.)

  MQRC_DATA_LENGTH_ERROR
  : *DataLength* parameter not valid (invalid parameter address).

MQRC_HBAG_ERROR
: Bag handle not valid.

MQRC_INQUIRY_COMMAND_ERROR
: mqAddInquiry used with a command code that is not recognized as an INQUIRE command.

MQRC_NESTED_BAG_NOT_SUPPORTED
: Input data bag contains one or more nested bags.

MQRC_OPTIONS_ERROR
: Options bag contains unsupported data items or a supported option has an invalid value.

MQRC_PARAMETER_MISSING
: An administration message requires a parameter that is not present in the bag.

  Note: This reason code occurs for bags created with the MQCBO_ADMIN_BAG or MQCBO_REORDER_AS_REQUIRED options only.

MQRC_SELECTOR_WRONG_TYPE
: mqAddString or mqSetString was used to add the MQIACF_INQUIRY selector to the bag.

MQRC_STORAGE_NOT_AVAILABLE
: Insufficient storage available.

Usage notes

1. The PCF message is generated with an encoding of MQENC_NATIVE for the numeric data.
2. The buffer that holds the message can be null if the `BufferLength` is zero. This is useful if you use the mqBagToBuffer call to calculate the size of buffer necessary to convert your bag.

C language invocation

```
mqBagToBuffer (OptionsBag, DataBag, BufferLength, Buffer, &DataLength,
&CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   OptionsBag;    /* Options bag handle */
MQHBAG   DataBag;       /* Data bag handle */
MQLONG   BufferLength;  /* Buffer length */
MQBYTE  Buffer[n];      /* Buffer to contain PCF */
MQLONG  DataLength;     /* Length of PCF returned in buffer */
MQLONG  CompCode;       /* Completion code */
MQLONG  Reason;         /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows only.)

```
mqBagToBuffer OptionsBag, DataBag, BufferLength, Buffer, DataLength,
CompCode, Reason
```

Declare the parameters as follows:

```
Dim OptionsBag   As Long 'Options bag handle'
Dim DataBag      As Long 'Data bag handle'
Dim BufferLength As Long 'Buffer length'
Dim Buffer       As Long 'Buffer to contain PCF'
Dim DataLength   As Long 'Length of PCF returned in buffer'
Dim CompCode     As Long 'Completion code'
Dim Reason       As Long 'Reason code qualifying CompCode'
```

mqBufferToBag

The mqBufferToBag call converts the supplied buffer into bag form.

Syntax

mqBufferToBag *(OptionsBag, BufferLength, Buffer, DataBag, CompCode, Reason)*

Parameters

OptionsBag (MQHBAG) – input
Handle of the bag containing options that control the processing of the call. This is a reserved parameter; the value must be MQHB_NONE.

BufferLength (MQLONG) – input
Length in bytes of the buffer.

Buffer (MQBYTE × *BufferLength*) – input
Pointer to the buffer containing the message to be converted.

Databag (MQHBAG) – input/output
Handle of the bag to receive the message. The MQAI performs an mqClearBag call on the bag before placing the message in the bag.

CompCode (MQLONG) – output
Completion code.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqBufferToBag call:

MQRC_BAG_CONVERSION_ERROR
Data could not be converted into a bag. This indicates a problem with the format of the data to be converted into a bag (for example, the message is not a valid PCF).

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INCONSISTENT_ITEM_TYPE
Datatype of second occurrence of selector differs from datatype of first occurrence.

MQRC_OPTIONS_ERROR
Options bag contains unsupported data items, or a supported option has a value that is not valid.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

Usage notes

The buffer must contain a valid PCF message. The encoding of numeric data in the buffer must be MQENC_NATIVE.

The Coded Character Set ID of the bag is unchanged by this call.

C language invocation

```
mqBufferToBag (OptionsBag, BufferLength, Buffer, DataBag,
&CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   OptionsBag;    /* Options bag handle */
MQLONG   BufferLength;  /* Buffer length  */
MQBYTE   Buffer[n];     /* Buffer containing PCF */
MQHBAG   DataBag;       /* Data bag handle */
MQLONG  CompCode;       /* Completion code */
MQLONG  Reason;         /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows only.)

```
mqBufferToBag OptionsBag, BufferLength, Buffer, DataBag,
CompCode, Reason
```

Declare the parameters as follows:

```
Dim OptionsBag   As Long 'Options bag handle'
Dim BufferLength As Long 'Buffer length'
Dim Buffer       As Long 'Buffer containing PCF'
Dim DataBag      As Long 'Data bag handle'
Dim CompCode     As Long 'Completion code'
Dim Reason       As Long 'Reason code qualifying CompCode'
```

mqClearBag

The mqClearBag call deletes all user items from the bag, and resets system items to their initial values.

Syntax

mqClearBag *(Bag, CompCode, Reason)*

Parameters

Bag (MQHBAG) – input
Handle of the bag to be cleared. This must be the handle of a bag created by the user, not the handle of a system bag. MQRC_SYSTEM_BAG_NOT_ALTERABLE results if you specify the handle of a system bag.

CompCode (MQLONG) – output
Completion code.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqClearBag call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

Usage notes

1. If the bag contains system bags, they are also deleted.
2. The call cannot be used to clear system bags.

C language invocation

```
mqClearBag (Bag, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;          /* Bag handle */
MQLONG   CompCode;     /* Completion code */
MQLONG   Reason;       /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows only.)

```
mqClearBag Bag, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag      As Long 'Bag handle'
Dim CompCode As Long 'Completion code'
Dim Reason   As Long 'Reason code qualifying CompCode'
```

mqCountItems

The mqCountItems call returns the number of occurrences of user items, system items, or both, that are stored in a bag with the same specific selector.

Syntax

mqCountItems *(Bag, Selector, ItemCount, CompCode, Reason)*

Parameters

Bag (MQHBAG) – input
Handle of the bag whose items are to be counted. This can be a user bag or a system bag.

Selector (MQLONG) – input
Selector of the data items to count.

If the selector is less than zero (a system selector), the selector must be one that is supported by the MQAI. MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

If the specified selector is not present in the bag, the call succeeds and zero is returned for *ItemCount*.

The following special values can be specified for *Selector*:

MQSEL_ALL_SELECTORS
All user and system items are to be counted.

MQSEL_ALL_USER_SELECTORS
All user items are to be counted; system items are excluded from the count.

MQSEL_ALL_SYSTEM_SELECTORS
All system items are to be counted; user items are excluded from the count.

ItemCount (MQLONG) – output
Number of items of the specified type in the bag (can be zero).

CompCode (MQLONG) – output
Completion code.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqCountItems call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_ITEM_COUNT_ERROR
ItemCount parameter not valid (invalid parameter address).

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

Usage notes

This call counts the number of data items, not the number of unique selectors in the bag. A selector can occur multiple times, so there may be fewer unique selectors in the bag than data items.

C language invocation

```
mqCountItems (Bag, Selector, &ItemCount, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;           /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG  ItemCount;      /* Number of items */
MQLONG  CompCode;       /* Completion code */
MQLONG  Reason;         /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows only.)

```
mqCountItems Bag, Selector, ItemCount, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag;      As Long 'Bag handle'
Dim Selector  As Long 'Selector'
Dim ItemCount As Long 'Number of items'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

mqCreateBag

The mqCreateBag call creates a new bag.

Syntax

mqCreateBag *(Options, Bag, CompCode, Reason)*

Parameters

Options (MQLONG) – input
Options for creation of the bag.

The following are valid:

MQCBO_ADMIN_BAG
Specifies that the bag is for administering WebSphere MQ objects. MQCBO_ADMIN_BAG automatically implies the MQCBO_LIST_FORM_ALLOWED, MQCBO_REORDER_AS_REQUIRED, and MQCBO_CHECK_SELECTORS options.

Administration bags are created with the MQIASY_TYPE system item set to MQCFT_COMMAND.

MQCBO_COMMAND_BAG
Specifies that the bag is a command bag. This is an alternative to the administration bag (MQCBO_ADMIN_BAG) and MQRC_OPTIONS_ERROR results if both are specified.

A command bag is processed in the same way as a user bag except that the value of the MQIASY_TYPE system item is set to MQCFT_COMMAND when the bag is created.

The command bag is also created for administering objects but they are not used to send administration messages to a command server as an administration bag is. The bag options assume the following default values:
- MQCBO_LIST_FORM_INHIBITIED
- MQCBO_DO_NOT_REORDER
- MQCBO_DO_NOT_CHECK_SELECTORS

Therefore, the MQAI will not change the order of data items or create lists within a message as with administration bags.

MQCBO_USER_BAG
Specifies that the bag is a user bag. This is the default bag-type option. User bags can also be used for the administration of WebSphere MQ objects, but the MQCBO_LIST_FORM_ALLOWED and MQCBO_REORDER_AS_REQUIRED options should be specified to ensure correct generation of the administration messages.

User bags are created with the MQIASY_TYPE system item set to MQCFT_USER.

For user bags, one or more of the following options can be specified:

MQCBO_LIST_FORM_ALLOWED
Specifies that the MQAI is allowed to use the more compact list form in the message sent whenever there are two or more adjacent occurrences of the same selector in the bag. However, this option does not allow the items to be reordered. Therefore, if the occurrences of the selector are not adjacent in the bag, and MQCBO_REORDER_AS_REQUIRED is not specified, the MQAI cannot use the list form for that particular selector.

If the data items are character strings, these strings must have the same Character Set ID as well as the same selector, in order to be compacted into list form. If the list form is used, the shorter strings are padded with blanks to the length of the longest string.

This option should be specified if the message to be sent is an administration message but MQCBO_ADMIN_BAG is not specified.

Note: MQCBO_LIST_FORM_ALLOWED does not imply that the MQAI will definitely use the list form. The MQAI considers various factors in deciding whether to use the list form.

MQCBO_LIST_FORM_INHIBITED
Specifies that the MQAI is not allowed to use the list form in the message sent, even if there are adjacent occurrences of the same selector in the bag. This is the default list-form option.

MQCBO_REORDER_AS_REQUIRED
Specifies that the MQAI is allowed to change the order of the data items in the message sent. This option does not affect the order of the items in the sending bag.

This means that you can insert items into a data bag in any order; that is, the items do not need to be inserted in the way that they must appear in the PCF message, because the MQAI can reorder these items as required.

If the message is a user message, the order of the items in the receiving bag will be the same as the order of the items in the message; this may be different from the order of the items in the sending bag.

If the message is an administration message, the order of the items in the receiving bag will be determined by the message received.

This option should be specified if the message to be sent is an administration message but MQCBO_ADMIN is not specified.

MQCBO_DO_NOT_REORDER
Specifies that the MQAI is not allowed to change the order of data items in the message sent. Both the message sent and the receiving bag contain the items in the same order as they occur in the sending bag. This is the default ordering option.

MQCBO_CHECK_SELECTORS
Specifies that user selectors (selectors that are zero or greater) should be checked to ensure that the selector is consistent with the datatype implied by the mqAddInteger, mqAddString, mqSetInteger, or mqSetString call:

- For the integer calls, the selector must be in the range MQIA_FIRST through MQIA_LAST.
- For the string calls, the selector must be in the range MQCA_FIRST through MQCA_LAST.
- For the handle calls, the selector must be in the range MQHA_FIRST through MQHA_LAST.

The call fails if the selector is outside the valid range. Note that system selectors (selectors less than zero) are always checked, and if a system selector is specified, it must be one that is supported by the MQAI.

MQCBO_DO_NOT_CHECK_SELECTORS
Specifies that user selectors (selectors that are zero or greater) should not be checked. This option allows any selector that is zero or positive to be used with any call. This is the default selectors option. Note that system selectors (selectors less than zero) are always checked.

MQCBO_NONE
Specifies that all options should have their default values. This is provided to aid program documentation, and should not be specified with any of the options that has a nonzero value.

The following list summarizes the default option values:
- MQCBO_USER_BAG
  - MQCBO_LIST_FORM_INHIBITIED
  - MQCBO_DO_NOT_REORDER
  - MQCBO_DO_NOT_CHECK_SELECTORS

Bag (MQHBAG) – output
The handle of the bag created by the call.

CompCode (MQLONG) – output
Completion code.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqCreateBag call:

MQRC_HBAG_ERROR
Bag handle not valid (invalid parameter address or the parameter location is read-only).

MQRC_OPTIONS_ERROR
Options not valid or not consistent.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

Usage notes

Any options used for creating your bag are contained in a system item within the bag when it is created.

C language invocation

```
mqCreateBag (Options, &Bag, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQLONG   Options;       /* Bag options */
MQHBAG  Bag;            /* Bag handle */
MQLONG  CompCode;       /* Completion code */
MQLONG  Reason;         /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows only.)

```
mqCreateBag Options, Bag, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Options  As Long 'Bag options'
Dim Bag      As Long 'Bag handle'
Dim CompCode As Long 'Completion code'
Dim Reason   As Long 'Reason code qualifying CompCode'
```

mqDeleteBag

The mqDeleteBag call deletes the specified bag.

Syntax

mqDeleteBag *(Bag, CompCode, Reason)*

Parameters

Bag (MQHBAG) – input/output
The handle of the bag to be deleted. This must be the handle of a bag created by the user, not the handle of a system bag. MQRC_SYSTEM_BAG_NOT_DELETABLE results if you specify the handle of a system bag. The handle is reset to MQHB_UNUSABLE_HBAG.

If the bag contains system-generated bags, they are also deleted.

CompCode (MQLONG) – output
Completion code.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqDeleteBag call:

MQRC_HBAG_ERROR
Bag handle not valid, or invalid parameter address, or parameter location is read only.

MQRC_SYSTEM_BAG_NOT_DELETABLE
System bag cannot be deleted.

Usage notes

1. Delete any bags created with mqCreateBag.
2. Nested bags are deleted automatically when the containing bag is deleted.

C language invocation

```
mqDeleteBag (&Bag, CompCode, Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;          /* Bag handle */
MQLONG   CompCode;     /* Completion code */
MQLONG   Reason;       /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows only.)

```
mqDeleteBag Bag, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag;     As Long 'Bag handle'
Dim CompCode As Long 'Completion code'
Dim Reason   As Long 'Reason code qualifying CompCode'
```

mqDeleteItem

The mqDeleteItem call removes one or more user items from a bag.

Syntax

mqDeleteItem *(Bag, Selector, ItemIndex, CompCode, Reason)*

Parameters

Hbag (MQHBAG) – input
Handle of the bag to be modified.

This must be the handle of a bag created by the user, and not the handle of a system bag; MQRC_SYSTEM_BAG_NOT_ALTERABLE results if it is a system bag.

Selector (MQLONG) – input
Selector identifying the user item to be deleted.

If the selector is less than zero (that is, a system selector), MQRC_SELECTOR_OUT_OF_RANGE results.

The following special values are valid:

MQSEL_ANY_SELECTOR
The item to be deleted is a user item identified by the `ItemIndex` parameter, the index relative to the set of items that contains both user and system items.

MQSEL_ANY_USER_SELECTOR
The item to be deleted is a user item identified by the `ItemIndex` parameter, the index relative to the set of user items.

If an explicit selector value is specified, but the selector is not present in the bag, the call succeeds if MQIND_ALL is specified for `ItemIndex`, and fails with reason code MQRC_SELECTOR_NOT_PRESENT if MQIND_ALL is not specified.

ItemIndex (MQLONG) – input
Index of the data item to be deleted.

The value must be zero or greater, or one of the following special values:

MQIND_NONE
This specifies that there must be one occurrence only of the selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results. If MQIND_NONE is specified with one of the MQSEL_XXX_SELECTOR values, MQRC_INDEX_ERROR results.

MQIND_ALL
This specifies that all occurrences of the selector in the bag are to be deleted. If MQIND_ALL is specified with one of the MQSEL_XXX_SELECTOR values, MQRC_INDEX_ERROR results. If MQIND_ALL is specified when the selector is not present within the bag, the call succeeds.

If MQSEL_ANY_SELECTOR is specified for the `Selector` parameter, the `ItemIndex` parameter is the index relative to the set of items that contains both user items and system items, and must be zero or greater. If ItemIndex identifies a system selector MQRC_SYSTEM_ITEM_NOT_DELETABLE results. If MQSEL_ANY_USER_SELECTOR is specified for the Selector parameter, the ItemIndex parameter is the index relative to the set of user items, and must be zero or greater.

If an explicit selector value is specified, ItemIndex is the index relative to the set of items that have that selector value, and can be MQIND_NONE, MQIND_ALL, zero, or greater.

If an explicit index is specified (that is, not MQIND_NONE or MQIND_ALL) and the item is not present in the bag, MQRC_INDEX_NOT_PRESENT results.

CompCode (MQLONG) – output
Completion code.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqDeleteItem call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
MQIND_NONE or MQIND_ALL specified with one of the MQSEL_ANY_XXX_SELECTOR values.

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag.

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag is read only and cannot be altered.

MQRC_SYSTEM_ITEM_NOT_DELETABLE
System item is read only and cannot be deleted.

Usage notes

1. Either a single occurrence of the specified selector can be removed, or all occurrences of the specified selector.
2. The call cannot remove system items from the bag, or remove items from a system bag. However, the call can remove the handle of a system bag from a user bag. This way, a system bag can be deleted.

C language invocation

```
mqDeleteItem (Bag, Selector, ItemIndex, &CompCode, &Reason)
```

Declare the parameters as follows:

```
MQHBAG   Hbag;         /* Bag handle */
MQLONG   Selector;     /* Selector */
MQLONG   ItemIndex;    /* Index of the data item */
MQLONG   CompCode;     /* Completion code */
MQLONG   Reason;       /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows only.)

```
mqDeleteItem Bag, Selector, ItemIndex, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag       As Long 'Bag handle'
Dim Selector  As Long 'Selector'
Dim ItemIndex As Long 'Index of the data item'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

mqExecute

The mqExecute call sends an administration command message and waits for the reply (if expected).

Syntax

mqExecute *(Hconn, Command, OptionsBag, AdminBag, ResponseBag, AdminQ, ResponseQ, CompCode, Reason)*

Parameters

Hconn (MQHCONN) – input
MQI Connection handle.

This is returned by a preceding MQCONN call issued by the application.

Command (MQLONG) – input
The command to be executed.

This should be one of the MQCMD_* values. If it is a value that is not recognized by the MQAI servicing the mqExecute call, the value is still accepted. However, if mqAddInquiry was used to insert values in the bag, the Command parameter must be an INQUIRE command recognized by the MQAI; MQRC_INQUIRY_COMMAND_ERROR results if it is not.

OptionsBag (MQHBAG) – input
Handle of a bag containing options that affect the operation of the call.

This must be the handle returned by a preceding mqCreateBag call or the following special value:

MQHB_NONE
No options bag; all options assume their default values.

Only the options listed below can be present in the options bag (MQRC_OPTIONS_ERROR results if other data items are present).

The appropriate default value is used for each option that is not present in the bag. The following option can be specified:

MQIACF_WAIT_INTERVAL
This data item specifies the maximum time in milliseconds that the MQAI should wait for each reply message. The time interval must be zero or greater, or the special value MQWI_UNLIMITED; the default is thirty seconds. The mqExecute call completes either when all of the reply messages are received or when the specified wait interval expires without the expected reply message having been received.

Note: The time interval is an approximate quantity.

If the MQIACF_WAIT_INTERVAL data item has the wrong datatype, or there is more than one occurrence of that selector in the options bag, or the value of the data item is not valid, MQRC_WAIT_INTERVAL_ERROR results.

AdminBag (MQHBAG) – input
Handle of the bag containing details of the administration command to be issued.

All user items placed in the bag are inserted into the administration message that is sent. It is the application's responsibility to ensure that only valid parameters for the command are placed in the bag.

If the value of the MQIASY_TYPE data item in the command bag is not MQCFT_COMMAND, MQRC_COMMAND_TYPE_ERROR results. If the bag contains nested bags, MQRC_NESTED_BAG_NOT_SUPPORTED results.

ResponseBag (MQHBAG) – input
Handle of the bag where reply messages are placed.

The MQAI performs an mqClearBag call on the bag before placing reply messages in the bag. To retrieve the reply messages, the selector, MQIACF_CONVERT_RESPONSE, can be specified.

Each reply message is placed into a separate system bag, whose handle is then placed in the response bag. Use the mqInquireBag call with selector MQHA_BAG_HANDLE to determine the handles of the system bags within the reply bag, and those bags can then be inquired to determine their contents.

If some but not all of the expected reply messages are received, MQCC_WARNING with MQRC_NO_MSG_AVAILABLE results. If none of the expected reply messages is received, MQCC_FAILED with MQRC_NO_MSG_AVAILABLE results.

AdminQ (MQHOBJ) – input
Object handle of the queue on which the administration message is to be placed.

This handle was returned by a preceding MQOPEN call issued by the application. The queue must be open for output.

The following special value can be specified:

MQHO_NONE
This indicates that the administration message should be placed on the SYSTEM.ADMIN.COMMAND.QUEUE belonging to the currently connected queue manager. If MQHO_NONE is specified, the application need not use MQOPEN to open the queue.

ResponseQ
Object handle of the queue on which reply messages are placed.

This handle was returned by a preceding MQOPEN call issued by the application. The queue must be open for input and for inquiry.

The following special value can be specified:

MQHO_NONE
This indicates that the reply messages should be placed on a dynamic queue created automatically by the MQAI. The queue is created by opening SYSTEM.DEFAULT.MODEL.QUEUE, that must therefore have suitable characteristics. The queue created exists for the duration of the call only, and is deleted by the MQAI on exit from the mqExecute call.

CompCode
Completion code.

Reason
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqExecute call:

MQRC_*
: Anything from the MQINQ, MQPUT, MQGET, or MQOPEN calls.

MQRC_CMD_SERVER_NOT_AVAILABLE
: The command server that processes administration commands is not available.

MQRC_COMMAND_TYPE_ERROR
: The value of the MQIASY_TYPE data item in the request bag is not MQCFT_COMMAND.

MQRC_HBAG_ERROR
: Bag handle not valid.

MQRC_INQUIRY_COMMAND_ERROR
: mqAddInteger call used with a command code that is not a recognized INQUIRE command.

MQRC_NESTED_BAG_NOT_SUPPORTED
: Input data bag contains one or more nested bags.

MQRC_NO_MSG_AVAILABLE
: Some reply messages received, but not all. Reply bag contains system-generated bags for messages that were received.

MQRC_NO_MSG_AVAILABLE
: No reply messages received during the specified wait interval.

MQRC_OPTIONS_ERROR
: Options bag contains unsupported data items, or a supported option has a value which is not valid.

MQRC_PARAMETER_MISSING
: Administration message requires a parameter which is not present in the bag. This reason code occurs for bags created with the MQCBO_ADMIN_BAG or MQCBO_REORDER_AS_REQUIRED options only.

MQRC_SELECTOR_NOT_UNIQUE
: Two or more instances of a selector exist within the bag for a mandatory parameter that permits one instance only.

MQRC_SELECTOR_WRONG_TYPE
: mqAddString or mqSetString was used to add the MQIACF_INQUIRY selector to the bag.

MQRC_STORAGE_NOT_AVAILABLE
: Insufficient storage available.

MQRCCF_COMMAND_FAILED
: Command failed; details of failure are contained in system-generated bags within the reply bag.

Usage notes

1. If no *AdminQ* is specified, the MQAI checks to see if the command server is active before sending the administration command message. However, if the command server is not active, the MQAI does not start it. If you are sending a large number of administration command messages, you are recommended to open the SYSTEM.ADMIN.COMMAND.QUEUE yourself and pass the handle of the administration queue on each administration request.
2. Specifying the MQHO_NONE value in the *ResponseQ* parameter simplifies the use of the mqExecute call, but if mqExecute is issued repeatedly by the

application (for example, from within a loop), the response queue will be created and deleted repeatedly. In this situation, it is better for the application itself to open the response queue prior to any mqExecute call, and close it after all mqExecute calls have been issued.

3. If the administration command results in a message being sent with a message type of MQMT_REQUEST, the call waits for the period of time given by the MQIACF_WAIT_INTERVAL data item in the options bag.
4. If an error occurs during the processing of the call, the response bag may contain some data from the reply message, but the data will usually be incomplete.

C language invocation

```
mqExecute (Hconn, Command, OptionsBag, AdminBag, ResponseBag,
AdminQ, ResponseQ, CompCode, Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;        /* MQI connection handle */
MQLONG   Command;      /* Command to be executed */
MQHBAG   OptionsBag;   /* Handle of a bag containing options */
MQHBAG   AdminBag;     /* Handle of administration bag containing
                       /* details of administration command */
MQHBAG   ResponseBag;  /* Handle of bag for response messages */
MQHOBJ   AdminQ        /* Handle of administration queue for
                          administration messages */
MQHOBJ   ResponseQ;    /* Handle of response queue for response
                          messages */
MQLONG   pCompCode;    /* Completion code */
MQLONG   pReason;      /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows only.)

```
mqExecute (Hconn, Command, OptionsBag, AdminBag, ResponseBag,
AdminQ, ResponseQ, CompCode, Reason);
```

Declare the parameters as follows:

```
Dim HConn       As Long 'MQI connection handle'
Dim Command     As Long 'Command to be executed'
Dim OptionsBag  As Long 'Handle of a bag containing options'
Dim AdminBag  As Long 'Handle of command bag containing details of
                        administration command'
Dim ResponseBag As Long 'Handle of bag for reply messages'
Dim AdminQ    As Long 'Handle of command queue for
                        administration messages'
Dim ResponseQ   As Long 'Handle of response queue for reply messages'
Dim CompCode    As Long 'Completion code'
Dim Reason      As Long 'Reason code qualifying CompCode'
```

mqGetBag

The mqGetBag call removes a message from the specified queue and converts the message data into a data bag.

Syntax

mqGetBag *(Hconn, Hobj, MsgDesc, GetMsgOpts, Bag, CompCode, Reason)*

Parameters

Hconn (MQHCONN) – input
: MQI connection handle.

Hobj (MQHOBJ) – input
: Object handle of the queue from which the message is to be retrieved. This handle was returned by a preceding MQOPEN call issued by the application. The queue must be open for input.

MsgDesc (MQMD) – input/output
: Message descriptor (for more information, see the *WebSphere MQ Application Programming Reference*).

  If the *Format* field in the message has a value other than MQFMT_ADMIN, MQFMT_EVENT, or MQFMT_PCF, MQRC_FORMAT_NOT_SUPPORTED results.

  If, on entry to the call, the *Encoding* field in the application's MQMD has a value other than MQENC_NATIVE and MQGMO_CONVERT is specified, MQRC_ENCODING_NOT_SUPPORTED results. Also, if MQGMO_CONVERT is not specified, the value of the *Encoding* parameter must be the retrieving application's MQENC_NATIVE; if not, again MQRC_ENCODING_NOT_SUPPORTED results.

GetMsgOpts (MQGMO) – input/output
: Get-message options (for more information, see the *WebSphere MQ Application Programming Guide*).

  MQGMO_ACCEPT_TRUNCATED_MSG cannot be specified; MQRC_OPTIONS_ERROR results if it is. MQGMO_LOCK and MQGMO_UNLOCK are not supported in a 16-bit or 32-bit Window environment. MQGMO_SET_SIGNAL is supported in a 32-bit Window environment only.

Bag (MQHBAG) – input/output
: Handle of a bag into which the retrieved message is placed. The MQAI performs an mqClearBag call on the bag before placing the message in the bag.

  MQHB_NONE
  : Gets the retrieved message. This provides a means of deleting messages from the queue.

    If an option of MQGMO_BROWSE_* is specified, this value sets the browse cursor to the selected message; it is not deleted in this case.

CompCode (MQLONG) – output
: Completion code.

Reason (MQLONG) – output
: Reason code qualifying *CompCode*.

The following reason codes indicating warning and error conditions can be returned from the mqGetBag call:

MQRC_*
: Anything from the MQGET call or bag manipulation.

MQRC_BAG_CONVERSION_ERROR
: Data could not be converted into a bag.

  This indicates a problem with the format of the data to be converted into a bag (for example, the message is not a valid PCF).

  If the message was retrieved destructively from the queue (that is, not browsing the queue), this reason code indicates that it has been discarded.

MQRC_ENCODING_NOT_SUPPORTED
: Encoding not supported; the value in the *Encoding* field of the MQMD must be MQENC_NATIVE.

MQRC_FORMAT_NOT_SUPPORTED
: Format not supported; the *Format* name in the message is not MQFMT_ADMIN, MQFMT_EVENT, or MQFMT_PCF. If the message was retrieved destructively from the queue (that is, not browsing the queue), this reason code indicates that it has been discarded.

MQRC_HBAG_ERROR
: Bag handle not valid.

MQRC_INCONSISTENT_ITEM_TYPE
: Datatype of second occurrence of selector differs from datatype of first occurrence.

MQRC_SELECTOR_OUT_OF_RANGE
: Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
: Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
: System bag cannot be altered or deleted.

Usage notes

1. Only messages that have a supported format can be returned by this call. If the message has a format that is not supported, the message is discarded, and the call completes with an appropriate reason code.
2. If the message is retrieved within a unit of work (that is, with the MQGMO_SYNCPOINT option), and the message has an unsupported format, the unit of work can be backed out, reinstating the message on the queue. This allows the message to be retrieved by using the MQGET call in place of the mqGetBag call.

C language invocation

```
mqGetBag (hConn, hObj, &MsgDesc, &GetMsgOpts, hBag, CompCode, Reason);
```

Declare the parameters as follows:

```
MQHCONN  hConn;         /* MQI connection handle */
MQHOBJ   hObj;          /* Object handle */
MQMD     MsgDesc;       /* Message descriptor */
MQGMO    GetMsgOpts;    /* Get-message options */
```

```
MQHBAG   hBag;          /* Bag handle */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows only.)

```
mqGetBag (HConn, HObj, MsgDesc, GetMsgOpts, Bag, CompCode, Reason);
```

Declare the parameters as follows:

```
Dim HConn      As Long 'MQI connection handle'
Dim HObj       As Long 'Object handle'
Dim MsgDesc    As Long 'Message descriptor'
Dim GetMsgOpts As Long 'Get-message options'
Dim Bag        As Long 'Bag handle'
Dim CompCode   As Long 'Completion code'
Dim Reason     As Long 'Reason code qualifying CompCode'
```

mqInquireBag

The mqInquireBag call inquires the value of a bag handle that is present in the bag. The data item can be a user item or a system item.

Syntax

mqInquireBag *(Bag, Selector, ItemIndex, ItemValue, CompCode, Reason)*

Parameters

Bag (MQHBAG) – input
Bag handle to be inquired. The bag can be a user bag or a system bag.

Selector (MQLONG) – input
Selector identifying the item to be inquired.

If the selector is less than zero (that is, a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

The specified selector must be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

The datatype of the item must agree with the datatype implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

The following special values can be specified for `Selector`:

MQSEL_ANY_SELECTOR
The item to be inquired is a user or system item identified by the `ItemIndex` parameter.

MQSEL_ANY_USER_SELECTOR
The item to be inquired is a user item identified by the `ItemIndex` parameter.

MQSEL_ANY_SYSTEM_SELECTOR
The item to be inquired is a system item identified by the `ItemIndex` parameter.

ItemIndex (MQLONG) – input
Index of the data item to be inquired.

The value must be zero or greater, or the special value MQIND_NONE. If the value is less than zero and not MQIND_NONE, MQRC_INDEX_ERROR results. If the item is not already present in the bag, MQRC_INDEX_NOT_PRESENT results.

The following special value can be specified:

MQIND_NONE
This specifies that there must be one occurrence only of the selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

If MQSEL_ANY_SELECTOR is specified for the `Selector` parameter, the `ItemIndex` parameter is the index relative to the set of items that contains both user items and system items, and must be zero or greater.

If MQSEL_ANY_USER_SELECTOR is specified for the `Selector` parameter, the `ItemIndex` parameter is the index relative to the set of system items, and must be zero or greater.

If MQSEL_ANY_SYSTEM_SELECTOR is specified for the `Selector` parameter, the `ItemIndex` parameter is the index relative to the set of system items, and must be zero or greater.

If an explicit selector value is specified, the `ItemIndex` parameter is the index relative to the set of items that have that selector value and can be MQIND_NONE, zero, or greater.

ItemValue (MQHBAG) – output
Value of the item in the bag.

CompCode (MQLONG) – output
Completion code.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqInquireBag call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE, or MQIND_NONE specified with one of the MQSEL_ANY_xxx_SELECTOR values).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_ITEM_VALUE_ERROR
The `ItemValue` parameter is not valid (invalid parameter address).

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present within the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong datatype for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

C language invocation

```
mqInquireBag (Bag, Selector, ItemIndex, &ItemValue, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;           /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   ItemIndex;     /* Index of the data item to be inquired */
MQHBAG   ItemValue;     /* Value of item in the bag */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows only.)

```
mqInquireBag (Bag, Selector, ItemIndex, ItemValue, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag       As Long 'Bag handle'
Dim Selector  As Long 'Selector'
Dim ItemIndex As Long 'Index of the data item to be inquired'
Dim ItemValue As Long 'Value of item in the bag'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

mqInquireInteger

The mqInquireInteger call requests the value of an integer data item that is present in the bag. The data item can be a user item or a system item.

Syntax

mqInquireInteger *(Bag, Selector, ItemIndex, ItemValue, CompCode, Reason)*

Parameters

Bag (MQHBAG) – input
Handle of the bag to which the inquiry relates. The bag can be a user bag or a system bag.

Selector (MQLONG) – input
Selector identifying the item to which the inquiry relates.

If the selector is less than zero (a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

The specified selector must be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

The datatype of the item must agree with the datatype implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

The following special values can be specified for *Selector*:

MQSEL_ANY_SELECTOR
The item to be inquired about is a user or system item identified by *ItemIndex*.

MQSEL_ANY_USER_SELECTOR
The item to be inquired about is a user item identified by *ItemIndex*.

MQSEL_ANY_SYSTEM_SELECTOR
The item to be inquired about is a system item identified by *ItemIndex*.

ItemIndex (MQLONG) – input
Index of the data item to which the inquiry relates. The value must be zero or greater, or the special value MQIND_NONE. If the value is less than zero and is not MQIND_NONE, MQRC_INDEX_ERROR results. If the item is not already present in the bag, MQRC_INDEX_NOT_PRESENT results. The following special value can be specified:

MQIND_NONE
This specifies that there must be one occurrence only of the selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

If MQSEL_ANY_SELECTOR is specified for *Selector*, *ItemIndex* is the index relative to the set of items that contains both user items and system items, and must be zero or greater.

If MQSEL_ANY_USER_SELECTOR is specified for *Selector*, *ItemIndex* is the index relative to the set of user items, and must be zero or greater.

If MQSEL_ANY_SYSTEM_SELECTOR is specified for *Selector*, *ItemIndex* is the index relative to the set of system items, and must be zero or greater.

If an explicit selector value is specified, *ItemIndex* is the index relative to the set of items that have that selector value, and can be MQIND_NONE, zero, or greater.

ItemValue (MQLONG) – output
The value of the item in the bag.

CompCode (MQLONG) – output
Completion code.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqInquireInteger call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE, or MQIND_NONE specified with one of the MQSEL_ANY_xxx_SELECTOR values).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_ITEM_VALUE_ERROR
ItemValue parameter not valid (invalid parameter address).

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong datatype for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

C language invocation

```
mqInquireInteger (Bag, Selector, ItemIndex, &ItemValue,
&CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;          /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   ItemIndex;     /* Item index */
MQLONG   ItemValue;     /* Item value */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows only.)

```
mqInquireInteger Bag, Selector, ItemIndex, ItemValue,
CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag       As Long 'Bag handle'
Dim Selector  As Long 'Selector'
Dim ItemIndex As Long 'Item index'
Dim ItemValue As Long 'Item value'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

mqInquireItemInfo

The mqInquireItemInfo call returns information about a specified item in a bag. The data item can be a user item or a system item.

Syntax

mqInquireItemInfo *(Bag, Selector, ItemIndex, ItemType, OutSelector, CompCode, Reason)*

Parameters

Bag (MQHBAG) – input
: Handle of the bag to be inquired.

The bag can be a user bag or a system bag.

Selector (MQLONG) – input
: Selector identifying the item to be inquired.

If the selector is less than zero (that is, a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

The specified selector must be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

The following special values can be specified for `Selector`:

MQSEL_ANY_SELECTOR
: The item to be inquired is a user or system item identified by the `ItemIndex` parameter.

MQSEL_ANY_USER_SELECTOR
: The item to be inquired is a user item identified by the `ItemIndex` parameter.

MQSEL_ANY_SYSTEM_SELECTOR
: The item to be inquired is a system item identified by the `ItemIndex` parameter.

ItemIndex (MQLONG) – input
: Index of the data item to be inquired.

The item must be present within the bag; MQRC_INDEX_NOT_PRESENT results if it is not. The value must be zero or greater, or the following special value:

MQIND_NONE
: This specifies that there must be one occurrence only of the selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

If MQSEL_ANY_SELECTOR is specified for the `Selector` parameter, the `ItemIndex` parameter is the index relative to the set of items that contains both user items and system items, and must be zero or greater.

If MQSEL_ANY_USER_SELECTOR is specified for the `Selector` parameter, the `ItemIndex` parameter is the index relative to the set of system items, and must be zero or greater.

If MQSEL_ANY_SYSTEM_SELECTOR is specified for the `Selector` parameter, the `ItemIndex` parameter is the index relative to the set of system items, and must be zero or greater. If an explicit selector value is specified, the `ItemIndex` parameter is the index relative to the set of items that have that selector value and can be MQIND_NONE, zero, or greater.

ItemType (MQLONG) – output
: The datatype of the specified data item.

The following can be returned:

MQIT_BAG
Bag handle item.

MQIT_INTEGER
Integer item.

MQIT_STRING
Character-string item.

OutSelector (MQLONG) – output
Selector of the specified data item.

CompCode (MQLONG) – output
Completion code.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqInquireItemInfo call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
MQIND_NONE specified with one of the MQSEL_ANY_XXX_SELECTOR values.

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_ITEM_TYPE_ERROR
`ItemType` parameter not valid (invalid parameter address).

MQRC_OUT_SELECTOR_ERROR
`OutSelector` parameter not valid (invalid parameter address).

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

C language invocation

```
mqInquireItemInfo (Bag, Selector, ItemIndex, &OutSelector, &ItemType,
&CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;           /* Bag handle */
MQLONG   Selector;      /* Selector identifying item */
MQLONG   ItemIndex;     /* Index of data item */
MQLONG   OutSelector;   /* Selector of specified data item */
MQLONG   ItemType;      /* Data type of data item */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows only.)

```
mqInquireItemInfo Bag, Selector, ItemIndex, OutSelector, ItemType,
CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag         As Long 'Bag handle'
Dim Selector    As Long 'Selector identifying item'
Dim ItemIndex   As Long 'Index of data item'
Dim OutSelector As Long 'Selector of specified data item'
Dim ItemType    As Long 'Data type of data item'
Dim CompCode    As Long 'Completion code'
Dim Reason      As Long 'Reason code qualifying CompCode'
```

mqInquireString

The mqInquireString call requests the value of a character data item that is present in the bag. The data item can be a user item or a system item.

Syntax

mqInquireString *(Bag, Selector, ItemIndex, Bufferlength, Buffer, StringLength, CodedCharSetId, CompCode, Reason)*

Parameters

Bag (MQHBAG) – input
: Handle of the bag to which the inquiry relates. The bag can be a user bag or a system bag.

Selector (MQLONG) – input
: Selector of the item to which the inquiry relates.

  If the selector is less than zero (that is, a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

  The specified selector must be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

  The datatype of the item must be the same as the datatype implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

  The following special values can be specified for *Selector*:

  MQSEL_ANY_SELECTOR
  : The item to be inquired about is a user or system item identified by *ItemIndex*.

  MQSEL_ANY_USER_SELECTOR
  : The item to be inquired about is a user item identified by *ItemIndex*.

  MQSEL_ANY_SYSTEM_SELECTOR
  : The item to be inquired about is a system item identified by *ItemIndex*.

ItemIndex (MQLONG) – input
: Index of the data item to which the inquiry relates. The value must be zero or greater, or the special value MQIND_NONE. If the value is less than zero and not MQIND_NONE, MQRC_INDEX_ERROR results. If the item is not already present in the bag, MQRC_INDEX_NOT_PRESENT results. The following special value can be specified:

  MQIND_NONE
  : This specifies that there must be one occurrence only of the selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

    If MQSEL_ANY_SELECTOR is specified for the *Selector* parameter, *ItemIndex* is the index relative to the set of items that contains both user items and system items, and must be zero or greater.

    If MQSEL_ANY_USER_SELECTOR is specified for the *Selector* parameter, *ItemIndex* is the index relative to the set of user items, and must be zero or greater.

If MQSEL_ANY_SYSTEM_SELECTOR is specified for *Selector*, *ItemIndex* is the index relative to the set of system items, and must be zero or greater.

If an explicit selector value is specified, *ItemIndex* is the index relative to the set of items that have that selector value, and can be MQIND_NONE, zero, or greater.

BufferLength (MQLONG) – input
Length in bytes of the buffer to receive the string. Zero is a valid value.

Buffer (MQCHAR × *BufferLength*) – output
Buffer to receive the character string. The length is given by the *BufferLength* parameter. If zero is specified for *BufferLength*, the null pointer can be specified for the address of the *Buffer* parameter; in all other cases, a valid (nonnull) address must be specified for the *Buffer* parameter.

The string is padded with blanks to the length of the buffer; the string is not null-terminated. If the string is longer than the buffer, the string is truncated to fit; in this case *StringLength* indicates the size of the buffer needed to accommodate the string without truncation.

StringLength (MQLONG) – output
The length in bytes of the string contained in the bag. If the *Buffer* parameter is too small, the length of the string returned is less than *StringLength*.

CodedCharSetId (MQLONG) – output
The coded character set identifier for the character data in the string. This parameter can be set to a null pointer if not required.

CompCode (MQLONG) – output
Completion code.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

The following reason codes indicating error and warning conditions can be returned from the mqInquireString call:

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not completely accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE, or MQIND_NONE specified with one of the MQSEL_ANY_xxx_SELECTOR values).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong datatype for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_STRING_LENGTH_ERROR
StringLength parameter not valid (invalid parameter address).

MQRC_STRING_TRUNCATED
Data too long for output buffer and has been truncated.

C language invocation

```
mqInquireString (Bag, Selector, ItemIndex,
BufferLength, Buffer, &StringLength, &CodedCharSetId,
&CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;            /* Bag handle */
MQLONG   Selector;       /* Selector */
MQLONG   ItemIndex;      /* Item index */
MQLONG   BufferLength;   /* Buffer length */
PMQCHAR  Buffer;         /* Buffer to contain string */
MQLONG   StringLength;   /* Length of string returned */
MQLONG   CodedCharSetId  /* Coded Character Set ID */
MQLONG   CompCode;       /* Completion code */
MQLONG   Reason;         /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows only.)

```
mqInquireString Bag, Selector, ItemIndex,
BufferLength, Buffer, StringLength, CodedCharSetId,
CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag             As Long   'Bag handle'
Dim Selector        As Long   'Selector'
Dim ItemIndex       As Long   'Item index'
Dim BufferLength    As Long   'Buffer length'
Dim Buffer          As String 'Buffer to contain string'
Dim StringLength    As Long   'Length of string returned'
Dim CodedCharSetId  As Long   'Coded Character Set ID'
Dim CompCode        As Long   'Completion code'
Dim Reason          As Long   'Reason code qualifying CompCode'
```

mqPad

The mqPad call pads a null-terminated string with blanks.

Syntax

mqPad *(String, BufferLength, Buffer, CompCode, Reason)*

Parameters

String (PMQCHAR) – input
: Null-terminated string. The null pointer is valid for the address of the *String* parameter, and denotes a string of zero length.

BufferLength (MQLONG) – input
: Length in bytes of the buffer to receive the string padded with blanks. Must be zero or greater.

Buffer (MQCHAR × *BufferLength*) – output
: Buffer to receive the blank-padded string. The length is given by the *BufferLength* parameter. If zero is specified for *BufferLength*, the null pointer can be specified for the address of the *Buffer* parameter; in all other cases, a valid (nonnull) address must be specified for the *Buffer* parameter.

  If the number of characters preceding the first null in the *String* parameter is greater than the *BufferLength* parameter, the excess characters are omitted and MQRC_DATA_TRUNCATED results.

CompCode (MQLONG) – output
: Completion code.

Reason (MQLONG) – output
: Reason code qualifying *CompCode*.

  The following reason codes indicating error and warning conditions can be returned from the mqPad call:

  MQRC_BUFFER_ERROR
  Buffer parameter not valid (invalid parameter address or buffer not completely accessible).

  MQRC_BUFFER_LENGTH_ERROR
  Buffer length not valid.

  MQRC_STRING_ERROR
  String parameter not valid (invalid parameter address or buffer not completely accessible).

  MQRC_STRING_TRUNCATED
  Data too long for output buffer and has been truncated.

Usage notes

1. If the buffer pointers are the same, the padding is done in place. If not, at most *BufferLength* characters are copied into the second buffer; any space remaining, including the null-termination character, is overwritten with spaces.
2. If the *String* and *Buffer* parameters partially overlap, the result is undefined.

C language invocation

```
mqPad (String, BufferLength, Buffer, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQCHAR   String;          /* String to be padded */
MQLONG   BufferLength;    /* Buffer length */
PMQCHAR  Buffer           /* Buffer to contain padded string */
MQLONG   CompCode;        /* Completion code */
MQLONG   Reason;          /* Reason code qualifying CompCode */
```

Note: This call is not supported in Visual Basic.

mqPutBag

The mqPutBag call converts the contents of the specified bag into a PCF message and sends the message to the specified queue. The contents of the bag are unchanged after the call.

Syntax

mqPutBag *(Hconn, Hobj, MsgDesc, PutMsgOpts, Bag, CompCode, Reason)*

Parameters

Hconn (MQHCONN) – input
: MQI connection handle.

Hobj (MQHOBJ) – input
: Object handle of the queue on which the message is to be placed. This handle was returned by a preceding MQOPEN call issued by the application. The queue must be open for output.

MsgDesc (MQMD) – input/output
: Message descriptor. (For more information, see the *WebSphere MQ Application Programming Reference*.)

If the *Format* field has a value other than MQFMT_ADMIN, MQFMT_EVENT, or MQFMT_PCF, MQRC_FORMAT_NOT_SUPPORTED results.

If the *Encoding* field has a value other than MQENC_NATIVE, MQRC_ENCODING_NOT_SUPPORTED results.

PutMsgOpts (MQPMO) – input/output
: Put-message options. (For more information, see the *WebSphere MQ Application Programming Reference*.)

Bag (MQHBAG) – input
: Handle of the data bag to be converted to a message.

If the bag contains an administration message, and mqAddInquiry was used to insert values into the bag, the value of the MQIASY_COMMAND data item must be an INQUIRE command recognized by the MQAI; MQRC_INQUIRY_COMMAND_ERROR results if it is not.

If the bag contains nested bags, MQRC_NESTED_BAG_NOT_SUPPORTED results.

CompCode (MQLONG) – output
: Completion code.

Reason (MQLONG) – output
: Reason code qualifying *CompCode*. The following reason codes indicating error and warning conditions can be returned from the mqPutBag call:

MQRC_*
: Anything from the MQPUT call or bag manipulation.

MQRC_ENCODING_NOT_SUPPORTED
: Encoding not supported (value in *Encoding* field in MQMD must be MQENC_NATIVE).

MQRC_FORMAT_NOT_SUPPORTED
: Format not supported (name in *Format* field in MQMD must be MQFMT_ADMIN, MQFMT_EVENT, or MQFMT_PCF).

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INQUIRY_COMMAND_ERROR
mqAddInquiry call used with a command code that is not a recognized INQUIRE command.

MQRC_NESTED_BAG_NOT_SUPPORTED
Input data bag contains one or more nested bags.

MQRC_PARAMETER_MISSING
Administration message requires a parameter that is not present in the bag. This reason code occurs for bags created with the MQCBO_ADMIN_BAG or MQCBO_REORDER_AS_REQUIRED options only.

MQRC_SELECTOR_WRONG_TYPE
mqAddString or mqSetString was used to add the MQIACF_INQUIRY selector to the bag.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

C language invocation

```
mqPutBag (HConn, HObj, &MsgDesc, &PutMsgOpts, Bag,
&CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  HConn;        /* MQI connection handle */
MQHOBJ   HObj;         /* Object handle */
MQMD     MsgDesc;      /* Message descriptor */
MQPMO    PutMsgOpts;   /* Put-message options */
MQHBAG   Bag;          /* Bag handle */
MQLONG   CompCode;     /* Completion code */
MQLONG   Reason;       /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows only.)

```
mqPutBag (HConn, HObj, MsgDesc, PutMsgOpts, Bag,
CompCode, Reason);
```

Declare the parameters as follows:

```
Dim HConn      As Long  'MQI connection handle'
Dim HObj       As Long  'Object handle'
Dim MsgDesc    As MQMD  'Message descriptor'
Dim PutMsgOpts As MQPMO 'Put-message options'
Dim Bag        As Long  'Bag handle'
Dim CompCode   As Long  'Completion code'
Dim Reason     As Long  'Reason code qualifying CompCode'
```

mqSetInteger

The mqSetInteger call either modifies an integer item that is already present in the bag, or deletes all existing occurrences of the specified selector and adds a new occurrence at the end of the bag. The data item is usually a user item, but specific system-data items can also be modified.

Syntax

mqSetInteger *(Bag, Selector, ItemIndex, ItemValue, CompCode, Reason)*

Parameters

Bag (MQHBAG) – input
Handle of the bag to be set. This must be the handle of a bag created by the user, and not the handle of a system bag; MQRC_SYSTEM_BAG_NOT_ALTERABLE results if the handle you specify refers to a system bag.

Selector (MQLONG) – input
Selector of the item to be modified. If the selector is less than zero (that is, a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

If the selector is a supported system selector, but is one that is read-only, MQRC_SYSTEM_ITEM_NOT_ALTERABLE results.

If the selector is an alterable system selector, but is always a single-instance selector and the application attempts to create a second instance in the bag, MQRC_MULTIPLE_INSTANCE_ERROR results.

If the selector is zero or greater (that is, a user selector), and the bag was created with the MQCBO_CHECK_SELECTORS option or as an administration bag (MQCBO_ADMIN_BAG), the selector must be in the range MQIA_FIRST through MQIA_LAST; MQRC_SELECTOR_OUT_OF_RANGE results if it is not. If MQCBO_CHECK_SELECTORS was not specified, the selector can be any value zero or greater.

If MQIND_ALL is *not* specified for the *ItemIndex* parameter, the specified selector must already be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

If MQIND_ALL is *not* specified for the *ItemIndex* parameter, the datatype of the item must agree with the datatype implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

ItemIndex (MQLONG) – input
This value identifies the occurrence of the item with the specified selector that is to be modified. The value must be zero or greater, or one of the special values described below; if it is none of these, MQRC_INDEX_ERROR results.

Zero or greater
The item with the specified index must already be present in the bag; MQRC_INDEX_NOT_PRESENT results if it is not. The index is counted relative to the items in the bag that have the specified selector. For example, if there are five items in the bag with the specified selector, the valid values for *ItemIndex* are 0 through 4.

MQIND_NONE
This specifies that there must be one occurrence only of the specified

selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

MQIND_ALL
This specifies that all existing occurrences of the specified selector (if any) are to be deleted from the bag, and a new occurrence of the selector created at the end of the bag.

Note: For system selectors, the order is not changed.

ItemValue (MQLONG) – input
The integer value to be placed in the bag.

CompCode (MQLONG) – output
Completion code.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

The following reason codes indicating error and warning conditions can be returned from the mqSetInteger call:

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE or MQIND_ALL).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_MULTIPLE_INSTANCE_ERROR
Multiple instances of system selector not valid.

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not in valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong datatype for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

MQRC_SYSTEM_ITEM_NOT_ALTERABLE
System item is read only and cannot be altered.

C language invocation

```
mqSetInteger (Bag, Selector, ItemIndex, ItemValue, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;          /* Bag handle */
MQLONG   Selector;     /* Selector */
MQLONG   ItemIndex;    /* Item index */
MQLONG   ItemValue;    /* Integer value */
MQLONG   CompCode;     /* Completion code */
MQLONG   Reason;       /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows only.)

```
mqSetInteger Bag, Selector, ItemIndex, ItemValue, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag       As Long 'Bag handle'
Dim Selector  As Long 'Selector'
Dim ItemIndex As Long 'Item index'
Dim ItemValue As Long 'Integer value'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

mqSetString

The mqSetString call either modifies a character data item that is already present in the bag, or deletes all existing occurrences of the specified selector and adds a new occurrence at the end of the bag. The data item is usually a user item, but certain system-data items can also be modified.

Syntax

mqSetString *(Bag, Selector, ItemIndex, Bufferlength, Buffer, CompCode, Reason)*

Parameters

Bag (MQHBAG) – input
: Handle of the bag to be set. This must be the handle of a bag created by the user, not the handle of a system bag; MQRC_SYSTEM_BAG_NOT_ALTERABLE results if you specify the handle of a system bag.

Selector (MQLONG) – input
: Selector of the item to be modified.

If the selector is less than zero (that is, a system selector), the selector must be one that is supported by the MQAI; MQRC_SELECTOR_NOT_SUPPORTED results if it is not.

If the selector is a supported system selector, but is one that is read only, MQRC_SYSTEM_ITEM_NOT_ALTERABLE results.

If the selector is an alterable system selector, but is always a single-instance selector and the application attempts to create a second instance in the bag, MQRC_MULTIPLE_INSTANCE_ERROR results.

If the selector is zero or greater (that is, a user selector), and the bag was created with the MQCBO_CHECK_SELECTORS option or as an administration bag (MQCBO_ADMIN_BAG), the selector must be in the range MQCA_FIRST through MQCA_LAST; MQRC_SELECTOR_OUT_OF_RANGE results if it is not. If MQCBO_CHECK_SELECTORS was not specified, the selector can be any value zero or greater.

If MQIND_ALL is *not* specified for the *ItemIndex* parameter, the specified selector must already be present in the bag; MQRC_SELECTOR_NOT_PRESENT results if it is not.

If MQIND_ALL is *not* specified for the *ItemIndex* parameter, the datatype of the item must be the same as the datatype implied by the call; MQRC_SELECTOR_WRONG_TYPE results if it is not.

ItemIndex (MQLONG) – input
: This identifies which occurrence of the item with the specified selector is to be modified. The value must be zero or greater, or one of the special values described below; if it is none of these, MQRC_INDEX_ERROR results.

Zero or greater
: The item with the specified index must already be present in the bag; MQRC_INDEX_NOT_PRESENT results if it is not. The index is counted relative to the items in the bag that have the specified selector. For example, if there are five items in the bag with the specified selector, the valid values for *ItemIndex* are 0 through 4.

MQIND_NONE
This specifies that there must be only one occurrence of the specified selector in the bag. If there is more than one occurrence, MQRC_SELECTOR_NOT_UNIQUE results.

MQIND_ALL
This specifies that all existing occurrences of the specified selector (if any) are to be deleted from the bag, and a new occurrence of the selector created at the end of the bag.

BufferLength (MQLONG) – input
The length in bytes of the string contained in the *Buffer* parameter. The value must be zero or greater, or the special value MQBL_NULL_TERMINATED.

If MQBL_NULL_TERMINATED is specified, the string is delimited by the first null encountered in the string.

If MQBL_NULL_TERMINATED is not specified, *BufferLength* characters are inserted into the bag, even if null characters are present; the nulls do not delimit the string.

Buffer (MQCHAR × *BufferLength*) – input
Buffer containing the character string. The length is given by the *BufferLength* parameter. If zero is specified for *BufferLength*, the null pointer can be specified for the address of the *Buffer* parameter; in all other cases, a valid (nonnull) address must be specified for the *Buffer* parameter.

CompCode (MQLONG) – output
Completion code.

Reason (MQLONG) – output
Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqSetString call:

MQRC_BUFFER_ERROR
Buffer parameter not valid (invalid parameter address or buffer not completely accessible).

MQRC_BUFFER_LENGTH_ERROR
Buffer length not valid.

MQRC_HBAG_ERROR
Bag handle not valid.

MQRC_INDEX_ERROR
Index not valid (index negative and not MQIND_NONE or MQIND_ALL).

MQRC_INDEX_NOT_PRESENT
No item with the specified index is present within the bag for the selector given.

MQRC_MULTIPLE_INSTANCE_ERROR
Multiple instances of system selector not valid.

MQRC_SELECTOR_NOT_PRESENT
No item with the specified selector is present within the bag.

MQRC_SELECTOR_NOT_SUPPORTED
Specified system selector not supported by the MQAI.

MQRC_SELECTOR_NOT_UNIQUE
MQIND_NONE specified when more than one occurrence of the specified selector is present in the bag.

MQRC_SELECTOR_OUT_OF_RANGE
Selector not within valid range for call.

MQRC_SELECTOR_WRONG_TYPE
Data item has wrong datatype for call.

MQRC_STORAGE_NOT_AVAILABLE
Insufficient storage available.

MQRC_SYSTEM_BAG_NOT_ALTERABLE
System bag cannot be altered or deleted.

MQRC_SYSTEM_ITEM_NOT_ALTERABLE
System item is read-only and cannot be altered.

Usage notes

The Coded Character Set ID (CCSID) associated with this string is copied from the current CCSID of the bag.

C language invocation

```
mqSetString (Bag, Selector, ItemIndex, BufferLength, Buffer,
&CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   Bag;           /* Bag handle */
MQLONG   Selector;      /* Selector */
MQLONG   ItemIndex;     /* Item index */
MQLONG   BufferLength;  /* Buffer length */
PMQCHAR  Buffer;        /* Buffer containing string */
MQLONG  CompCode;       /* Completion code */
MQLONG  Reason;         /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows only.)

```
mqSetString Bag, Selector, ItemIndex, BufferLength, Buffer,
CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag          As Long   'Bag handle'
Dim Selector     As Long   'Selector'
Dim ItemIndex    As Long   'Item index'
Dim BufferLength As Long   'Buffer length'
Dim Buffer       As String 'Buffer containing string'
Dim CompCode     As Long   'Completion code'
Dim Reason       As Long   'Reason code qualifying CompCode'
```

mqTrim

The mqTrim call trims the blanks from a blank-padded string, then terminates it with a null.

Syntax

mqTrim *(BufferLength, Buffer, String, CompCode, Reason)*

Parameters

BufferLength (MQLONG) – input
: Length in bytes of the buffer containing the string padded with blanks. Must be zero or greater.

Buffer (MQCHAR × *BufferLength*) – input
: Buffer containing the blank-padded string. The length is given by the *BufferLength* parameter. If zero is specified for *BufferLength*, the null pointer can be specified for the address of the *Buffer* parameter; in all other cases, a valid (nonnull) address must be specified for the *Buffer* parameter.

String (MQCHAR × (*BufferLength*+1)) – output
: Buffer to receive the null-terminated string. The length of this buffer must be at least one byte greater than the value of the *BufferLength* parameter.

CompCode (MQLONG) – output
: Completion code.

Reason (MQLONG) – output
: Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqTrim call:

MQRC_BUFFER_ERROR
: Buffer parameter not valid (invalid parameter address or buffer not completely accessible).

MQRC_BUFFER_LENGTH_ERROR
: Buffer length not valid.

MQRC_STRING_ERROR
: String parameter not valid (invalid parameter address or buffer not completely accessible).

Usage notes

1. If the two buffer pointers are the same, the trimming is done in place. If they are not the same, the blank-padded string is copied into the null-terminated string buffer. After copying, the buffer is scanned backwards from the end until a nonspace character is found. The byte following the nonspace character is then overwritten with a null character.
2. If *String* and *Buffer* partially overlap, the result is undefined.

C language invocation

```
mqTrim (BufferLength, Buffer, String, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQLONG   BufferLength;     /* Buffer length */
PMQCHAR  Buffer;           /* Buffer containing blank-padded string */
MQCHAR   String[n+1];      /* String with blanks discarded */
MQLONG   CompCode;         /* Completion code */
MQLONG   Reason;           /* Reason code qualifying CompCode */
```

Note: This call is not supported in Visual Basic.

mqTruncateBag

The mqTruncateBag call reduces the number of user items in a user bag to the specified value, by deleting user items from the end of the bag.

Syntax

mqTruncateBag *(Bag, ItemCount, CompCode, Reason)*

Parameters

Bag (MQHBAG) – input
: Handle of the bag to be truncated. This must be the handle of a bag created by the user, not the handle of a system bag; MQRC_SYSTEM_BAG_NOT_ALTERABLE results if you specify the handle of a system bag.

ItemCount (MQLONG) – input
: The number of user items to remain in the bag after truncation. Zero is a valid value.

Note: The *ItemCount* parameter is the number of data items, not the number of unique selectors. (If there are one or more selectors that occur multiple times in the bag, there will be fewer selectors than data items before truncation.) Data items are deleted from the end of the bag, in the opposite order to which they were added to the bag.

If the number specified exceeds the number of user items currently in the bag, MQRC_ITEM_COUNT_ERROR results.

CompCode (MQLONG) – output
: Completion code.

Reason (MQLONG) – output
: Reason code qualifying *CompCode*.

The following reason codes indicating error conditions can be returned from the mqTruncateBag call:

MQRC_HBAG_ERROR
: Bag handle not valid.

MQRC_ITEM_COUNT_ERROR
: *ItemCount* parameter not valid (value exceeds the number of user data items in the bag).

MQRC_SYSTEM_BAG_NOT_ALTERABLE
: System bag cannot be altered or deleted.

Usage notes

1. System items in a bag are not affected by mqTruncateBag; the call cannot be used to truncate system bags.
2. mqTruncateBag with an *ItemCount* of zero is not the same as the mqClearBag call. The former deletes all of the user items but leaves the system items intact, and the latter deletes all of the user items and resets the system items to their initial values.

C language invocation

```
mqTruncateBag (Bag, ItemCount, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHBAG   hBag;          /* Bag handle */
MQLONG   ItemCount;     /* Number of items to remain in bag */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

Visual Basic invocation

(Supported on Windows only.)

```
mqTruncateBag Bag, ItemCount, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Bag       As Long 'Bag handle'
Dim ItemCount As Long 'Number of items to remain in bag'
Dim CompCode  As Long 'Completion code'
Dim Reason    As Long 'Reason code qualifying CompCode'
```

Chapter 18. Examples of using the MQAI

This chapter includes some example programs that demonstrate use of the MQAI. The samples perform the following tasks:
1. Create a local queue.
2. Print a list of all local queues and their current depths.
3. Display events on the screen using a simple event monitor.

Creating a local queue (amqsaicq.c)

```
/******************************************************************************/
/*                                                                            */
/* Program name: AMQSAICQ.C                                                   */
/*                                                                            */
/* Description:  Sample C program to create a local queue using the          */
/*               WebSphere MQ Administration Interface (MQAI).               */
/*                                                                            */
/* Statement:    Licensed Materials - Property of IBM                         */
/*                                                                            */
/*               84H2000, 5765-B73                                            */
/*               84H2001, 5639-B42                                            */
/*               84H2002, 5765-B74                                            */
/*               84H2003, 5765-B75                                            */
/*               84H2004, 5639-B43                                            */
/*                                                                            */
/*               (C) Copyright IBM Corp. 1999                                 */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* Function:                                                                  */
/*    AMQSAICQ is a sample C program that creates a local queue and is an     */
/*    example of the use of the mqExecute call.                               */
/*                                                                            */
/*     - The name of the queue to be created is a parameter to the program.   */
/*                                                                            */
/*     - A PCF command is built by placing items into an MQAI bag.            */
/*       These are:-                                                          */
/*            - The name of the queue                                         */
/*            - The type of queue required, which, in this case, is local.    */
/*                                                                            */
/*     - The mqExecute call is executed with the command MQCMD_CREATE_Q.      */
/*       The call generates the correct PCF structure.                        */
/*       The call receives the reply from the command server and formats into */
/*       the response bag.                                                    */
/*                                                                            */
/*     - The completion code from the mqExecute call is checked and if there  */
/*       is a failure from the command server then the code returned by the   */
/*       command server is retrieved from the system bag that is              */
/*       embedded in the response bag to the mqExecute call.                  */
/*                                                                            */
/* Note: The command server must be running.                                  */
/*                                                                            */
/*                                                                            */
```

Figure 14. AMQSAICQ.C: Creating a local queue (Part 1 of 6)

Creating a local queue

```
/******************************************************************************/
/*                                                                            */
/* AMQSAICQ has 2 parameters - the name of the local queue to be created      */
/*                           - the queue manager name (optional)              */
/*                                                                            */
/******************************************************************************/
/******************************************************************************/
/* Includes                                                                   */
/******************************************************************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

#include <cmqc.h>                          /* MQI                             */
#include <cmqcfc.h>                        /* PCF                             */
#include <cmqbc.h>                         /* MQAI                            */

void CheckCallResult(MQCHAR *, MQLONG , MQLONG );
void CreateLocalQueue(MQHCONN, MQCHAR *);

int main(int argc, char *argv[])
{
   MQHCONN hConn;                          /* handle to WebSphere MQ connection   */
   MQCHAR QMName[MQ_Q_MGR_NAME_LENGTH+1]=""; /* default QMgr name   */
   MQLONG connReason;                      /* MQCONN reason code              */
   MQLONG compCode;                        /* completion code                 */
   MQLONG reason;                          /* reason code                     */

   /***************************************************************************/
   /* First check the required parameters                                     */
   /***************************************************************************/
   printf("Sample Program to Create a Local Queue\n");
   if (argc < 2)
   {
     printf("Required parameter missing - local queue name\n");
     exit(99);
   }

   /***************************************************************************/
   /* Connect to the queue manager                                            */
   /***************************************************************************/
   if (argc > 2)
      strncpy(QMName, argv[2], (size_t)MQ_Q_MGR_NAME_LENGTH);
      MQCONN(QMName, &hConn, &compCode, &connReason);

/******************************************************************************/
/* Report reason and stop if connection failed                                */
/******************************************************************************/
   if (compCode == MQCC_FAILED)
   {
      CheckCallResult("MQCONN", compCode, connReason);
      exit( (int)connReason);
   }
```

Figure 14. AMQSAICQ.C: Creating a local queue (Part 2 of 6)

```
/******************************************************************************/
/* Call the routine to create a local queue, passing the handle to the       */
/* queue manager and also passing the name of the queue to be created.       */
/******************************************************************************/
   CreateLocalQueue(hConn, argv[1]);

   /***************************************************************************/
   /* Disconnect from the queue manager if not already connected             */
   /***************************************************************************/
   if (connReason != MQRC_ALREADY_CONNECTED)
   {
      MQDISC(&hConn, &compCode, &reason);
      CheckCallResult("MQDISC", compCode, reason);
   }
   return 0;

}
/******************************************************************************/
/*                                                                            */
/* Function:    CreateLocalQueue                                              */
/* Description: Create a local queue by sending a PCF command to the command  */
/*              server.                                                       */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* Input Parameters:  Handle to the queue manager                             */
/*                    Name of the queue to be created                         */
/*                                                                            */
/* Output Parameters: None                                                    */
/*                                                                            */
/* Logic: The mqExecute call is executed with the command MQCMD_CREATE_Q.     */
/*        The call generates the correct PCF structure.                       */
/*        The default options to the call are used so that the command is sent*/
/*        to the SYSTEM.ADMIN.COMMAND.QUEUE.                                  */
/*        The reply from the command server is placed on a temporary dynamic  */
/*        queue.                                                              */
/*        The reply is read from the temporary queue and formatted into the   */
/*        response bag.                                                       */
/*                                                                            */
/*        The completion code from the mqExecute call is checked and if there */
/*        is a failure from the command server then the code returned by the  */
/*        command server is retrieved from the system bag that is             */
/*        embedded in the response bag to the mqExecute call.                 */
/*                                                                            */
/******************************************************************************/
void CreateLocalQueue(MQHCONN hConn, MQCHAR *qName)
{
   MQLONG reason;                          /* reason code                     */
   MQLONG compCode;                        /* completion code                 */
   MQHBAG commandBag = MQHB_UNUSABLE_HBAG; /* command bag for mqExecute       */
   MQHBAG responseBag = MQHB_UNUSABLE_HBAG;/* response bag for mqExecute      */
   MQHBAG resultBag;                       /* result bag from mqExecute       */
   MQLONG mqExecuteCC;                     /* mqExecute completion code       */
   MQLONG mqExecuteRC;                     /* mqExecute reason code           */

   printf("\nCreating Local Queue %s\n\n", qName);
```

Figure 14. AMQSAICQ.C: Creating a local queue (Part 3 of 6)

Creating a local queue

```
/**************************************************************************/
/* Create a command Bag for the mqExecute call. Exit the function if the   */
/* create fails.                                                           */
/**************************************************************************/
mqCreateBag(MQCBO_ADMIN_BAG, &commandBag, &compCode, &reason);
CheckCallResult("Create the command bag", compCode, reason);
if (compCode !=MQCC_OK)
   return;

/**************************************************************************/
/* Create a response Bag for the mqExecute call, exit the function if the  */
/* create fails.                                                           */
/**************************************************************************/
mqCreateBag(MQCBO_ADMIN_BAG, &responseBag, &compCode, &reason);
CheckCallResult("Create the response bag", compCode, reason);
if (compCode !=MQCC_OK)
   return;

/**************************************************************************/
/* Put the name of the queue to be created into the command bag. This will */
/* be used by the mqExecute call.                                          */
/**************************************************************************/
mqAddString(commandBag, MQCA_Q_NAME, MQBL_NULL_TERMINATED, qName, &compCode,
            &reason);
CheckCallResult("Add q name to command bag", compCode, reason);

/**************************************************************************/
/* Put queue type of local into the command bag. This will be used by the  */
/* mqExecute call.                                                         */
/**************************************************************************/
mqAddInteger(commandBag, MQIA_Q_TYPE, MQQT_LOCAL, &compCode, &reason);
CheckCallResult("Add q type to command bag", compCode, reason);

/**************************************************************************/
/* Send the command to create the required local queue.                    */
/* The mqExecute call will create the PCF structure required, send it to   */
/* the command server and receive the reply from the command server into   */
/* the response bag.                                                       */
/**************************************************************************/
mqExecute(hConn,                  /* WebSphere MQ connection handle           */
          MQCMD_CREATE_Q,         /* Command to be executed              */
          MQHB_NONE,              /* No options bag                      */
          commandBag,             /* Handle to bag containing commands   */
          responseBag,            /* Handle to bag to receive the response*/
          MQHO_NONE,              /* Put msg on SYSTEM.ADMIN.COMMAND.QUEUE*/
          MQHO_NONE,              /* Create a dynamic q for the response  */
          &compCode,              /* Completion code from the mqExecute   */
          &reason);               /* Reason code from mqExecute call      */

if (reason == MQRC_CMD_SERVER_NOT_AVAILABLE)
{
   printf("Please start the command server: <strmqcsv QMgrName>\n")
   MQDISC(&hConn, &compCode, &reason);
   CheckCallResult("MQDISC", compCode, reason);
   exit(98);
}
```

Figure 14. AMQSAICQ.C: Creating a local queue (Part 4 of 6)

```
   /***************************************************************************/
   /* Check the result from mqExecute call and find the error if it failed.   */
   /***************************************************************************/
   if ( compCode == MQCC_OK )
      printf("Local queue %s successfully created\n", qName);
   else
   {
      printf("Creation of local queue %s failed: Completion Code = %d
               qName, compCode, reason);
      if (reason == MQRCCF_COMMAND_FAILED)
      {
         /*********************************************************************/
         /* Get the system bag handle out of the mqExecute response bag.      */
         /* This bag contains the reason from the command server why the      */
         /* command failed.                                                   */
         /*********************************************************************/
         mqInquireBag(responseBag, MQHA_BAG_HANDLE, 0, &resultBag, &compCode,
                      &reason);
         CheckCallResult("Get the result bag handle", compCode, reason);

         /*********************************************************************/
         /* Get the completion code and reason code, returned by the command  */
         /* server, from the embedded error bag.                              */
         /*********************************************************************/
         mqInquireInteger(resultBag, MQIASY_COMP_CODE, MQIND_NONE, &mqExecuteCC,
                          &compCode, &reason);
         CheckCallResult("Get the completion code from the result bag",
                          compCode, reason);
         mqInquireInteger(resultBag, MQIASY_REASON, MQIND_NONE, &mqExecuteRC,
                          &compCode, &reason);
         CheckCallResult("Get the reason code from the result bag", compCode,
                          reason);
         printf("Error returned by the command server: Completion code = %d :
                 Reason = %d\n", mqExecuteCC, mqExecuteRC);
      }
   }
   /***************************************************************************/
   /* Delete the command bag if successfully created.                         */
   /***************************************************************************/
   if (commandBag != MQHB_UNUSABLE_HBAG)
   {
      mqDeleteBag(&commandBag, &compCode, &reason);
      CheckCallResult("Delete the command bag", compCode, reason);
   }

   /***************************************************************************/
   /* Delete the response bag if successfully created.                        */
   /***************************************************************************/
   if (responseBag != MQHB_UNUSABLE_HBAG)
   {
      mqDeleteBag(&responseBag, &compCode, &reason);
      CheckCallResult("Delete the response bag", compCode, reason);
   }
} /* end of CreateLocalQueue */
```

Figure 14. AMQSAICQ.C: Creating a local queue (Part 5 of 6)

```
/******************************************************************************/
/*                                                                            */
/* Function: CheckCallResult                                                  */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* Input Parameters:  Description of call                                     */
/*                    Completion code                                         */
/*                    Reason code                                             */
/*                                                                            */
/* Output Parameters: None                                                    */
/*                                                                            */
/* Logic: Display the description of the call, the completion code and the    */
/*        reason code if the completion code is not successful                */
/*                                                                            */
/******************************************************************************/
void  CheckCallResult(char *callText, MQLONG cc, MQLONG rc)
{
   if (cc != MQCC_OK)
         printf("%s failed: Completion Code = %d :
                 Reason = %d\n", callText, cc, rc);

}
```

Figure 14. AMQSAICQ.C: Creating a local queue (Part 6 of 6)

Inquiring about queues and printing information (amqsailq.c)

```
/******************************************************************************/
/*                                                                            */
/* Program name: AMQSAILQ.C                                                   */
/*                                                                            */
/* Description:  Sample C program to inquire the current depth of the local   */
/*               queues using the WebSphere MQ Administration Interface (MQAI)*/
/*                                                                            */
/* Statement:    Licensed Materials - Property of IBM                         */
/*                                                                            */
/*               84H2000, 5765-B73                                            */
/*               84H2001, 5639-B42                                            */
/*               84H2002, 5765-B74                                            */
/*               84H2003, 5765-B75                                            */
/*               84H2004, 5639-B43                                            */
/*                                                                            */
/*               (C) Copyright IBM Corp. 1999                                 */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* Function:                                                                  */
/*    AMQSAILQ is a sample C program that demonstrates how to inquire         */
/*    attributes of the local queue manager using the MQAI interface. In      */
/*    particular, it inquires the current depths of all the local queues.     */
/*                                                                            */
/*     - A PCF command is built by placing items into an MQAI administration  */
/*       bag.                                                                 */
/*       These are:-                                                          */
/*            - The generic queue name "*"                                    */
/*            - The type of queue required. In this sample we want to         */
/*              inquire local queues.                                         */
/*            - The attribute to be inquired. In this sample we want the      */
/*              current depths.                                               */
/*                                                                            */
/*     - The mqExecute call is executed with the command MQCMD_INQUIRE_Q.     */
/*       The call generates the correct PCF structure.                        */
/*       The default options to the call are used so that the command is sent */
/*       to the SYSTEM.ADMIN.COMMAND.QUEUE.                                   */
/*       The reply from the command server is placed on a temporary dynamic   */
/*       queue.                                                               */
/*       The reply from the MQCMD_INQUIRE_Q command is read from the          */
/*       temporary queue and formatted into the response bag.                 */
/*                                                                            */
/*     - The completion code from the mqExecute call is checked and if there  */
/*       is a failure from the command server, then the code returned by      */
/*       command server is retrieved from the system bag that has been        */
/*       embedded in the response bag to the mqExecute call.                  */
/*                                                                            */
/*     - If the call is successful, the depth of each local queue is placed   */
/*       in system bags embedded in the response bag of the mqExecute call.   */
/*       The name and depth of each queue is obtained from each of the bags   */
/*       and the result displayed on the screen.                              */
/*                                                                            */
/* Note: The command server must be running.                                  */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* AMQSAILQ has 1 parameter - the queue manager name (optional)               */
/*                                                                            */
/******************************************************************************/
```

Figure 15. AMQSAILQ.C: Inquiring on queues and printing information (Part 1 of 6)

```
/******************************************************************************/
/* Includes                                                                   */
/******************************************************************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>

#include <cmqc.h>                          /* MQI                             */
#include <cmqcfc.h>                        /* PCF                             */
#include <cmqbc.h>                         /* MQAI                            */

/******************************************************************************/
/* Function prototypes                                                        */
/******************************************************************************/
void CheckCallResult(MQCHAR *, MQLONG , MQLONG);

/******************************************************************************/
/* Function: main                                                             */
/******************************************************************************/
int main(int argc, char *argv[])
{
   /***************************************************************************/
   /* MQAI variables                                                          */
   /***************************************************************************/
   MQHCONN hConn;                          /* handle to WebSphere MQ connection   */
   MQCHAR qmName[MQ_Q_MGR_NAME_LENGTH+1]=""; /* default QMgr name             */
   MQLONG reason;                          /* reason code                     */
   MQLONG connReason;                      /* MQCONN reason code              */
   MQLONG compCode;                        /* completion code                 */
   MQHBAG adminBag = MQHB_UNUSABLE_HBAG;   /* admin bag for mqExecute         */
   MQHBAG responseBag = MQHB_UNUSABLE_HBAG;/* response bag for mqExecute      */
   MQHBAG qAttrsBag;                       /* bag containing q attributes     */
   MQHBAG errorBag;                        /* bag containing cmd server error */
   MQLONG mqExecuteCC;                     /* mqExecute completion code       */
   MQLONG mqExecuteRC;                     /* mqExecute reason code           */
   MQLONG qNameLength;                     /* Actual length of q name         */
   MQLONG qDepth;                          /* depth of queue                  */
   MQLONG i;                               /* loop counter                    */
   MQLONG numberOfBags;                    /* number of bags in response bag  */
   MQCHAR qName[MQ_Q_NAME_LENGTH+1];       /* name of queue extracted from bag*/


   printf("Display current depths of local queues\n\n");

   /***************************************************************************/
   /* Connect to the queue manager                                            */
   /***************************************************************************/
   if (argc > 1)
      strncpy(qmName, argv[1], (size_t)MQ_Q_MGR_NAME_LENGTH);
   MQCONN(qmName, &hConn, &compCode, &connReason);

   /***************************************************************************/
   /* Report the reason and stop if the connection failed.                    */
   /***************************************************************************/
   if (compCode == MQCC_FAILED)
   {
      CheckCallResult("Queue Manager connection", compCode, connReason
      exit( (int)connReason);
   }
```

Figure 15. AMQSAILQ.C: Inquiring on queues and printing information (Part 2 of 6)

```
/**************************************************************************/
/* Create an admin bag for the mqExecute call                             */
/**************************************************************************/
mqCreateBag(MQCBO_ADMIN_BAG, &adminBag, &compCode, &reason);
CheckCallResult("Create admin bag", compCode, reason);
/**************************************************************************/
/* Create a response bag for the mqExecute call                           */
/**************************************************************************/
mqCreateBag(MQCBO_ADMIN_BAG, &responseBag, &compCode, &reason);
CheckCallResult("Create response bag", compCode, reason);

/**************************************************************************/
/* Put the generic queue name into the admin bag                          */
/**************************************************************************/
mqAddString(adminBag, MQCA_Q_NAME, MQBL_NULL_TERMINATED, "*",
            &compCode, &reason);
CheckCallResult("Add q name", compCode, reason);

/**************************************************************************/
/* Put the local queue type into the admin bag                            */
/**************************************************************************/
mqAddInteger(adminBag, MQIA_Q_TYPE, MQQT_LOCAL, &compCode, &reason);
CheckCallResult("Add q type", compCode, reason);

/**************************************************************************/
/* Add an inquiry for current queue depths                                */
/**************************************************************************/
mqAddInquiry(adminBag, MQIA_CURRENT_Q_DEPTH, &compCode, &reason);
CheckCallResult("Add inquiry", compCode, reason);

/**************************************************************************/
/* Send the command to find all the local queue names and queue depths.   */
/* The mqExecute call creates the PCF structure required, sends it to     */
/* the command server, and receives the reply from the command server into */
/* the response bag. The attributes are contained in system bags that are  */
/* embedded in the response bag, one set of attributes per bag.           */
/**************************************************************************/
mqExecute(hConn,                   /* WebSphere MQ connection handle          */
          MQCMD_INQUIRE_Q,         /* Command to be executed              */
          MQHB_NONE,               /* No options bag                      */
          adminBag,                /* Handle to bag containing commands   */
          responseBag,             /* Handle to bag to receive the response*/
          MQHO_NONE,               /* Put msg on SYSTEM.ADMIN.COMMAND.QUEUE*/
          MQHO_NONE,               /* Create a dynamic q for the response */
          &compCode,               /* Completion code from the mqExecute  */
          &reason);                /* Reason code from mqExecute call     */


/**************************************************************************/
/* Check the command server is started. If not exit.                      */
/**************************************************************************/
if (reason == MQRC_CMD_SERVER_NOT_AVAILABLE)
{
   printf("Please start the command server: <strmqcsv QMgrName>\n");
   MQDISC(&hConn, &compCode, &reason);
   CheckCallResult("Disconnect from Queue Manager", compCode, reason);
   exit(98);
}
```

Figure 15. AMQSAILQ.C: Inquiring on queues and printing information (Part 3 of 6)

```
/******************************************************************************/
/* Check the result from mqExecute call. If successful find the current    */
/* depths of all the local queues. If failed find the error.               */
/******************************************************************************/
if ( compCode == MQCC_OK )                      /* Successful mqExecute    */
{
  /****************************************************************************/
  /* Count the number of system bags embedded in the response bag from the */
  /* mqExecute call. The attributes for each queue are in a separate bag.  */
  /****************************************************************************/
  mqCountItems(responseBag, MQHA_BAG_HANDLE, &numberOfBags, &compCode,
               &reason);
  CheckCallResult("Count number of bag handles", compCode, reason);

  for ( i=0; i<numberOfBags; i++)
  {
    /**************************************************************************/
    /* Get the next system bag handle out of the mqExecute response bag.   */
    /* This bag contains the queue attributes                              */
    /**************************************************************************/
    mqInquireBag(responseBag, MQHA_BAG_HANDLE, i, &qAttrsBag, &compCode,
                 &reason);
    CheckCallResult("Get the result bag handle", compCode, reason);

    /**************************************************************************/
    /* Get the queue name out of the queue attributes bag                  */
    /**************************************************************************/
    mqInquireString(qAttrsBag, MQCA_Q_NAME, 0, MQ_Q_NAME_LENGTH, qName,
                    &qNameLength, NULL, &compCode, &reason);
    CheckCallResult("Get queue name", compCode, reason);

    /**************************************************************************/
    /* Get the depth out of the queue attributes bag                       */
    /**************************************************************************/
    mqInquireInteger(qAttrsBag, MQIA_CURRENT_Q_DEPTH, MQIND_NONE, &qDepth,
                     &compCode, &reason);
    CheckCallResult("Get depth", compCode, reason);

    /**************************************************************************/
    /* Use mqTrim to prepare the queue name for printing.                  */
    /* Print the result.                                                   */
    /**************************************************************************/
    mqTrim(MQ_Q_NAME_LENGTH, qName, qName, &compCode, &reason)
    printf("%4d  %-48s\n", qDepth, qName);
  }
}

else                                               /* Failed mqExecute     */
{
  printf("Call to get queue attributes failed: Completion Code = %d :
          Reason = %d\n", compCode, reason);
```

Figure 15. AMQSAILQ.C: Inquiring on queues and printing information (Part 4 of 6)

```
      /*********************************************************************/
      /* If the command fails get the system bag handle out of the mqExecute   */
      /* response bag. This bag contains the reason from the command server    */
      /* why the command failed.                                              */
      /*********************************************************************/
      if (reason == MQRCCF_COMMAND_FAILED)
      {
        mqInquireBag(responseBag, MQHA_BAG_HANDLE, 0, &errorBag, &compCode,
                     &reason);
        CheckCallResult("Get the result bag handle", compCode, reason);

       /*********************************************************************/
       /* Get the completion code and reason code, returned by the command     */
       /* server, from the embedded error bag.                                */
       /*********************************************************************/
       mqInquireInteger(errorBag, MQIASY_COMP_CODE, MQIND_NONE, &mqExecuteCC,
                        &compCode, &reason );
       CheckCallResult("Get the completion code from the result bag",
                        compCode, reason);
       mqInquireInteger(errorBag, MQIASY_REASON, MQIND_NONE, &mqExecuteRC,
                         &compCode, &reason);
       CheckCallResult("Get the reason code from the result bag",
                        compCode, reason);
       printf("Error returned by the command server: Completion Code = %d :
               Reason = %d\n", mqExecuteCC, mqExecuteRC);
     }
  }

  /***************************************************************************/
  /* Delete the admin bag if successfully created.                           */
  /***************************************************************************/
  if (adminBag != MQHB_UNUSABLE_HBAG)
  {
     mqDeleteBag(&adminBag, &compCode, &reason);
     CheckCallResult("Delete the admin bag", compCode, reason);
  }

  /***************************************************************************/
  /* Delete the response bag if successfully created.                        */
  /***************************************************************************/
  if (responseBag != MQHB_UNUSABLE_HBAG)
  {
     mqDeleteBag(&responseBag, &compCode, &reason);
     CheckCallResult("Delete the response bag", compCode, reason);
  }

  /***************************************************************************/
  /* Disconnect from the queue manager if not already connected             */
  /***************************************************************************/
  if (connReason != MQRC_ALREADY_CONNECTED)
  {
     MQDISC(&hConn, &compCode, &reason);
      CheckCallResult("Disconnect from queue manager", compCode, reason);
  }
  return 0;
}
```

Figure 15. AMQSAILQ.C: Inquiring on queues and printing information (Part 5 of 6)

```
******************************************************************************/
*                                                                            */
* Function: CheckCallResult                                                  */
*                                                                            */
******************************************************************************/
*                                                                            */
* Input Parameters:  Description of call                                     */
*                    Completion code                                         */
*                    Reason code                                             */
*                                                                            */
* Output Parameters: None                                                    */
*                                                                            */
* Logic: Display the description of the call, the completion code and the    */
*        reason code if the completion code is not successful                */
*                                                                            */
******************************************************************************/
void  CheckCallResult(char *callText, MQLONG cc, MQLONG rc)
{
  if (cc != MQCC_OK)
        printf("%s failed: Completion Code = %d : Reason = %d\n",
                callText, cc, rc);
}
```

Figure 15. AMQSAILQ.C: Inquiring on queues and printing information (Part 6 of 6)

Displaying events using an event monitor (amqsaiem.c)

```
/******************************************************************************/
/*                                                                            */
/* Program name: AMQSAIEM.C                                                   */
/*                                                                            */
/* Description:  Sample C program to demonstrate a basic event monitor       */
/*               using the WebSphere MQ Administration Interface (MQAI).     */
/*                                                                            */
/* Statement:    Licensed Materials - Property of IBM                         */
/*                                                                            */
/*               84H2000, 5765-B73                                            */
/*               84H2001, 5639-B42                                            */
/*               84H2002, 5765-B74                                            */
/*               84H2003, 5765-B75                                            */
/*               84H2004, 5639-B43                                            */
/*                                                                            */
/*               (C) Copyright IBM Corp. 1999                                 */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* Function:                                                                  */
/*    AMQSAIEM is a sample C program that demonstrates how to write a simple  */
/*    event monitor using the mqGetBag call and other MQAI calls.             */
/*                                                                            */
/*    The name of the event queue to be monitored is passed as a parameter    */
/*    to the program. This would usually be one of the system event queues:-  */
/*            SYSTEM.ADMIN.QMGR.EVENT        queue manager events             */
/*            SYSTEM.ADMIN.PERFM.EVENT       Performance events               */
/*            SYSTEM.ADMIN.CHANNEL.EVENT     Channel events                   */
/*    To monitor the queue manager event queue or the Performance event queue */
/*    the attributes of the queue manager will need to be changed to enable   */
/*    the events, refer to the WebSphere MQ Event Monitoring*/
/*    book for more information.                                              */
/*    The queue manager attributes can be changed either by                   */
/*    MQSC commands or using the MQAI interface.                              */
/*    Channel events are enabled by default.                                  */
/*                                                                            */
/* Program logic                                                              */
/*    Connect to the queue manager.                                           */
/*    Open the requested event queue with the wait unlimited option.          */
/*    Wait for a message and when it arrives get the message from the queue   */
/*    and format it into an MQAI bag with the mqGetBag call.                  */
/*    There are many types of event messages and it is beyond the scope of    */
/*    this sample to program for all event messages. Instead print out the    */
/*    contents of the formatted bag.                                          */
/*    Loop around to wait for another message until either there is an error  */
/*    or the program is stopped by a user interrupt.                          */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* AMQSAIEM has 2 parameters - the name of the event queue to be monitored    */
/*                           - the queue manager name (optional)              */
/*                                                                            */
/******************************************************************************/
```

Figure 16. AMQSAIEM.C: Displaying events (Part 1 of 8)

```
/******************************************************************************/
/* Includes                                                                   */
/******************************************************************************/
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#include <ctype.h>
#include <cmqc.h>                          /* MQI                             */
#include <cmqcfc.h>                        /* PCF                             */
#include <cmqbc.h>                         /* MQAI                            */

/******************************************************************************/
/* Function prototypes                                                        */
/******************************************************************************/
void CheckCallResult(MQCHAR *, MQLONG , MQLONG);
void GetQEvents(MQHCONN, MQCHAR *);
int PrintBag(MQHBAG);
int PrintBagContents(MQHBAG, int);

int main(int argc, char *argv[])
{
   MQHCONN hConn;                          /* handle to connection            */
   MQCHAR QMName[MQ_Q_MGR_NAME_LENGTH+1]=""; /* default QM name               */
   MQLONG reason;                          /* reason code                     */
   MQLONG connReason;                      /* MQCONN reason code              */
   MQLONG compCode;                        /* completion code                 */

   /***************************************************************************/
   /* First check the required parameters                                     */
   /***************************************************************************/
   printf("Sample Event Monitor (^C to stop)\n");
     if (argc < 2)
   {
     printf("Required parameter missing - event queue to be monitored\n")
     exit(99);
   }

   /***************************************************************************/
   /* Connect to the queue manager                                            */
   /***************************************************************************/
   if (argc > 2)
     strncpy(QMName, argv[2], (size_t)MQ_Q_MGR_NAME_LENGTH);
   MQCONN(QMName, &hConn, &compCode, &connReason);

   /***************************************************************************/
   /* Report the reason and stop if the connection failed                     */
   /***************************************************************************/
   if (compCode == MQCC_FAILED)
   {
      CheckCallResult("MQCONN", compCode, connReason);
      exit( (int)connReason);
   }

   /***************************************************************************/
   /* Call the routine to open the event queue and format any event message   */
   /* read from the queue.                                                    */
   /***************************************************************************/
   GetQEvents(hConn, argv[1]);
```

Figure 16. AMQSAIEM.C: Displaying events (Part 2 of 8)

```
   /******************************************************************************/
   /* Disconnect from the queue manager if not already connected                 */
   /******************************************************************************/
   if (connReason != MQRC_ALREADY_CONNECTED)
   {
      MQDISC(&hConn, &compCode, &reason);
      CheckCallResult("MQDISC", compCode, reason);
   }

   return 0;

}

/******************************************************************************/
/*                                                                            */
/* Function: CheckCallResult                                                  */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* Input Parameters:  Description of call                                     */
/*                    Completion code                                         */
/*                    Reason code                                             */
/*                                                                            */
/* Output Parameters: None                                                    */
/*                                                                            */
/* Logic: Display the description of the call, the completion code and the    */
/*        reason code if the completion code is not successful                */
/*                                                                            */
/******************************************************************************/
void  CheckCallResult(char *callText, MQLONG cc, MQLONG rc)
{
   if (cc != MQCC_OK)
         printf("%s failed: Completion Code = %d : Reason = %d\n",
                 callText, cc, rc);

}


/******************************************************************************/
/*                                                                            */
/* Function: GetQEvents                                                       */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* Input Parameters:  Handle to the queue manager                             */
/*                    Name of the event queue to be monitored                 */
/*                                                                            */
/* Output Parameters: None                                                    */
/*                                                                            */
/* Logic:   Open the event queue.                                             */
/*          Get a message off the event queue and format the message into     */
/*          a bag.                                                            */
/*          A real event monitor would need to be programmed to deal with     */
/*          each type of event that it receives from the queue. This is       */
/*          outside the scope of this sample so instead the contents of       */
/*          the bag are printed.                                              */
/*          The program waits forever for a message on the queue so the      */
/*          program must be terminated by a user interrupt (^C).              */
/*                                                                            */
/******************************************************************************/
```

Figure 16. AMQSAIEM.C: Displaying events (Part 3 of 8)

```
void GetQEvents(MQHCONN hConn, MQCHAR *qName)
{
   MQLONG openReason;                       /* MQOPEN reason code            */
   MQLONG reason;                           /* reason code                   */
   MQLONG compCode;                         /* completion code               */
   MQHOBJ eventQueue;                       /* handle to event queue         */
   MQHBAG eventBag = MQHB_UNUSABLE_HBAG;    /* event bag to receive event msg */
   MQOD   od = {MQOD_DEFAULT};              /* Object Descriptor             */
   MQMD   md = {MQMD_DEFAULT};              /* Message Descriptor            */
   MQGMO  gmo = {MQGMO_DEFAULT};            /* get message options           */
   MQLONG bQueueOK = 1;                     /* keep reading msgs while true  */
   /***************************************************************************/
   /* Create an Event Bag in which to receive the event. Message exit the     */
   /* function if the create fails.                                           */
   /***************************************************************************/
   mqCreateBag(MQCBO_USER_BAG, &eventBag, &compCode, &reason);
   CheckCallResult("Create event bag", compCode, reason);
   if (compCode !=MQCC_OK)
      return;

   /***************************************************************************/
   /* Open the event queue chosen by the user                                 */
   /***************************************************************************/
   strncpy(od.ObjectName, qName, (size_t)MQ_Q_NAME_LENGTH);
   MQOPEN(hConn, &od, MQOO_INPUT_AS_Q_DEF+MQOO_FAIL_IF_QUIESCING, &eventQueue,
          &compCode, &openReason);
   CheckCallResult("Open event queue", compCode, openReason);

   /***************************************************************************/
   /* Set the GMO options to control the action of the get message from the   */
   /* queue.                                                                  */
   /***************************************************************************/
   gmo.WaitInterval = MQWI_UNLIMITED;      /* Wait forever for a message      */
   gmo.Options = MQGMO_WAIT + MQGMO_FAIL_IF_QUIESCING;
   gmo.Version = MQGMO_VERSION_2;          /* Avoid need to reset Message ID  */
   gmo.MatchOptions = MQMO_NONE;           /* and Correlation ID after every  */
                                           /* mqGetBag                        */

   /***************************************************************************/
   /* If open failed we cannot access the queue and must stop the monitor.    */
   /***************************************************************************/
   if (compCode != MQCC_OK)
     bQueueOK = 0;

   /***************************************************************************/
   /* Main loop to get an event message when it arrives                       */
   /***************************************************************************/
   while (bQueueOK)
   {
     printf("\nWaiting for an event\n");

     /*************************************************************************/
     /* Get the message from the event queue and convert it into the event    */
     /* bag.                                                                  */
     /*************************************************************************/
     mqGetBag(hConn, eventQueue, &md, &gmo, eventBag, &compCode, &reason);
     CheckCallResult("Get bag", compCode, reason);
```

Figure 16. AMQSAIEM.C: Displaying events (Part 4 of 8)

```
     if (compCode != MQCC_OK)
        bQueueOK = 0;

     else
     {
       /********************************************************************/
       /* Event message read - Print the contents of the event bag         */
       /********************************************************************/
       if ( PrintBag(eventBag) )
           printf("\nError found while printing bag contents\n");

   }  /* end of msg found */
 } /* end of main loop */
   /***********************************************************************/
   /* Close the event queue if successfully opened                        */
   /***********************************************************************/
   if (openReason == MQRC_NONE)
   {
      MQCLOSE(hConn, &eventQueue, MQCO_NONE, &compCode, &reason);
      CheckCallResult("Close event queue", compCode, reason);
   }

   /***********************************************************************/
   /* Delete the event bag if successfully created.                       */
   /***********************************************************************/
   if (eventBag != MQHB_UNUSABLE_HBAG)
   {
      mqDeleteBag(&eventBag, &compCode, &reason);
      CheckCallResult("Delete the event bag", compCode, reason);
   }

} /* end of GetQEvents */

/**************************************************************************/
/*                                                                        */
/* Function: PrintBag                                                     */
/*                                                                        */
/**************************************************************************/
/*                                                                        */
/* Input Parameters:  Bag Handle                                          */
/*                                                                        */
/* Output Parameters: None                                                */
/*                                                                        */
/* Returns:           Number of errors found                              */
/*                                                                        */
/* Logic: Calls PrintBagContents to display the contents of the bag.      */
/*                                                                        */
/**************************************************************************/
int PrintBag(MQHBAG dataBag)
{
    int errors;

    printf("\n");
    errors = PrintBagContents(dataBag, 0);
    printf("\n");

    return errors;
}
```

Figure 16. AMQSAIEM.C: Displaying events (Part 5 of 8)

```
/******************************************************************************/
/*                                                                            */
/* Function: PrintBagContents                                                 */
/*                                                                            */
/******************************************************************************/
/*                                                                            */
/* Input Parameters:  Bag Handle                                              */
/*                    Indentation level of bag                                */
/*                                                                            */
/* Output Parameters: None                                                    */
/*                                                                            */
/* Returns:           Number of errors found                                  */
/*                                                                            */
/* Logic: Count the number of items in the bag                                */
/*        Obtain selector and item type for each item in the bag.             */
/*        Obtain the value of the item depending on item type and display the */
/*        index of the item, the selector and the value.                      */
/*        If the item is an embedded bag handle then call this function again */
/*        to print the contents of the embedded bag increasing the            */
/*        indentation level.                                                  */
/*                                                                            */
/******************************************************************************/
int PrintBagContents(MQHBAG dataBag, int indent)
{
   #define LENGTH 500                      /* Max length of string to be read*/
   #define INDENT 4                        /* Number of spaces to indent     */
                                           /* embedded bag display           */

   MQLONG  itemCount;                      /* Number of items in the bag     */
   MQLONG  itemType;                       /* Type of the item               */
   int     i;                              /* Index of item in the bag       */
   char    stringVal[LENGTH+1];            /* Value if item is a string      */
   MQLONG  stringLength;                   /* Length of string value         */
   MQLONG  ccsid;                          /* CCSID of string value          */
   MQLONG  iValue;                         /* Value if item is an integer    */
   MQLONG  selector;                       /* Selector of item               */
   MQHBAG  bagHandle;                      /* Value if item is a bag handle  */
   MQLONG  reason;                         /* reason code                    */
   MQLONG  compCode;                       /* completion code                */
   MQLONG  trimLength;                     /* Length of string to be trimmed */
   int     errors = 0;                     /* Count of errors found          */
   char    blanks[]= "                      "; /* Blank string used to      */
                                               /* indent display            */

   /***************************************************************************/
   /* Count the number of items in the bag                                    */
   /***************************************************************************/
   mqCountItems(dataBag, MQSEL_ALL_SELECTORS, &itemCount, &compCode, &reason);

   if (compCode != MQCC_OK)
      errors++;
   else
   {
      printf("%.*sHandle:%d ", indent, blanks, dataBag);
      printf("%.*sSize:%d\n", indent, blanks, itemCount);
      printf("%.*sIndex: Selector: Value:\n", indent, blanks);
   }
```

Figure 16. AMQSAIEM.C: Displaying events (Part 6 of 8)

```
/******************************************************************************/
/* If no errors found then display each item in the bag                      */
/******************************************************************************/
if (!errors)
{
   for (i = 0; i < itemCount; i++)
   {
       /*******************************************************************/
       /* First inquire the type of the item for each item in the bag     */
       /*******************************************************************/
       mqInquireItemInfo(dataBag,            /* Bag handle               */
                         MQSEL_ANY_SELECTOR,  /* Item can have any selector*/
                         i,                   /* Index position in the bag */
                         &selector,           /* Actual value of selector  */
                                              /* returned by call          */
                         &itemType,           /* Actual type of item       */
                                              /* returned by call          */
                         &compCode,           /* Completion code           */
                         &reason);            /* Reason Code               */

       if (compCode != MQCC_OK)
          errors++;
       switch(itemType)
       {
       case MQIT_INTEGER:
            /**************************************************************/
            /* Item is an integer. Find its value and display its index,   */
            /* selector and value.                                         */
            /**************************************************************/
            mqInquireInteger(dataBag,        /* Bag handle               */
                             MQSEL_ANY_SELECTOR, /* Allow any selector     */
                             i,               /* Index position in the bag */
                             &iValue,         /* Returned integer value    */
                             &compCode        /* Completion code           */
                             &reason);        /* Reason Code               */

            if (compCode != MQCC_OK)
               errors++;
            else
               printf("%.*s  %-2d  %-4d  (%d)\n",
                       indent, blanks, i, selector, iValue);
            break;

       case MQIT_STRING:
            /**************************************************************/
            /* Item is a string. Obtain the string in a buffer, prepare    */
            /* the string for displaying and display the index, selector,  */
            /* string and character set ID.                                */
            /**************************************************************/
            mqInquireString(dataBag,         /* Bag handle               */
                            MQSEL_ANY_SELECTOR, /* Allow any selector      */
                            i,                /* Index position in the bag */
                            LENGTH,           /* Maximum length of buffer  */
                            stringVal,        /* Buffer to receive string  */
                            &stringLen        /* Actual length of string   */
                            &ccsid,           /* Coded character set ID    */
                            &reason);         /* Reason Code               */
```

Figure 16. AMQSAIEM.C: Displaying events (Part 7 of 8)

```
                  if (compCode == MQCC_FAILED)
                      errors++;
                  else
                  {
                     /************************************************************/
                     /* Remove trailing blanks from the string and terminate with*/
                     /* a null. First check that the string should not have been */
                     /* longer than the maximum buffer size allowed.             */
                     /************************************************************/
                     if (stringLength > LENGTH)
                        trimLength = LENGTH;
                     else
                        trimLength = stringLength;
                     mqTrim(trimLength, stringVal, stringVal, &compCode, &reason);
                     printf("%.*s  %-2d     %-4d'%s' %d\n",
                             indent, blanks, i, selector, stringVal, ccsid);
                  }
                  break;
             case MQIT_BAG:
                  /**************************************************************/
                  /* Item is an embedded bag handle, so call the function again  */
                  /* to display the contents.                                    */
                  /**************************************************************/
                  mqInquireBag(dataBag,             /* Bag handle                */
                               MQSEL_ANY_SELECTOR,  /* Allow any selector        */
                               i,                   /* Index position in the bag */
                               &bagHandle,          /* Returned embedded bag hdle*/
                               &compCode,           /* Completion code           */
                               &reason);            /* Reason Code               */

                  if (compCode != MQCC_OK)
                     errors++;
                  else
                  {
                     printf("%.*s  %-2d     %-4d     (%d)\n", indent, blanks,
                             i, selector, bagHandle);
                     printf("%.*sSystem Bag:\n", indent+INDENT, blanks);
                     PrintBagContents(bagHandle, indent+INDENT);
                  }
                  break;

             default:
                  printf("%.*sUnknown item type", indent, blanks);
             }
         }
      }
      return errors;
   }
```

Figure 16. AMQSAIEM.C: Displaying events (Part 8 of 8)

Chapter 19. Advanced topics

This chapter discusses the following:
- Indexing
- Data conversion
- Use of the message descriptor

Indexing

Each selector and value within a data item in a bag have three associated index numbers:
- The index relative to other items that have the same selector.
- The index relative to the category of selector (user or system) to which the item belongs.
- The index relative to all the data items in the bag (user and system).

This allows indexing by user selectors, system selectors, or both as shown in Figure 17.

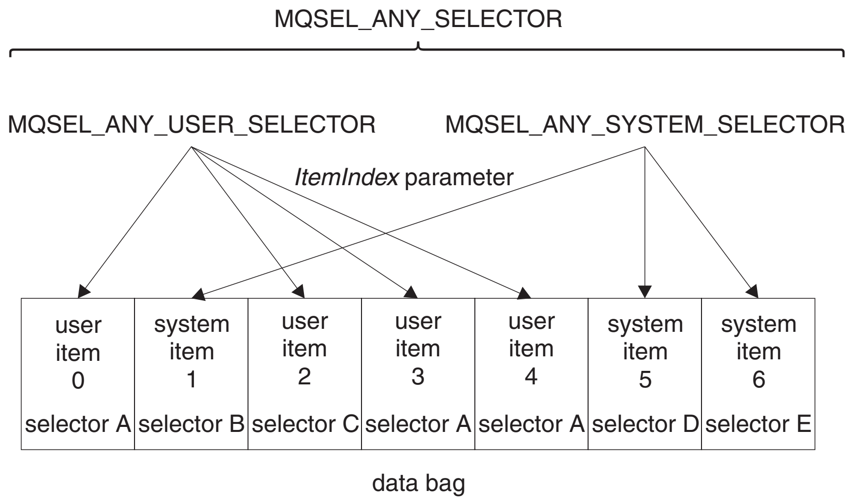


Figure 17. Indexing

In Figure 17, user item 3 (selector A) can be referred to by the following index pairs:

Selector	*ItemIndex*
selector A	1
MQSEL_ANY_USER_SELECTOR	2
MQSEL_ANY_SELECTOR	3

The index is zero-based like an array in C; if there are ‘n’ occurrences, the index ranges from zero through ‘n-1’, with no gaps.

Indexes are used when replacing or removing existing data items from a bag. When used in this way, the insertion order is preserved, but indexes of other data items can be affected. For examples of this, see “Changing information within a bag” on page 242 and “Deleting data items” on page 243.

The three types of indexing allow easy retrieval of data items. For example, if there are three instances of a particular selector in a bag, the mqCountItems call can count the number of instances of that selector, and the mqInquire* calls can specify both the selector and the index to inquire those values only. This is useful for attributes that can have a list of values such as some of the exits on channels.

Data conversion

Like PCF messages, the strings contained in an MQAI data bag can be in a variety of coded character sets. Usually, all of the strings in a PCF message are in the same coded character set; that is, the same set as the queue manager.

Each string item in a data bag contains two values; the string itself and the CCSID. The string that is added to the bag is obtained from the `Buffer` parameter of the mqAddString or mqSetString call. The CCSID is obtained from the system item containing a selector of MQIASY_CODED_CHAR_SET_ID. This is known as the *bag CCSID* and can be changed using the mqSetInteger call.

When you inquire the value of a string contained in a data bag, the CCSID is an output parameter from the call.

Table 10 shows the rules applied when converting data bags into messages and vice versa:

Table 10. CCSID processing

MQAI call	CCSID	Input to call	Output to call
mqBagToBuffer	Bag CCSID (1)	Ignored	Unchanged
mqBagToBuffer	String CCSIDs in bag	Used	Unchanged
mqBagToBuffer	String CCSIDs in buffer	Not applicable	Copied from string CCSIDs in bag
mqBufferToBag	Bag CCSID (1)	Ignored	Unchanged
mqBufferToBag	String CCSIDs in buffer	Used	Unchanged
mqBufferToBag	String CCSIDs in bag	Not applicable	Copied from string CCSIDs in buffer
mqPutBag	MQMD CCSID	Used	Unchanged (2)
mqPutBag	Bag CCSID (1)	Ignored	Unchanged
mqPutBag	String CCSIDs in bag	Used	Unchanged
mqPutBag	String CCSIDs in message sent	Not applicable	Copied from string CCSIDs in bag
mqGetBag	MQMD CCSID	Used for data conversion of message	Set to CCSID of data returned (3)
mqGetBag	Bag CCSID (1)	Ignored	Unchanged
mqGetBag	String CCSIDs in message	Used	Unchanged
mqGetBag	String CCSIDs in bag	Not applicable	Copied from string CCSIDs in message

Table 10. CCSID processing (continued)

MQAI call	CCSID	Input to call	Output to call
mqExecute	Request-bag CCSID	Used for MQMD of request message (4)	Unchanged
mqExecute	Reply-bag CCSID	Used for data conversion of reply message (4)	Set to CCSID of data returned (3)
mqExecute	String CCSIDs in request bag	Used for request message	Unchanged
mqExecute	String CCSIDs in reply bag	Not applicable	Copied from string CCSIDs in reply message

Notes:
1. Bag CCSID is the system item with selector MQIASY_CODED_CHAR_SET_ID.
2. MQCCSI_Q_MGR is changed to the actual queue manager CCSID.
3. If data conversion is requested, the CCSID of data returned is the same as the output value. If data conversion is not requested, the CCSID of data returned is the same as the message value. Note that no message is returned if data conversion is requested but fails.
4. If the CCSID is MQCCSI_DEFAULT, the queue manager's CCSID is used.

Use of the message descriptor

The PCF command type is obtained from the system item with selector MQIASY_TYPE. When you create your data bag, the initial value of this item is set depending on the type of bag you create:

Table 11. PCF command type

Type of bag	Initial value of MQIASY_TYPE item
MQCBO_ADMIN_BAG	MQCFT_COMMAND
MQCBO_COMMAND_BAG	MQCFT_COMMAND
MQCBO_*	MQCFT_USER

When the MQAI generates a message descriptor, the values used in the *Format* and *MsgType* parameters depend on the value of the system item with selector MQIASY_TYPE as shown in Table 11.

Table 12. Format and MsgType parameters of the MQMD

PCF command type	Format	MsgType
MQCFT_COMMAND	MQFMT_ADMIN	MQMT_REQUEST
MQCFT_RESPONSE	MQFMT_ADMIN	MQMT_REPLY
MQCFT_EVENT	MQFMT_EVENT	MQMT_DATAGRAM
MQCFT_*	MQFMT_PCF	MQMT_DATAGRAM

Table 12 shows that if you create an administration bag or a command bag, the *Format* of the message descriptor is MQFMT_ADMIN and the *MsgType* is MQMT_REQUEST. This is suitable for a PCF request message sent to the command server when a response is expected back.

Message descriptor

Other parameters in the message descriptor take the values shown in Table 13.

Table 13. Message descriptor values

Parameter	**Value**
StrucId	MQMD_STRUC_ID
Version	MQMD_VERSION_1
Report	MQRO_NONE
MsgType	see Table 12 on page 337
Expiry	30 seconds (note 1)
Feedback	MQFB_NONE
Encoding	MQENC_NATIVE
CodedCharSetId	depends on the bag CCSID (note 2)
Format	see Table 12 on page 337
Priority	MQPRI_PRIORITY_AS_Q_DEF
Persistence	MQPER_NOT_PERSISTENT
MsgId	MQMI_NONE
CorelId	MQCI_NONE
BackoutCount	0
ReplyToQ	see note 3
ReplyToQMgr	blank

Notes:

1. This value can be overridden on the the mqExecute call by using the *OptionsBag* parameter. For information about this, see "mqExecute" on page 279.
2. See "Data conversion" on page 336.
3. Name of the user-specified reply queue or MQAI-generated temporary dynamic queue for messages of type MQMT_REQUEST. Blank otherwise.

Part 3. Appendixes

Appendix A. Error codes

This book contains the return codes associated with PCFs. The return codes associated with the Message Queuing Interface (MQI) are listed in:

- *WebSphere MQ for z/OS Messages and Codes* for WebSphere MQ for z/OS
- *WebSphere MQ Messages* for all other WebSphere MQ platforms

This chapter discusses:

- "Completion code"
- "Reason code"

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 might 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:

| • *WebSphere MQ for z/OS Messages and Codes* for WebSphere MQ for z/OS
| • *WebSphere MQ Messages* for all other WebSphere MQ platforms

The following is a list, in alphabetic order, of the MQRCCF_* reason codes:

3091 (X'0C13') MQRCCF_ACTION_VALUE_ERROR

Explanation: Action value not valid.

The value specified for *Action* is not valid. There is only one valid value.

Programmer Response: Specify MQACT_FORCE_REMOVE as the value of the *Action* parameter.

| **3166 (X'0C5E')**
| **MQRCCF_ALLOC_FAST_TIMER_ERROR**

| **Explanation:** Allocation fast retry timer value not
| valid.

| The *AllocRetryFastTimer* value was not valid.

| **Programmer Response:** Specify a valid value.

| **3164 (X'0C5C') MQRCCF_ALLOC_RETRY_ERROR**

| **Explanation:** Allocation retry count not valid.

| The *AllocRetryCount* value was not valid.

| **Programmer Response:** Specify a valid count.

| **3165 (X'0C5D')**
| **MQRCCF_ALLOC_SLOW_TIMER_ERROR**

| **Explanation:** Allocation slow retry timer value not
| valid.

| The *AllocRetrySlowTimer* value was not valid.

| **Programmer Response:** Specify a valid timer value.

4009 (X'0FA9') MQRCCF_ALLOCATE_FAILED

Explanation: Allocation failed.

An attempt to allocate a conversation to a remote system failed. The error may be due to an entry in the channel definition that is not valid, or it might be that the listening program at the remote system is not running.

Programmer Response: Ensure that the channel definition is correct, and start the listening program if necessary. If the error persists, consult your systems administrator.

4005 (X'0FA5') MQRCCF_ATTR_VALUE_ERROR

Explanation: 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).

Programmer Response: Specify only valid attribute values.

4086 (X'0FF6') MQRCCF_BATCH_INT_ERROR

Explanation: Batch interval not valid.

The batch interval specified was not valid.

Programmer Response: Specify a valid batch interval value.

4087 (X'0FF7')
MQRCCF_BATCH_INT_WRONG_TYPE

Explanation: Batch interval parameter not allowed for this channel type.

The *BatchInterval* parameter is allowed only for sender and server channels.

Programmer Response: Remove the parameter.

3037 (X'0BDD') MQRCCF_BATCH_SIZE_ERROR

Explanation: Batch size not valid.

The batch size specified was not valid.

Programmer Response: Specify a valid batch size value.

4024 (X'0FB8') MQRCCF_BIND_FAILED

Explanation: Bind failed.

The bind to a remote system during session negotiation has failed.

Programmer Response: Consult your systems administrator.

3049 (X'0BE9') MQRCCF_CCSID_ERROR

Explanation: 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.

Programmer Response: 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.

4068 (X'0FE4') MQRCCF_CELL_DIR_NOT_AVAILABLE

Explanation: 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.

Programmer Response: Configure the queue manager with a suitable name service.

3007 (X'0BBF') MQRCCF_CFH_COMMAND_ERROR

Explanation: Command identifier not valid.

The MQCFH *Command* field value was not valid.

Programmer Response: Specify a valid command identifier.

3005 (X'0BBD') MQRCCF_CFH_CONTROL_ERROR

Explanation: Control option not valid.

The MQCFH *Control* field value was not valid.

Programmer Response: Specify a valid control option.

3002 (X'0BBA') MQRCCF_CFH_LENGTH_ERROR

Explanation: Structure length not valid.

The MQCFH *StrucLength* field value was not valid.

Programmer Response: Specify a valid structure length.

3004 (X'0BBC') MQRCCF_CFH_MSG_SEQ_NUMBER_ERR

Explanation: Message sequence number not valid.

The MQCFH *MsgSeqNumber* field value was not valid.

Programmer Response: Specify a valid message sequence number.

3006 (X'0BBE') MQRCCF_CFH_PARM_COUNT_ERROR

Explanation: Parameter count not valid.

The MQCFH *ParameterCount* field value was not valid.

Programmer Response: Specify a valid parameter count.

3001 (X'0BB9') MQRCCF_CFH_TYPE_ERROR

Explanation: Type not valid.

The MQCFH *Type* field value was not valid.

Programmer Response: Specify a valid type.

3003 (X'0BBB') MQRCCF_CFH_VERSION_ERROR

Explanation: Structure version number is not valid.

The MQCFH *Version* field value was not valid.

Programmer Response: Specify a valid structure version number.

3027 (X'0BD3') MQRCCF_CFIL_COUNT_ERROR

Explanation: Count of parameter values not valid.

The MQCFIL *Count* field value was not valid.

Programmer Response: Specify a valid count of parameter values.

3026 (X'0BD2') MQRCCF_CFIL_DUPLICATE_VALUE

Explanation: Duplicate parameter.

In the MQCFIL structure, a duplicate parameter was detected in the list selector.

Programmer Response: Check for and remove duplicate parameters.

3028 (X'0BD4') MQRCCF_CFIL_LENGTH_ERROR

Explanation: Structure length not valid.

The MQCFIL *StrucLength* field value was not valid.

Programmer Response: Specify a valid structure length.

3047 (X'0BE7') MQRCCF_CFIL_PARM_ID_ERROR

Explanation: Parameter identifier is not valid.

The MQCFIL *Parameter* field value was not valid.

Programmer Response: Specify a valid parameter identifier.

3017 (X'0BC9') MQRCCF_CFIN_DUPLICATE_PARM

Explanation: Duplicate parameter.

A MQCFIN duplicate parameter was detected.

Programmer Response: Check for and remove duplicate parameters.

3009 (X'0BC1') MQRCCF_CFIN_LENGTH_ERROR

Explanation: Structure length not valid.

The MQCFIN `StrucLength` field value was not valid.

Programmer Response: Specify a valid structure length.

3014 (X'0BC6') MQRCCF_CFIN_PARM_ID_ERROR

Explanation: Parameter identifier is not valid.

The MQCFIN `Parameter` field value was not valid.

Programmer Response: Specify a valid parameter identifier.

3068 (X'0BFC') MQRCCF_CFSL_COUNT_ERROR

Explanation: Name count value not valid.

Maximum number of names in a namelist exceeded. Maximum number of names is 256.

Programmer Response: Reduce number of names.

3066 (X'0BFA') MQRCCF_CFSL_DUPLICATE_PARM

Explanation: Duplicate parameter.

A MQCFSL duplicate parameter was detected.

Programmer Response: 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.

3024 (X'0BD0') MQRCCF_CFSL_LENGTH_ERROR

Explanation: Structure length not valid.

The MQCFSL `StrucLength` field value was not valid. The value was not a multiple of four or was inconsistent with the MQCFSL `StringLength` field value.

Programmer Response: Specify a valid structure length.

3033 (X'0BD9') MQRCCF_CFSL_PARM_ID_ERROR

Explanation: Parameter identifier is not valid.

The MQCFSL `Parameter` field value was not valid.

Programmer Response: Specify a valid parameter identifier.

3069 (X'0BFD') MQRCCF_CFSL_STRING_LENGTH_ERR

Explanation: String length not valid.

A name, within a namelist, with a nonblank string length of greater than 48 bytes was detected.

Programmer Response: Check that all names have a nonblank length of less than 48 bytes. Strings greater than 48 bytes are accepted if all bytes over 48 are blanks.

3067 (X'0BFB') MQRCCF_CFSL_TOTAL_LENGTH_ERROR

Explanation: 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.

Programmer Response: Check that the structure has been specified correctly, and if so reduce the number of strings.

3095 (X'0C17') MQRCCF_CFST_CONFLICTING_PARM

Explanation: Conflicting parameters.

The command was rejected because the parameter identified in the error response was in conflict with another parameter in the command.

Programmer Response: Consult the description of the parameter identified to ascertain the nature of the conflict, and the correct command.

3018 (X'0BCA') MQRCCF_CFST_DUPLICATE_PARM

Explanation: Duplicate parameter.

A MQCFST duplicate parameter was detected.

Programmer Response: 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.

3010 (X'0BC2') MQRCCF_CFST_LENGTH_ERROR

Explanation: 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.

Programmer Response: Specify a valid structure length.

3015 (X'0BC7') MQRCCF_CFST_PARM_ID_ERROR

Explanation: Parameter identifier is not valid.

The MQCFST *Parameter* field value was not valid.

Programmer Response: Specify a valid parameter identifier.

3011 (X'0BC3') MQRCCF_CFST_STRING_LENGTH_ERR

Explanation: 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.

Programmer Response: Specify a valid string length for the parameter.

4079 (X'0FEF') MQRCCF_CHAD_ERROR

Explanation: Channel automatic definition error.

The *ChannelAutoDef* value was not valid.

Programmer Response: Specify MQCHAD_ENABLED or MQCHAD_DISABLED.

4081 (X'0FF1') MQRCCF_CHAD_EVENT_ERROR

Explanation: Channel automatic definition event error.

The *ChannelAutoDefEvent* value was not valid.

Programmer Response: Specify MQEVR_ENABLED or MQEVR_DISABLED.

4082 (X'0FF2') MQRCCF_CHAD_EVENT_WRONG_TYPE

Explanation: Channel automatic definition event parameter not allowed for this channel type.

The *ChannelAutoDefEvent* parameter is allowed only for receiver and server-connection channels.

Programmer Response: Remove the parameter.

4083 (X'0FF3') MQRCCF_CHAD_EXIT_ERROR

Explanation: Channel automatic definition exit name error.

The *ChannelAutoDefExit* value contained characters that are not allowed for program names on the platform in question.

Programmer Response: Specify a valid name.

4084 (X'0FF4') MQRCCF_CHAD_EXIT_WRONG_TYPE

Explanation: Channel automatic definition exit parameter not allowed for this channel type.

The *ChannelAutoDefExit* parameter is allowed only for receiver and server-connection channels.

Programmer Response: Remove the parameter.

4080 (X'0FF0') MQRCCF_CHAD_WRONG_TYPE

Explanation: Channel automatic definition parameter not allowed for this channel type.

The *ChannelAutoDef* parameter is allowed only for receiver and server-connection channels.

Programmer Response: Remove the parameter.

4042 (X'0FCA') MQRCCF_CHANNEL_ALREADY_EXISTS

Explanation: Channel already exists.

An attempt was made to create a channel but the channel already existed and *Replace* was not specified as MQRP_YES.

Programmer Response: Specify *Replace* as MQRP_YES or use a different name for the channel to be created.

4090 (X'0FFA') MQRCCF_CHANNEL_CLOSED

Explanation: Channel closed

The channel was closed prematurely. This can occur because a user stopped the channel while it was running, or a channel exit decided to close the channel.

Programmer Response: Determine the reason that the channel was closed prematurely. Restart the channel if required.

4038 (X'0FC6') MQRCCF_CHANNEL_DISABLED

Explanation: Channel disabled.

An attempt was made to use a channel, but the channel was disabled.

Programmer Response: Start the channel.

4031 (X'0FBF') MQRCCF_CHANNEL_IN_USE

Explanation: Channel in use.

An attempt was made to perform an operation on a channel, but the channel is currently active.

Programmer Response: Stop the channel or wait for it to terminate.

4025 (X'0FB9') MQRCCF_CHANNEL_INDOUBT

Explanation: Channel in-doubt.

The requested operation cannot complete because the channel is in doubt.

Programmer Response: Examine the status of the channel, and either restart a channel to resolve the in-doubt state, or resolve the channel.

4044 (X'0FCC') MQRCCF_CHANNEL_NAME_ERROR

Explanation: Channel name error.

The *ChannelName* parameter contained characters that are not allowed for channel names.

Programmer Response: Specify a valid name.

4064 (X'0FE0') MQRCCF_CHANNEL_NOT_ACTIVE

Explanation: Channel not active.

An attempt was made to stop a channel, but the channel was already stopped.

Programmer Response: No action is required.

4032 (X'0FC0') MQRCCF_CHANNEL_NOT_FOUND

Explanation: Channel not found.

The channel specified does not exist.

Programmer Response: Specify the name of a channel which exists.

3062 (X'0BF6') MQRCCF_CHANNEL_TABLE_ERROR

Explanation: 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.

Programmer Response: Specify a valid channel table value.

3034 (X'0BDA') MQRCCF_CHANNEL_TYPE_ERROR

Explanation: 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.

Programmer Response: Specify a valid channel type.

3064 (X'0BF8') MQRCCF_CHL_INST_TYPE_ERROR

Explanation: Channel instance type not valid.

The *ChannelInstanceType* specified was not valid.

Programmer Response: Specify a valid channel instance type.

3065 (X'0BF9') MQRCCF_CHL_STATUS_NOT_FOUND

Explanation: 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.

Programmer Response: None, unless this is unexpected, in which case consult your systems administrator.

| **3168 (X'0C60') MQRCCF_CHL_SYSTEM_NOT_ACTIVE**

| **Explanation:** Channel system is not active.

| An attempt was made to start a channel while the
| channel system was inactive.

| **Programmer Response:** Activate the channel system
| before starting a channel.

3088 (X'0C10') MQRCCF_CLUSTER_NAME_CONFLICT

Explanation: *ClusterName* and *ClusterNamelist* attributes conflict.

The command was rejected because it would have resulted in the *ClusterName* attribute and the *ClusterNamelist* attribute both being nonblank. At least one of these attributes must be blank.

Programmer Response: If the command specified one of these attributes only, you must also specify the other one, but with a value of blanks. If the command specified both attributes, ensure that one of them has a value of blanks.

3090 (X'0C12') MQRCCF_CLUSTER_Q_USAGE_ERROR

Explanation: Cluster queue cannot be a transmission queue.

The command was rejected because it would have resulted in a cluster queue also being a transmission queue. This is not permitted.

Programmer Response: Ensure that the command specifies either:

- The *Usage* parameter with a value of MQUS_NORMAL, or
- The *ClusterName* and *ClusterNamelist* parameters with values of blanks.

3008 (X'0BC0') MQRCCF_COMMAND_FAILED

Explanation: Command failed.

The command has failed.

Programmer Response: Refer to the previous error messages for this command.

4040 (X'0FC8') MQRCCF_COMMIT_FAILED

Explanation: Commit failed.

An error was received when an attempt was made to commit a unit of work.

Programmer Response: Consult your systems administrator.

3092 (X'0C14') MQRCCF_COMMS_LIBRARY_ERROR

Explanation: Library for requested communications protocol could not be loaded.

The library needed for the requested communications protocol could not be loaded.

Programmer Response: Install the library for the required communications protocol, or specify a communications protocol that has already been installed.

4011 (X'0FAB') MQRCCF_CONFIGURATION_ERROR

Explanation: Configuration error.

A configuration error was detected in the channel definition or communication subsystem, and allocation of a conversation was not possible. This might 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 might not be established.
- For TCP, the *ConnectionName* in the channel definition cannot be resolved to a network address. This might be because the name has not been correctly specified, or because the name server is not available.
- The requested communications protocol might not be supported on the platform.

Programmer Response: Identify the error and take appropriate action.

4062 (X'0FDE') MQRCCF_CONN_NAME_ERROR

Explanation: Error in connection name parameter.

The *ConnectionName* parameter contains one or more blanks at the start of the name.

Programmer Response: Specify a valid connection name.

4017 (X'0FB1') MQRCCF_CONNECTION_CLOSED

Explanation: Connection closed.

An error occurred while receiving data from a remote system. The connection to the remote system has unexpectedly terminated.

Programmer Response: Contact your systems administrator.

4012 (X'0FAC') MQRCCF_CONNECTION_REFUSED

Explanation: 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 the remote system might not recognize the local system as valid, or the TCP listener program might not be started.

Programmer Response: Correct the error or restart the listener program.

3052 (X'0BEC') MQRCCF_DATA_CONV_VALUE_ERROR

Explanation: Data conversion value not valid.

The value specified for *DataConversion* is not valid.

Programmer Response: Specify a valid value.

4043 (X'0FCB') MQRCCF_DATA_TOO_LARGE

Explanation: Data too large.

The data to be sent exceeds the maximum that can be supported for the command.

Programmer Response: Reduce the size of the data.

3038 (X'0BDE') MQRCCF_DISC_INT_ERROR

Explanation: Disconnection interval not valid.

The disconnection interval specified was not valid.

Programmer Response: Specify a valid disconnection interval.

4054 (X'0FD6') MQRCCF_DISC_INT_WRONG_TYPE

Explanation: Disconnection interval not allowed for this channel type.

The *DiscInterval* parameter is only allowed for sender or server channel types.

Programmer Response: Remove the parameter.

| **3163 (X'0C5B') MQRCCF_DISC_RETRY_ERROR**

| **Explanation:** Disconnection retry count not valid.

| The *DiscRetryCount* value was not valid.

| **Programmer Response:** Specify a valid count.

4067 (X'0FE3') MQRCCF_DYNAMIC_Q_SCOPE_ERROR

Explanation: Dynamic queue scope error.

The *Scope* attribute of the queue is to be MQSCO_CELL, but this is not allowed for a dynamic queue.

Programmer Response: Predefine the queue if it is to have cell scope.

3050 (X'0BEA') MQRCCF_ENCODING_ERROR

Explanation: 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.

Programmer Response: Construct the command with the correct encoding, and specify this in the message descriptor when sending the command.

4013 (X'0FAD') MQRCCF_ENTRY_ERROR

Explanation: Connection name not valid.

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.

Programmer Response: Ensure that the connection name is correctly specified and that the name server is available.

3054 (X'0BEE') MQRCCF_ESCAPE_TYPE_ERROR

Explanation: Escape type not valid.

The value specified for *EscapeType* is not valid.

Programmer Response: Specify a valid value.

| **3162 (X'0C5A') MQRCCF_FILE_NOT_AVAILABLE**

| **Explanation:** File not available to CICS.

| A file name parameter identifies a file that is defined to
| CICS, but is not available.

| **Programmer Response:** Check that the CSD definition
| for the file is correct and enabled.

3150 (X'0C4E') MQRCCF_FILTER_ERROR

Explanation: Content based filter expression not valid.

The filter expression supplied in the publish/subscribe command message contains invalid syntax, and cannot be used.

Programmer Response: Correct the syntax of the filter expression in the publish/subscribe command message. The filter expression is the value of the *Filter* tag in the *psc* folder in the MQRFH2 structure. See the *Websphere MQ Integrator V2 Programming Guide* for details of valid syntax.

3012 (X'0BC4') MQRCCF_FORCE_VALUE_ERROR

Explanation: Force value not valid.

The force value specified was not valid.

Programmer Response: Specify a valid force value.

4077 (X'0FED') MQRCCF_HB_INTERVAL_ERROR

Explanation: Heartbeat interval not valid.

The *HeartbeatInterval* value was not valid.

Programmer Response: Specify a value in the range 0-999 999.

4078 (X'0FEE') MQRCCF_HB_INTERVAL_WRONG_TYPE

Explanation: Heartbeat interval parameter not allowed for this channel type.

The *HeartbeatInterval* parameter is allowed only for receiver and requester channels.

Programmer Response: Remove the parameter.

4010 (X'0FAA') MQRCCF_HOST_NOT_AVAILABLE

Explanation: Remote system not available.

An attempt to allocate a conversation to a remote system was unsuccessful. The error might be transitory, and the allocate might succeed later. This reason can occur if the listening program at the remote system is not running.

Programmer Response: Ensure that the listening program is running, and retry the operation.

3053 (X'0BED') MQRCCF_INDOUBT_VALUE_ERROR

Explanation: In-doubt value not valid.

The value specified for *InDoubt* is not valid.

Programmer Response: Specify a valid value.

4003 (X'0FA3') MQRCCF_LIKE_OBJECT_WRONG_TYPE

Explanation: 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.

Programmer Response: Ensure that the new object has the same type as the one on which it is based.

4020 (X'0FB4') MQRCCF_LISTENER_NOT_STARTED

Explanation: 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.

Programmer Response: Ensure the communications subsystem is started or retry the operation later.

3041 (X'0BE1') MQRCCF_LONG_RETRY_ERROR

Explanation: Long retry count not valid.

The long retry count value specified was not valid.

Programmer Response: Specify a valid long retry count value.

4057 (X'0FD9') MQRCCF_LONG_RETRY_WRONG_TYPE

Explanation: Long retry parameter not allowed for this channel type.

The `LongRetryCount` parameter is only allowed for sender or server channel types.

Programmer Response: Remove the parameter.

3042 (X'0BE2') MQRCCF_LONG_TIMER_ERROR

Explanation: Long timer not valid.

The long timer (long retry wait interval) value specified was not valid.

Programmer Response: Specify a valid long timer value.

4058 (X'0FDA') MQRCCF_LONG_TIMER_WRONG_TYPE

Explanation: Long timer parameter not allowed for this channel type.

The `LongRetryInterval` parameter is only allowed for sender or server channel types.

Programmer Response: Remove the parameter.

3044 (X'0BE4') MQRCCF_MAX_MSG_LENGTH_ERROR

Explanation: Maximum message length not valid.

The maximum message length value specified was not valid.

Programmer Response: Specify a valid maximum message length.

4047 (X'0FCF') MQRCCF_MCA_NAME_ERROR

Explanation: Message channel agent name error.

The `MCAName` value contained characters that are not allowed for program names on the platform in question.

Programmer Response: Specify a valid name.

4053 (X'0FD5') MQRCCF_MCA_NAME_WRONG_TYPE

Explanation: Message channel agent name not allowed for this channel type.

The `MCAName` parameter is only allowed for sender, server or requester channel types.

Programmer Response: Remove the parameter.

3063 (X'0BF7') MQRCCF_MCA_TYPE_ERROR

Explanation: Message channel agent type not valid.

The `MCAType` value specified was not valid.

Programmer Response: Specify a valid value.

3023 (X'0BCF') MQRCCF_MD_FORMAT_ERROR

Explanation: Format not valid.

The MQMD `Format` field value was not MQFMT_ADMIN.

Programmer Response: Specify the valid format.

4061 (X'0FDD') MQRCCF_MISSING_CONN_NAME

Explanation: Connection name parameter required but missing.

The `ConnectionName` parameter is required for sender or requester channel types, but is not present.

Programmer Response: Add the parameter.

3029 (X'0BD5') MQRCCF_MODE_VALUE_ERROR

Explanation: Mode value not valid.

The `Mode` value was not valid.

Programmer Response: Specify a valid mode value.

4026 (X'0FBA') MQRCCF_MQCONN_FAILED

Explanation: MQCONN call failed.

Programmer Response: Check whether the queue manager is active.

4028 (X'0FBC') MQRCCF_MQGET_FAILED

Explanation: MQGET call failed.

Programmer Response: Check whether the queue manager is active, and the queues involved are correctly set up, and enabled for MQGET.

4036 (X'0FC4') MQRCCF_MQINQ_FAILED

Explanation: MQINQ call failed.

Programmer Response: Check whether the queue manager is active.

4027 (X'0FBB') MQRCCF_MQOPEN_FAILED

Explanation: MQOPEN call failed.

Programmer Response: Check whether the queue manager is active, and the queues involved are correctly set up.

4029 (X'0FBD') MQRCCF_MQPUT_FAILED

Explanation: MQPUT call failed.

Programmer Response: Check whether the queue manager is active, and the queues involved are correctly set up, and not inhibited for puts.

4063 (X'0FDF') MQRCCF_MQSET_FAILED

Explanation: MQSET call failed.

Programmer Response: Check whether the queue manager is active.

4069 (X'0FE5') MQRCCF_MR_COUNT_ERROR

Explanation: Message retry count not valid.

The *MsgRetryCount* value was not valid.

Programmer Response: Specify a value in the range 0-999 999 999.

4070 (X'0FE6') MQRCCF_MR_COUNT_WRONG_TYPE

Explanation: Message-retry count parameter not allowed for this channel type.

The *MsgRetryCount* parameter is allowed only for receiver and requester channels.

Programmer Response: Remove the parameter.

4071 (X'0FE7') MQRCCF_MR_EXIT_NAME_ERROR

Explanation: Channel message-retry exit name error.

The *MsgRetryExit* value contained characters that are not allowed for program names on the platform in question.

Programmer Response: Specify a valid name.

4072 (X'0FE8') MQRCCF_MR_EXIT_NAME_WRONG_TYPE

Explanation: Message-retry exit parameter not allowed for this channel type.

The *MsgRetryExit* parameter is allowed only for receiver and requester channels.

Programmer Response: Remove the parameter.

4073 (X'0FE9') MQRCCF_MR_INTERVAL_ERROR

Explanation: Message retry interval not valid.

The *MsgRetryInterval* value was not valid.

Programmer Response: Specify a value in the range 0-999 999 999.

4074 (X'0FEA') MQRCCF_MR_INTERVAL_WRONG_TYPE

Explanation: Message-retry interval parameter not allowed for this channel type.

The *MsgRetryInterval* parameter is allowed only for receiver and requester channels.

Programmer Response: Remove the parameter.

4050 (X'0FD2') MQRCCF_MSG_EXIT_NAME_ERROR

Explanation: Channel message exit name error.

The *MsgExit* value contained characters that are not allowed for program names on the platform in question.

Programmer Response: Specify a valid name.

3016 (X'0BC8') MQRCCF_MSG_LENGTH_ERROR

Explanation: 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.

Programmer Response: Specify a valid message length, and check that positional parameters are in the correct sequence.

3030 (X'0BD6') MQRCCF_MSG_SEQ_NUMBER_ERROR

Explanation: Message sequence number not valid.

The message sequence number parameter value was not valid.

Programmer Response: Specify a valid message sequence number.

3048 (X'0BE8') MQRCCF_MSG_TRUNCATED

Explanation: Message truncated.

The command server received a message that is larger than its maximum valid message size.

Programmer Response: Check the message contents are correct.

4088 (X'0FF8') MQRCCF_NET_PRIORITY_ERROR

Explanation: Network priority value is not valid.

Programmer Response: Specify a valid value.

4089 (X'0FF9') MQRCCF_NET_PRIORITY_WRONG_TYPE

Explanation: Network priority parameter not allowed for this channel type.

The *NetworkPriority* parameter is allowed for sender and server channels only.

Programmer Response: Remove the parameter.

3093 (X'0C15') MQRCCF_NETBIOS_NAME_ERROR

Explanation: NetBIOS listener name not defined.

The NetBIOS listener name is not defined.

Programmer Response: Add a local name to the configuration file and retry the operation.

4019 (X'0FB3') MQRCCF_NO_COMMS_MANAGER

Explanation: Communications manager not available.

The communications subsystem is not available.

Programmer Response: Ensure that the communications subsystem has been started.

4018 (X'0FB2') MQRCCF_NO_STORAGE

Explanation: Not enough storage available.

Insufficient storage is available.

Programmer Response: Consult your systems administrator.

4037 (X'0FC5') MQRCCF_NOT_XMIT_Q

Explanation: Queue is not a transmission queue.

The queue specified in the channel definition is not a transmission queue.

Programmer Response: Ensure that the queue is specified correctly in the channel definition, and that it is correctly defined to the queue manager.

4075 (X'0FEB') MQRCCF_NPM_SPEED_ERROR

Explanation: Nonpersistent message speed not valid.

The *NonPersistentMsgSpeed* value was not valid.

Programmer Response: Specify MQNPMS_NORMAL or MQNPMS_FAST.

4076 (X'0FEC') MQRCCF_NPM_SPEED_WRONG_TYPE

Explanation: Nonpersistent message speed parameter not allowed for this channel type.

The *NonPersistentMsgSpeed* parameter is allowed only for sender, receiver, server, requester, cluster sender, and cluster receiver channels.

Programmer Response: Remove the parameter.

4001 (X'0FA1') MQRCCF_OBJECT_ALREADY_EXISTS

Explanation: 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.

Programmer Response: Specify *Replace* as MQRP_YES, or use a different name for the object to be created.

| **3160 (X'0C58') MQRCCF_OBJECT_IN_USE**

| **Explanation:** Object in use by another command.

| A modification of an object was attempted while the
| object was being modified by another command.

| **Programmer Response:** Retry the command.

4008 (X'0FA8') MQRCCF_OBJECT_NAME_ERROR

Explanation: Object name not valid.

An object name was specified using characters that were not valid.

Programmer Response: Specify only valid characters for the name.

4004 (X'0FA4') MQRCCF_OBJECT_OPEN

Explanation: Object is open.

An attempt was made to delete or change an object that was in use.

Programmer Response: Wait until the object is not in use, and then retry the operation. Alternatively specify *Force* as MQFC_YES for a change command.

4002 (X'0FA2') MQRCCF_OBJECT_WRONG_TYPE

Explanation: Object has wrong type.

An attempt was made to replace a queue object with one of a different type.

Programmer Response: Ensure that the new object is the same type as the one it is replacing.

3020 (X'0BCC') MQRCCF_PARM_COUNT_TOO_BIG

Explanation: Parameter count too big.

The MQCFH *ParameterCount* field value was more than the maximum for the command.

Programmer Response: Specify a parameter count that is valid for the command.

3019 (X'0BCB') MQRCCF_PARM_COUNT_TOO_SMALL

Explanation: Parameter count too small.

The MQCFH *ParameterCount* field value was less than the minimum required for the command.

Programmer Response: Specify a parameter count that is valid for the command.

3035 (X'0BDB') MQRCCF_PARM_SEQUENCE_ERROR

Explanation: Parameter sequence not valid.

The sequence of parameters is not valid for this command.

Programmer Response: Specify the positional parameters in a valid sequence for the command.

3097 (X'0C19') MQRCCF_PARM_SYNTAX_ERROR

Explanation: Syntax error found in parameter.

The parameter specified contained a syntax error.

Programmer Response: Check the syntax for this parameter.

3096 (X'0C18') MQRCCF_PATH_NOT_VALID

Explanation: Path not valid.

The path specified was not valid.

Programmer Response: Specify a valid path.

3032 (X'0BD8') MQRCCF_PING_DATA_COMPARE_ERROR

Explanation: 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).

Programmer Response: Consult your systems administrator.

3031 (X'0BD7') MQRCCF_PING_DATA_COUNT_ERROR

Explanation: Data count not valid.

The Ping Channel *DataCount* value was not valid.

Programmer Response: Specify a valid data count value.

4030 (X'0FBE') MQRCCF_PING_ERROR

Explanation: 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.

Programmer Response: Reissue the ping request for a different channel of the correct type, or for a receiver channel from a different queue manager.

| **3167 (X'0C5F') MQRCCF_PORT_NUMBER_ERROR**

| **Explanation:** Port number value not valid.

| The *PortNumber* value was not valid.

| **Programmer Response:** Specify a valid port number
| value.

3046 (X'0BE6') MQRCCF_PURGE_VALUE_ERROR

Explanation: Purge value not valid.

The *Purge* value was not valid.

Programmer Response: Specify a valid purge value.

3045 (X'0BE5') MQRCCF_PUT_AUTH_ERROR

Explanation: Put authority value not valid.

The *PutAuthority* value was not valid.

Programmer Response: Specify a valid authority value.

4059 (X'0FDB') MQRCCF_PUT_AUTH_WRONG_TYPE

Explanation: Put authority parameter not allowed for this channel type.

The *PutAuthority* parameter is only allowed for receiver or requester channel types.

Programmer Response: Remove the parameter.

3098 (X'0C1A') MQRCCF_PWD_LENGTH_ERROR

Explanation: Password length error.

The password string length is rounded up by to the nearest eight bytes. This rounding causes the total length of the *SSLCryptoHardware* string to exceed its maximum.

Programmer Response: Decrease the size of the password, or of earlier fields in the *SSLCryptoHardware* string.

3021 (X'0BCD') MQRCCF_Q_ALREADY_IN_CELL

Explanation: 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.

Programmer Response: 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.

3086 (X'0C0E') MQRCCF_Q_MGR_CCSID_ERROR

Explanation: Queue manager coded character set identifier error.

The coded character set value for the queue manager was not valid.

Programmer Response: Specify a valid value.

3022 (X'0BCE') MQRCCF_Q_TYPE_ERROR

Explanation: Queue type not valid.

The *QType* value was not valid.

Programmer Response: Specify a valid queue type.

4007 (X'0FA7') MQRCCF_Q_WRONG_TYPE

Explanation: Action not valid for the queue of specified type.

An attempt was made to perform an action on a queue of the wrong type.

Programmer Response: Specify a queue of the correct type.

3029 (X'0BD5') MQRCCF_QUIESCE_VALUE_ERROR

Explanation: Former name for MQRCCF_MODE_VALUE_ERROR.

4051 (X'0FD3') MQRCCF_RCV_EXIT_NAME_ERROR

Explanation: Channel receive exit name error.

The *ReceiveExit* value contained characters that are not allowed for program names on the platform in question.

Programmer Response: Specify a valid name.

4016 (X'0FB0') MQRCCF_RECEIVE_FAILED

Explanation: Receive failed.

The receive operation failed.

Programmer Response: Correct the error and retry the operation.

4015 (X'0FAF') MQRCCF_RECEIVED_DATA_ERROR

Explanation: Received data error.

An error occurred while receiving data from a remote system. This might be caused by a communications failure.

Programmer Response: Consult your systems administrator.

4035 (X'0FC3') MQRCCF_REMOTE_QM_TERMINATING

Explanation: Remote queue manager terminating.

The channel is ending because the remote queue manager is terminating.

Programmer Response: Restart the remote queue manager.

4034 (X'0FC2') MQRCCF_REMOTE_QM_UNAVAILABLE

Explanation: Remote queue manager not available.

The channel cannot be started because the remote queue manager is not available.

Programmer Response: Start the remote queue manager.

3025 (X'0BD1') MQRCCF_REPLACE_VALUE_ERROR

Explanation: Replace value not valid.

The *Replace* value was not valid.

Programmer Response: Specify a valid replace value.

3089 (X'0C11') MQRCCF_REPOS_NAME_CONFLICT

Explanation: *RepositoryName* and *RepositoryNamelist* attributes conflict.

The command was rejected because it would have resulted in the *RepositoryName* and *RepositoryNamelist* attributes both being nonblank. At least one of these attributes must be blank.

Programmer Response: If the command specified only one of these attributes, specify the other as well, but with a value of blanks. If the command specified both attributes, ensure that one of them has a value of blanks.

4049 (X'0FD1') MQRCCF_SEC_EXIT_NAME_ERROR

Explanation: Channel security exit name error.

The *SecurityExit* value contained characters that are not allowed for program names on the platform in question.

Programmer Response: Specify a valid name.

4048 (X'0FD0') MQRCCF_SEND_EXIT_NAME_ERROR

Explanation: Channel send exit name error.

The *SendExit* value contained characters that are not allowed for program names on the platform in question.

Programmer Response: Specify a valid name.

4014 (X'0FAE') MQRCCF_SEND_FAILED

Explanation: Send failed.

An error occurred while sending data to a remote system. This might be caused by a communications failure.

Programmer Response: Consult your systems administrator.

3043 (X'0BE3') MQRCCF_SEQ_NUMBER_WRAP_ERROR

Explanation: Sequence wrap number not valid.

The *SeqNumberWrap* value was not valid.

Programmer Response: Specify a valid sequence wrap number.

3039 (X'0BDF') MQRCCF_SHORT_RETRY_ERROR

Explanation: Short retry count not valid.

The *ShortRetryCount* value was not valid.

Programmer Response: Specify a valid short retry count value.

4055 (X'0FD7') MQRCCF_SHORT_RETRY_WRONG_TYPE

Explanation: Short retry parameter not allowed for this channel type.

The *ShortRetryCount* parameter is only allowed for sender or server channel types.

Programmer Response: Remove the parameter.

3040 (X'0BE0') MQRCCF_SHORT_TIMER_ERROR

Explanation: Short timer value not valid.

The *ShortRetryInterval* value was not valid.

Programmer Response: Specify a valid short timer value.

4056 (X'0FD8') MQRCCF_SHORT_TIMER_WRONG_TYPE

Explanation: Short timer parameter not allowed for this channel type.

The *ShortRetryInterval* parameter is only allowed for sender or server channel types.

Programmer Response: Remove the parameter.

4092 (X'0FFC') MQRCCF_SSL_CIPHER_SPEC_ERROR

Explanation: SSL cipher specification not valid.

The *SSLCipherSpec* specified is not valid.

Programmer Response: Specify a valid cipher specification.

4094 (X'0FFE') MQRCCF_SSL_CLIENT_AUTH_ERROR

Explanation: SSL client authentication not valid.

The *SSLClientAuth* specified is not valid.

Programmer Response: Specify a valid client authentication.

4093 (X'0FFD') MQRCCF_SSL_PEER_NAME_ERROR

Explanation: SSL peer name not valid.

The *SSLPeerName* specified is not valid.

Programmer Response: Specify a valid peer name.

3013 (X'0BC5') MQRCCF_STRUCTURE_TYPE_ERROR

Explanation: Structure type not valid.

The structure *Type* value was not valid.

Programmer Response: Specify a valid structure type.

4085 (X'0FF5') MQRCCF_SUPPRESSED_BY_EXIT

Explanation: 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.

Programmer Response: Examine the value of the *AuxErrorDataInt1* parameter, and take any action that is appropriate.

4065 (X'0FE1') MQRCCF_TERMINATED_BY_SEC_EXIT

Explanation: Channel terminated by security exit.

A channel security exit terminated the channel.

Programmer Response: 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.

| **3161 (X'0C59') MQRCCF_UNKNOWN_FILE_NAME**

| **Explanation:** File not defined to CICS.

| A file name parameter identifies a file that is not
| defined to CICS.

| **Programmer Response:** Provide a valid file name or
| create a CSD definition for the required file.

4006 (X'0FA6') MQRCCF_UNKNOWN_Q_MGR

Explanation: Queue manager not known.

The queue manager specified was not known.

Programmer Response: Specify the name of the queue manager to which the command is sent, or blank.

4033 (X'0FC1') MQRCCF_UNKNOWN_REMOTE_CHANNEL

Explanation: Remote channel not known.

There is no definition of the referenced channel at the remote system.

Programmer Response: Ensure that the local channel is correctly defined. If it is, add an appropriate channel definition at the remote system.

4039 (X'0FC7') MQRCCF_USER_EXIT_NOT_AVAILABLE

Explanation: User exit not available.

The channel was terminated because the user exit specified does not exist.

Programmer Response: Ensure that the user exit is correctly specified and the program is available.

4041 (X'0FC9') MQRCCF_WRONG_CHANNEL_TYPE

Explanation: Parameter not allowed for this channel type.

The parameter is not allowed for the type of channel being created, copied, or changed. Refer to the description of the parameter in error to determine the types of channel for which the parameter is valid

Programmer Response: Remove the parameter.

3151 (X'0C4F') MQRCCF_WRONG_USER

Explanation: Wrong user.

A publish/subscribe command message cannot be executed on behalf of the requesting user because the subscription that it would update is already owned by a different user. A subscription can be updated or deregistered only by the user that originally registered the subscription.

Programmer Response: Ensure that applications that need to issue commands against existing subscriptions are running under the user identifier that originally registered the subscription. Alternatively, use different subscriptions for different users.

3036 (X'0BDC') MQRCCF_XMIT_PROTOCOL_TYPE_ERR

Explanation: Transmission protocol type not valid.

The *TransportType* value was not valid.

Programmer Response: Specify a valid transmission protocol type.

Error codes

4045 (X'0FCD') MQRCCF_XMIT_Q_NAME_ERROR

Explanation: 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.

Programmer Response: Specify a valid name, or add the parameter.

4052 (X'0FD4') MQRCCF_XMIT_Q_NAME_WRONG_TYPE

Explanation: Transmission queue name not allowed for this channel type.

The *XmitQName* parameter is only allowed for sender or server channel types.

Programmer Response: Remove the parameter.

Appendix B. MQ constants

This appendix specifies the values of the named constants that apply to PCF commands and responses.

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`*`xxxxx`*`_`, where *`xxxxx`* represents a string of 0 through 5 characters that indicates the parameter or field to which the values relate. The constants are ordered alphabetically by this prefix.

Notes:

1. For constants with numeric values, the values are shown in both decimal and hexadecimal forms.
2. Hexadecimal values are represented using the notation `X'hhhh'`, where each `h` denotes a single hexadecimal digit.
3. Character values are shown delimited by single quotation marks; the quotation marks are not part of the value.
4. Blanks in character values are represented by one or more occurrences of the symbol ƀ.
5. If the value is shown as `(variable)`, it indicates that the value of the constant depends on the environment in which the application is running.

List of constants

The following sections list all of the named constants that are mentioned in this book, and show their values.

MQ_* (Lengths of character string and byte fields)

MQ_APPL_NAME_LENGTH	28	X'0000001C'
MQ_APPL_TAG_LENGTH	28	X'0000001C'
MQ_AUTH_INFO_CONN_NAME_LENGTH	264	X'00000108'
MQ_AUTH_INFO_DESC_LENGTH	64	X'00000040'
MQ_AUTH_INFO_NAME_LENGTH	48	X'00000030'
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_CLUSTER_NAME_LENGTH	48	X'00000030'
MQ_CONN_NAME_LENGTH	264	X'00000108'
MQ_CREATION_DATE_LENGTH	12	X'0000000C'
MQ_CREATION_TIME_LENGTH	8	X'00000008'
MQ_DATE_LENGTH	12	X'0000000C'
MQ_DISTINGUISHED_NAME_LENGTH	1024	X'00000400'
MQ_EXIT_DATA_LENGTH	32	X'00000020'
MQ_EXIT_NAME_LENGTH	(variable)	
MQ_FORMAT_LENGTH	8	X'00000008'
MQ_LDAP_PASSWORD_LENGTH	32	X'00000020'
MQ_LOCAL_ADDRESS_LENGTH	48	X'00000030'
MQ_LUWID_LENGTH	16	X'00000010'
MQ_MAX_EXIT_NAME_LENGTH	128	X'00000080'

MQ_MAX_MCA_USER_ID_LENGTH	64	X'00000040'
MQ_MAX_USER_ID_LENGTH	64	X'00000040'
MQ_MCA_JOB_NAME_LENGTH	28	X'0000001C'
MQ_MCA_NAME_LENGTH	20	X'00000014'
MQ_MCA_USER_ID_LENGTH	(variable)	
MQ_MODE_NAME_LENGTH	8	X'00000008'
MQ_NAMELIST_DESC_LENGTH	64	X'00000040'
MQ_NAMELIST_NAME_LENGTH	48	X'00000030'
MQ_OBJECT_NAME_LENGTH	48	X'00000030'
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_Q_DESC_LENGTH	64	X'00000040'
MQ_Q_MGR_DESC_LENGTH	64	X'00000040'
MQ_Q_MGR_IDENTIFIER_LENGTH	48	X'00000030'
MQ_Q_MGR_NAME_LENGTH	48	X'00000030'
MQ_Q_NAME_LENGTH	48	X'00000030'
MQ_SECURITY_ID_LENGTH	40	X'00000028'
MQ_SSL_CIPHER_SPEC_LENGTH	32	X'00000020'
MQ_SSL_CRYPTO_HARDWARE_LENGTH	256	X'00000100'
MQ_SSL_HANDSHAKE_STAGE_LENGTH	32	X'00000020'
MQ_SSL_KEY_REPOSITORY_LENGTH	256	X'00000100'
MQ_SSL_SHORT_PEER_NAME_LENGTH	256	X'00000100'
MQ_TIME_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_TRIGGER_DATA_LENGTH	64	X'00000040'
MQ_USER_ID_LENGTH	12	X'0000000C'

MQACT_* (Action option)

MQACT_FORCE_REMOVE	1	X'00000001'

MQAIT_* (Authentication information type)

MQAIT_CRL_LDAP	1	X'00000001'

MQAT_*

MQAT_QMGR		X''
MQAT_CHANNEL_INITIATOR		X''
MQAT_USER		X''

MQCA_* (Character attribute selector)

MQCA_FIRST	2001	X'000007D1'
MQCA_APPL_ID	2001	X'000007D1'
MQCA_BASE_Q_NAME	2002	X'000007D2'

MQCA_COMMAND_INPUT_Q_NAME	2003	X'000007D3'
MQCA_CREATION_DATE	2004	X'000007D4'
MQCA_CREATION_TIME	2005	X'000007D5'
MQCA_DEAD_LETTER_Q_NAME	2006	X'000007D6'
MQCA_ENV_DATA	2007	X'000007D7'
MQCA_INITIATION_Q_NAME	2008	X'000007D8'
MQCA_NAMELIST_DESC	2009	X'000007D9'
MQCA_NAMELIST_NAME	2010	X'000007DA'
MQCA_PROCESS_DESC	2011	X'000007DB'
MQCA_PROCESS_NAME	2012	X'000007DC'
MQCA_Q_DESC	2013	X'000007DD'
MQCA_Q_MGR_DESC	2014	X'000007DE'
MQCA_Q_MGR_NAME	2015	X'000007DF'
MQCA_Q_NAME	2016	X'000007E0'
MQCA_REMOTE_Q_MGR_NAME	2017	X'000007E1'
MQCA_REMOTE_Q_NAME	2018	X'000007E2'
MQCA_BACKOUT_REQ_Q_NAME	2019	X'000007E3'
MQCA_NAMES	2020	X'000007E4'
MQCA_USER_DATA	2021	X'000007E5'
MQCA_STORAGE_CLASS	2022	X'000007E6'
MQCA_TRIGGER_DATA	2023	X'000007E7'
MQCA_XMIT_Q_NAME	2024	X'000007E8'
MQCA_DEF_XMIT_Q_NAME	2025	X'000007E9'
MQCA_CHANNEL_AUTO_DEF_EXIT	2026	X'000007EA'
MQCA_ALTERATION_DATE	2027	X'000007EB'
MQCA_ALTERATION_TIME	2028	X'000007EC'
MQCA_CLUSTER_NAME	2029	X'000007ED'
MQCA_CLUSTER_NAMELIST	2030	X'000007EE'
MQCA_CLUSTER_Q_MGR_NAME	2031	X'000007EF'
MQCA_Q_MGR_IDENTIFIER	2032	X'000007F0'
MQCA_CLUSTER_WORKLOAD_EXIT	2033	X'000007F1'
MQCA_CLUSTER_WORKLOAD_DATA	2034	X'000007F2'
MQCA_REPOSITORY_NAME	2035	X'000007F3'
MQCA_REPOSITORY_NAMELIST	2036	X'000007F4'
MQCA_CLUSTER_DATE	2037	X'000007F5'
MQCA_CLUSTER_TIME	2038	X'000007F6'
MQCA_CF_STRUC_NAME	2039	X'000007F7'
MQCA_QSG_NAME	2040	X'000007F8'
MQCA_IGQ_USER_ID	2041	X'000007F9'
MQCA_STORAGE_CLASS_DESC	2042	X'000007FA'
MQCA_XCF_GROUP_NAME	2043	X'000007FB'
MQCA_XCF_MEMBER_NAME	2044	X'000007FC'
MQCA_AUTH_INFO_NAME	2045	X'000007FD'
MQCA_AUTH_INFO_DESC	2046	X'000007FE'
MQCA_LDAP_USER_NAME	2047	X'000007FF'
MQCA_LDAP_PASSWORD	2048	X'00000800'
MQCA_SSL_KEY_REPOSITORY	2049	X'00000801'
MQCA_SSL_CRL_NAMELIST	2050	X'00000802'
MQCA_SSL_CRYPTO_HARDWARE	2051	X'00000803'
MQCA_CF_STRUC_DESC	2052	X'00000804'
MQCA_AUTH_INFO_CONN_NAME	2053	X'00000805'
MQCA_LAST	4000	X'00000FA0'
MQCA_LAST_USED	(variable)	

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_NAMELIST_NAME	3005	X'00000BBD'
MQCACF_TO_NAMELIST_NAME	3006	X'00000BBE'
MQCACF_FROM_CHANNEL_NAME	3007	X'00000BBF'
MQCACF_TO_CHANNEL_NAME	3008	X'00000BC0'
MQCACF_FROM_AUTH_INFO_NAME	3009	X'00000BC1'
MQCACF_TO_AUTH_INFO_NAME	3010	X'00000BC2'
MQCACF_Q_NAMES	3011	X'00000BC3'
MQCACF_PROCESS_NAMES	3012	X'00000BC4'
MQCACF_NAMELIST_NAMES	3013	X'00000BC5'
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_EVENT_USER_ID	3045	X'00000BE5'
MQCACF_EVENT_Q_MGR	3047	X'00000BE7'
MQCACF_AUTH_INFO_NAMES	3048	X'00000BE8'
MQCACF_EVENT_APPL_IDENTITY	3049	X'00000BE9'
MQCACF_EVENT_APPL_NAME	3050	X'00000BEA'
MQCACF_EVENT_APPL_ORIGIN	3051	X'00000BEB'
MQCACF_APPL_TAG	3058	X'00000BF2'
MQCACF_LAST_USED	(variable)	

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_LOCAL_ADDRESS	3520	X'00000DC0'
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_SSL_CIPHER_SPEC	3544	X'00000DD8'
MQCACH_SSL_PEER_NAME	3545	X'00000DD9'
MQCACH_SSL_HANDSHAKE_STAGE	3546	X'00000DDA'
MQCACH_SSL_SHORT_PEER_NAME	3547	X'00000DDB'
MQCACH_LAST_USED	(variable)	

MQCC_* (Completion code)

MQCC_UNKNOWN	-1	X'FFFFFFFF'
MQCC_OK	0	X'00000000'
MQCC_WARNING	1	X'00000001'
MQCC_FAILED	2	X'00000002'

MQCCSI_* (Coded character set identifier)

MQCCSI_DEFAULT	0	X'00000000'

MQCDC_* (Channel data conversion)

MQCDC_NO_SENDER_CONVERSION	0	X'00000000'
MQCDC_SENDER_CONVERSION	1	X'00000001'

MQCFBS_* (Command format byte string parameter structure length)

MQCFBS_STRUC_LENGTH_FIXED	16	X'00000010'

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_VERSION_2	2	X'00000002'
MQCFH_CURRENT_VERSION	(variable)	

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'
MQCFT_USER	8	X'00000008'
MQCFT_BYTE_STRING	9	X'00000009'

MQCHAD_* (Channel auto-definition)

MQCHAD_DISABLED	0	X'00000000'
MQCHAD_ENABLED	1	X'00000001'

MQCHIDS_* (Channel indoubt status)

MQCHIDS_NOT_INDOUBT	0	X'00000000'
MQCHIDS_INDOUBT	1	X'00000001'

MQCHS_* (Channel status)

MQCHS_INACTIVE	0	X'00000000'
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'

MQCHSR_* (Channel stop requested)

MQCHSR_STOP_NOT_REQUESTED	0	X'00000000'
MQCHSR_STOP_REQUESTED	1	X'00000001'

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'

MQCHTAB_* (Channel table)

MQCHTAB_Q_MGR	1	X'00000001'
MQCHTAB_CLNTCONN	2	X'00000002'

MQCMD_* (Command identifier)

MQCMD_NONE	0	X'00000000'
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_CHANGE_NAMELIST	32	X'00000020'
MQCMD_COPY_NAMELIST	33	X'00000021'
MQCMD_CREATE_NAMELIST	34	X'00000022'
MQCMD_DELETE_NAMELIST	35	X'00000023'
MQCMD_INQUIRE_NAMELIST	36	X'00000024'
MQCMD_INQUIRE_NAMELIST_NAMES	37	X'00000025'
MQCMD_ESCAPE	38	X'00000026'
MQCMD_RESOLVE_CHANNEL	39	X'00000027'
MQCMD_PING_Q_MGR	40	X'00000028'
MQCMD_INQUIRE_Q_STATUS	41	X'00000029'
MQCMD_INQUIRE_CHANNEL_STATUS	42	X'0000002A'
MQCMD_CONFIG_EVENT	43	X'0000002B'
MQCMD_Q_MGR_EVENT	44	X'0000002C'
MQCMD_PERFM_EVENT	45	X'0000002D'
MQCMD_CHANNEL_EVENT	46	X'0000002E'
MQCMD_INQUIRE_CLUSTER_Q_MGR	70	X'00000046'
MQCMD_RESUME_Q_MGR_CLUSTER	71	X'00000047'
MQCMD_SUSPEND_Q_MGR_CLUSTER	72	X'00000048'
MQCMD_REFRESH_CLUSTER	73	X'00000049'
MQCMD_RESET_CLUSTER	74	X'0000004A'
MQCMD_REFRESH_SECURITY	78	X'0000004E'
MQCMD_CHANGE_AUTH_INFO	79	X'0000004F'
MQCMD_COPY_AUTH_INFO	80	X'00000050'
MQCMD_CREATE_AUTH_INFO	81	X'00000051'
MQCMD_DELETE_AUTH_INFO	82	X'00000052'
MQCMD_INQUIRE_AUTH_INFO	83	X'00000053'
MQCMD_INQUIRE_AUTH_INFO_NAMES	84	X'00000054'

MQCMDL_* (Command level)

MQCMDL_LEVEL_1	100	X'00000064'
MQCMDL_LEVEL_101	101	X'00000065'
MQCMDL_LEVEL_110	110	X'0000006E'
MQCMDL_LEVEL_114	114	X'00000072'
MQCMDL_LEVEL_200	200	X'000000C8'
MQCMDL_LEVEL_201	201	X'000000C9'
MQCMDL_LEVEL_220	220	X'000000DC'
MQCMDL_LEVEL_221	221	X'000000DD'
MQCMDL_LEVEL_320	320	X'00000140'
MQCMDL_LEVEL_420	420	X'000001A4'
MQCMDL_LEVEL_500	500	X'000001F4'
MQCMDL_LEVEL_510	510	X'000001FE'
MQCMDL_LEVEL_520	520	X'00000208'
MQCMDL_LEVEL_530	530	X'00000212'

MQCQT_* (Cluster queue type)

MQCQT_LOCAL_Q	1	X'00000001'
MQCQT_ALIAS_Q	2	X'00000002'
MQCQT_REMOTE_Q	3	X'00000003'
MQCQT_Q_MGR_ALIAS	4	X'00000004'

MQET_* (Escape type)

MQET_MQSC	1	X'00000001'

MQEVR_* (Event reporting)

MQEVR_DISABLED	0	X'00000000'
MQEVR_ENABLED	1	X'00000001'

MQFC_* (Force option)

MQFC_NO	0	X'00000000'
MQFC_YES	1	X'00000001'

MQIA_* (Integer attribute selector)

MQIA_FIRST	1	X'00000001'
MQIA_APPL_TYPE	1	X'00000001'
MQIA_CODED_CHAR_SET_ID	2	X'00000002'
MQIA_CURRENT_Q_DEPTH	3	X'00000003'
MQIA_DEF_INPUT_OPEN_OPTION	4	X'00000004'
MQIA_DEF_PERSISTENCE	5	X'00000005'
MQIA_DEF_PRIORITY	6	X'00000006'
MQIA_DEFINITION_TYPE	7	X'00000007'
MQIA_HARDEN_GET_BACKOUT	8	X'00000008'
MQIA_INHIBIT_GET	9	X'00000009'
MQIA_INHIBIT_PUT	10	X'0000000A'

MQIA_MAX_HANDLES	11	X'0000000B'
MQIA_USAGE	12	X'0000000C'
MQIA_MAX_MSG_LENGTH	13	X'0000000D'
MQIA_MAX_PRIORITY	14	X'0000000E'
MQIA_MAX_Q_DEPTH	15	X'0000000F'
MQIA_MSG_DELIVERY_SEQUENCE	16	X'00000010'
MQIA_OPEN_INPUT_COUNT	17	X'00000011'
MQIA_OPEN_OUTPUT_COUNT	18	X'00000012'
MQIA_NAME_COUNT	19	X'00000013'
MQIA_Q_TYPE	20	X'00000014'
MQIA_RETENTION_INTERVAL	21	X'00000015'
MQIA_BACKOUT_THRESHOLD	22	X'00000016'
MQIA_SHAREABILITY	23	X'00000017'
MQIA_TRIGGER_CONTROL	24	X'00000018'
MQIA_TRIGGER_INTERVAL	25	X'00000019'
MQIA_TRIGGER_MSG_PRIORITY	26	X'0000001A'
MQIA_TRIGGER_TYPE	28	X'0000001C'
MQIA_TRIGGER_DEPTH	29	X'0000001D'
MQIA_SYNCPOINT	30	X'0000001E'
MQIA_COMMAND_LEVEL	31	X'0000001F'
MQIA_PLATFORM	32	X'00000020'
MQIA_MAX_UNCOMMITTED_MSGS	33	X'00000021'
MQIA_DIST_LISTS	34	X'00000022'
MQIA_TIME_SINCE_RESET	35	X'00000023'
MQIA_HIGH_Q_DEPTH	36	X'00000024'
MQIA_MSG_ENQ_COUNT	37	X'00000025'
MQIA_MSG_DEQ_COUNT	38	X'00000026'
MQIA_EXPIRY_INTERVAL	39	X'00000027'
MQIA_Q_DEPTH_HIGH_LIMIT	40	X'00000028'
MQIA_Q_DEPTH_LOW_LIMIT	41	X'00000029'
MQIA_Q_DEPTH_MAX_EVENT	42	X'0000002A'
MQIA_Q_DEPTH_HIGH_EVENT	43	X'0000002B'
MQIA_Q_DEPTH_LOW_EVENT	44	X'0000002C'
MQIA_SCOPE	45	X'0000002D'
MQIA_Q_SERVICE_INTERVAL_EVENT	46	X'0000002E'
MQIA_AUTHORITY_EVENT	47	X'0000002F'
MQIA_INHIBIT_EVENT	48	X'00000030'
MQIA_LOCAL_EVENT	49	X'00000031'
MQIA_REMOTE_EVENT	50	X'00000032'
MQIA_CONFIGURATION_EVENT	51	X'00000033'
MQIA_START_STOP_EVENT	52	X'00000034'
MQIA_PERFORMANCE_EVENT	53	X'00000035'
MQIA_Q_SERVICE_INTERVAL	54	X'00000036'
MQIA_CHANNEL_AUTO_DEF	55	X'00000037'
MQIA_CHANNEL_AUTO_DEF_EVENT	56	X'00000038'
MQIA_INDEX_TYPE	57	X'00000039'
MQIA_CLUSTER_WORKLOAD_LENGTH	58	X'0000003A'
MQIA_CLUSTER_Q_TYPE	59	X'0000003B'
MQIA_ARCHIVE	60	X'0000003C'
MQIA_DEF_BIND	61	X'0000003D'
MQIA_PAGESET_ID	62	X'0000003E'
MQIA_QSG_DISP	63	X'0000003F'
MQIA_INTRA_GROUP_QUEUING	64	X'00000040'
MQIA_IGQ_PUT_AUTHORITY	65	X'00000041'
MQIA_AUTH_INFO_TYPE	68	X'00000044'

|	MQIA_SSL_TASKS	69	X'00000045'
|	MQIA_CF_LEVEL	70	X'00000046'
|	MQIA_CF_RECOVER	71	X'00000047'
|	MQIA_NAMELIST_TYPE	72	X'00000048'
	MQIA_LAST	2000	X'000007D0'
	MQIA_LAST_USED	(variable)	

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_NAMELIST_ATTRS	1004	X'000003EC'
	MQIACF_FORCE	1005	X'000003ED'
	MQIACF_REPLACE	1006	X'000003EE'
	MQIACF_PURGE	1007	X'000003EF'
	MQIACF_QUIESCE	1008	X'000003F0'
|	MQIACF_MODE	1008	X'000003F0'
	MQIACF_ALL	1009	X'000003F1'
|	MQIACF_EVENT_APPL_TYPE	1010	X'000003F2'
|	MQIACF_EVENT_ORIGIN	1011	X'000003F3'
	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_OBJECT_TYPE	1016	X'000003F8'
	MQIACF_ESCAPE_TYPE	1017	X'000003F9'
	MQIACF_ERROR_OFFSET	1018	X'000003FA'
|	MQIACF_AUTH_INFO_ATTRS	1019	X'000003FB'
	MQIACF_REASON_QUALIFIER	1020	X'000003FC'
	MQIACF_COMMAND	1021	X'000003FD'
	MQIACF_OPEN_OPTIONS	1022	X'000003FE'
|	MQIACF_OPEN_TYPE	1023	X'000003FF'
|	MQIACF_PROCESS_ID	1024	X'00000400'
|	MQIACF_THREAD_ID	1025	X'00000401'
|	MQIACF_Q_STATUS_ATTRS	1026	X'00000402'
|	MQIACF_UNCOMMITTED_MSGS	1027	X'00000403'
	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_INQUIRY	1074	X'00000432'
	MQIACF_WAIT_INTERVAL	1075	X'00000433'
	MQIACF_CLUSTER_INFO	1083	X'0000043B'
	MQIACF_Q_MGR_DEFINITION_TYPE	1084	X'0000043C'
	MQIACF_Q_MGR_TYPE	1085	X'0000043D'
	MQIACF_ACTION	1086	X'0000043E'
	MQIACF_SUSPEND	1087	X'0000043F'
	MQIACF_CLUSTER_Q_MGR_ATTRS	1093	X'00000445'
	MQIACF_LAST_USED	(variable)	
|	MQIACF_REFRESH_REPOSITORY	1095	X'00000447'
|	MQIACF_REMOVE_QUEUES	1096	X'00000448'

MQIACF_OPEN_INPUT_TYPE	1098	X'0000044A'
MQIACF_OPEN_OUTPUT	1099	X'0000044B'
MQIACF_OPEN_SET	1100	X'0000044C'
MQIACF_OPEN_INQUIRE	1101	X'0000044D'
MQIACF_OPEN_BROWSE	1102	X'0000044E'
MQIACF_Q_STATUS_TYPE	1103	X'0000044F'
MQIACF_Q_HANDLE	1104	X'00000450'
MQIACF_Q_STATUS	1105	X'00000451'

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_SSL_RETURN_CODE	1533	X'000005FD'
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'

MQIACH_NETWORK_PRIORITY	1565	X'0000061D'
MQIACH_BATCH_HB	1567	X'0000061F'
MQIACH_SSL_CLIENT_AUTH	1568	X'00000620'
MQIACH_LAST_USED	(variable)	

MQIDO_* (Indoubt resolution)

MQIDO_COMMIT	1	X'00000001'
MQIDO_BACKOUT	2	X'00000002'

MQMCAS_* (MCA status)

MQMCAS_STOPPED	0	X'00000000'
MQMCAS_RUNNING	3	X'00000003'

MQMODE_* (Mode option)

MQMODE_FORCE	0	X'00000000'
MQMODE_QUIESCE	1	X'00000001'
MQMODE_TERMINATE	2	X'00000002'

MQNT_* (Namelist type)

MQNT_NONE	0	X'00000000'
MQNT_Q	1	X'00000001'
MQNT_CLUSTER	2	X'00000002'
MQNT_AUTH_INFO	4	X'00000004'
MQNT_ALL	1001	X'000003E9'

MQNPMS_* (Nonpersistent message speed)

MQNPMS_NORMAL	1	X'00000001'
MQNPMS_FAST	2	X'00000002'

MQOT_* (Object type)

MQOT_Q	1	X'00000001'
MQOT_NAMELIST	2	X'00000002'
MQOT_PROCESS	3	X'00000003'
MQOT_STORAGE_CLASS	4	X'00000004'
MQOT_Q_MGR	5	X'00000005'
MQOT_CHANNEL	6	X'00000006'
MQOT_AUTH_INFO	7	X'00000007'
MQOT_CF_STRUC	10	X'0000000A'
MQOT_RESERVED_1	999	X'000003E7'
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'
MQOT_SVRCONN_CHANNEL	1013	X'000003F5'
MQOT_CLNTCONN_CHANNEL	1014	X'000003F6'

MQPL_* (Platform)

MQPL_OS2	2	X'00000002'
MQPL_AIX	3	X'00000003'
MQPL_UNIX	3	X'00000003'
MQPL_OS400	4	X'00000004'
MQPL_WINDOWS_NT	11	X'0000000B'

MQPO_* (Purge option)

MQPO_NO	0	X'00000000'
MQPO_YES	1	X'00000001'

MQQMDT_* (Queue-manager definition type)

MQQMDT_EXPLICIT_CLUSTER_SENDER	1	X'00000001'
MQQMDT_AUTO_CLUSTER_SENDER	2	X'00000002'
MQQMDT_CLUSTER_RECEIVER	3	X'00000003'
MQQMDT_AUTO_EXP_CLUSTER_SENDER	4	X'00000004'

MQQMT_* (Queue-manager type)

MQQMT_NORMAL	0	X'00000000'
MQQMT_REPOSITORY	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'

MQQSOT_* (Queue status open type)

MQQSOT_ALL	X''
MQQSOT_INPUT	X''
MQQSOT_OUTPUT	X''

MQQSUM_* (Queue status uncommited messages)

MQQSUM_YES	X''
MQQSUM_NO	X''

MQQSO_* (Queue status open options)

MQQSO_YES	0
MQQSO_NO	1
MQQSO_SHARED	1
MQQSO_EXCLUSIVE	2

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)

For an alphabetic listing of these codes, with a complete description of each, including suggested responses, see Appendix A, “Error codes”, on page 341. Note: the following list is in **numeric order**.

3001 (X'0BB9')	MQRCCF_CFH_TYPE_ERROR
3002 (X'0BBA')	MQRCCF_CFH_LENGTH_ERROR
3003 (X'0BBB')	MQRCCF_CFH_VERSION_ERROR
3004 (X'0BBC')	MQRCCF_CFH_MSG_SEQ_NUMBER_ERR
3005 (X'0BBD')	MQRCCF_CFH_CONTROL_ERROR
3006 (X'0BBE')	MQRCCF_CFH_PARM_COUNT_ERROR
3007 (X'0BBF')	MQRCCF_CFH_COMMAND_ERROR
3008 (X'0BC0')	MQRCCF_COMMAND_FAILED
3009 (X'0BC1')	MQRCCF_CFIN_LENGTH_ERROR
3010 (X'0BC2')	MQRCCF_CFST_LENGTH_ERROR
3011 (X'0BC3')	MQRCCF_CFST_STRING_LENGTH_ERR
3012 (X'0BC4')	MQRCCF_FORCE_VALUE_ERROR
3013 (X'0BC5')	MQRCCF_STRUCTURE_TYPE_ERROR
3014 (X'0BC6')	MQRCCF_CFIN_PARM_ID_ERROR
3015 (X'0BC7')	MQRCCF_CFST_PARM_ID_ERROR
3016 (X'0BC8')	MQRCCF_MSG_LENGTH_ERROR
3017 (X'0BC9')	MQRCCF_CFIN_DUPLICATE_PARM
3018 (X'0BCA')	MQRCCF_CFST_DUPLICATE_PARM
3019 (X'0BCB')	MQRCCF_PARM_COUNT_TOO_SMALL

3020 (X'0BCC')	MQRCCF_PARM_COUNT_TOO_BIG
3021 (X'0BCD')	MQRCCF_Q_ALREADY_IN_CELL
3022 (X'0BCE')	MQRCCF_Q_TYPE_ERROR
3023 (X'0BCF')	MQRCCF_MD_FORMAT_ERROR
3024 (X'0BD0')	MQRCCF_CFSL_LENGTH_ERROR
3025 (X'0BD1')	MQRCCF_REPLACE_VALUE_ERROR
3026 (X'0BD2')	MQRCCF_CFIL_DUPLICATE_VALUE
3027 (X'0BD3')	MQRCCF_CFIL_COUNT_ERROR
3028 (X'0BD4')	MQRCCF_CFIL_LENGTH_ERROR
3029 (X'0BD5')	MQRCCF_MODE_VALUE_ERROR
3029 (X'0BD5')	MQRCCF_QUIESCE_VALUE_ERROR
3030 (X'0BD6')	MQRCCF_MSG_SEQ_NUMBER_ERROR
3031 (X'0BD7')	MQRCCF_PING_DATA_COUNT_ERROR
3032 (X'0BD8')	MQRCCF_PING_DATA_COMPARE_ERROR
3033 (X'0BD9')	MQRCCF_CFSL_PARM_ID_ERROR
3034 (X'0BDA')	MQRCCF_CHANNEL_TYPE_ERROR
3035 (X'0BDB')	MQRCCF_PARM_SEQUENCE_ERROR
3036 (X'0BDC')	MQRCCF_XMIT_PROTOCOL_TYPE_ERR
3037 (X'0BDD')	MQRCCF_BATCH_SIZE_ERROR
3038 (X'0BDE')	MQRCCF_DISC_INT_ERROR
3039 (X'0BDF')	MQRCCF_SHORT_RETRY_ERROR
3040 (X'0BE0')	MQRCCF_SHORT_TIMER_ERROR
3041 (X'0BE1')	MQRCCF_LONG_RETRY_ERROR
3042 (X'0BE2')	MQRCCF_LONG_TIMER_ERROR
3043 (X'0BE3')	MQRCCF_SEQ_NUMBER_WRAP_ERROR
3044 (X'0BE4')	MQRCCF_MAX_MSG_LENGTH_ERROR
3045 (X'0BE5')	MQRCCF_PUT_AUTH_ERROR
3046 (X'0BE6')	MQRCCF_PURGE_VALUE_ERROR
3047 (X'0BE7')	MQRCCF_CFIL_PARM_ID_ERROR
3048 (X'0BE8')	MQRCCF_MSG_TRUNCATED
3049 (X'0BE9')	MQRCCF_CCSID_ERROR
3050 (X'0BEA')	MQRCCF_ENCODING_ERROR
3052 (X'0BEC')	MQRCCF_DATA_CONV_VALUE_ERROR
3053 (X'0BED')	MQRCCF_INDOUBT_VALUE_ERROR
3054 (X'0BEE')	MQRCCF_ESCAPE_TYPE_ERROR
3062 (X'0BF6')	MQRCCF_CHANNEL_TABLE_ERROR
3063 (X'0BF7')	MQRCCF_MCA_TYPE_ERROR
3064 (X'0BF8')	MQRCCF_CHL_INST_TYPE_ERROR
3065 (X'0BF9')	MQRCCF_CHL_STATUS_NOT_FOUND
3066 (X'0BFA')	MQRCCF_CFSL_DUPLICATE_PARM
3067 (X'0BFB')	MQRCCF_CFSL_TOTAL_LENGTH_ERROR
3068 (X'0BFC')	MQRCCF_CFSL_COUNT_ERROR
3069 (X'0BFD')	MQRCCF_CFSL_STRING_LENGTH_ERR
3086 (X'0C0E')	MQRCCF_Q_MGR_CCSID_ERROR
3088 (X'0C10')	MQRCCF_CLUSTER_NAME_CONFLICT
3089 (X'0C11')	MQRCCF_REPOS_NAME_CONFLICT
3090 (X'0C12')	MQRCCF_CLUSTER_Q_USAGE_ERROR
3091 (X'0C13')	MQRCCF_ACTION_VALUE_ERROR
3092 (X'0C14')	MQRCCF_COMMS_LIBRARY_ERROR
3093 (X'0C15')	MQRCCF_NETBIOS_NAME_ERROR
3095 (X'0C17')	MQRCCF_CFST_CONFLICTING_PARM
3096 (X'0C18')	MQRCCF_PATH_NOT_VALID
3097 (X'0C19')	MQRCCF_PARM_SYNTAX_ERROR
3098 (X'0C1A')	MQRCCF_PWD_LENGTH_ERROR
3150 (X'0C4E')	MQRCCF_FILTER_ERROR
3151 (X'0C4F')	MQRCCF_WRONG_USER

3160 (X'0C58')	MQRCCF_OBJECT_IN_USE
3161 (X'0C59')	MQRCCF_UNKNOWN_FILE_NAME
3162 (X'0C5A')	MQRCCF_FILE_NOT_AVAILABLE
3163 (X'0C5B')	MQRCCF_DISC_RETRY_ERROR
3164 (X'0C5C')	MQRCCF_ALLOC_RETRY_ERROR
3165 (X'0C5D')	MQRCCF_ALLOC_SLOW_TIMER_ERROR
3166 (X'0C5E')	MQRCCF_ALLOC_FAST_TIMER_ERROR
3167 (X'0C5F')	MQRCCF_PORT_NUMBER_ERROR
3168 (X'0C60')	MQRCCF_CHL_SYSTEM_NOT_ACTIVE
4001 (X'0FA1')	MQRCCF_OBJECT_ALREADY_EXISTS
4002 (X'0FA2')	MQRCCF_OBJECT_WRONG_TYPE
4003 (X'0FA3')	MQRCCF_LIKE_OBJECT_WRONG_TYPE
4004 (X'0FA4')	MQRCCF_OBJECT_OPEN
4005 (X'0FA5')	MQRCCF_ATTR_VALUE_ERROR
4006 (X'0FA6')	MQRCCF_UNKNOWN_Q_MGR
4007 (X'0FA7')	MQRCCF_Q_WRONG_TYPE
4008 (X'0FA8')	MQRCCF_OBJECT_NAME_ERROR
4009 (X'0FA9')	MQRCCF_ALLOCATE_FAILED
4010 (X'0FAA')	MQRCCF_HOST_NOT_AVAILABLE
4011 (X'0FAB')	MQRCCF_CONFIGURATION_ERROR
4012 (X'0FAC')	MQRCCF_CONNECTION_REFUSED
4013 (X'0FAD')	MQRCCF_ENTRY_ERROR
4014 (X'0FAE')	MQRCCF_SEND_FAILED
4015 (X'0FAF')	MQRCCF_RECEIVED_DATA_ERROR
4016 (X'0FB0')	MQRCCF_RECEIVE_FAILED
4017 (X'0FB1')	MQRCCF_CONNECTION_CLOSED
4018 (X'0FB2')	MQRCCF_NO_STORAGE
4019 (X'0FB3')	MQRCCF_NO_COMMS_MANAGER
4020 (X'0FB4')	MQRCCF_LISTENER_NOT_STARTED
4024 (X'0FB8')	MQRCCF_BIND_FAILED
4025 (X'0FB9')	MQRCCF_CHANNEL_INDOUBT
4026 (X'0FBA')	MQRCCF_MQCONN_FAILED
4027 (X'0FBB')	MQRCCF_MQOPEN_FAILED
4028 (X'0FBC')	MQRCCF_MQGET_FAILED
4029 (X'0FBD')	MQRCCF_MQPUT_FAILED
4030 (X'0FBE')	MQRCCF_PING_ERROR
4031 (X'0FBF')	MQRCCF_CHANNEL_IN_USE
4032 (X'0FC0')	MQRCCF_CHANNEL_NOT_FOUND
4033 (X'0FC1')	MQRCCF_UNKNOWN_REMOTE_CHANNEL
4034 (X'0FC2')	MQRCCF_REMOTE_QM_UNAVAILABLE
4035 (X'0FC3')	MQRCCF_REMOTE_QM_TERMINATING
4036 (X'0FC4')	MQRCCF_MQINQ_FAILED
4037 (X'0FC5')	MQRCCF_NOT_XMIT_Q
4038 (X'0FC6')	MQRCCF_CHANNEL_DISABLED
4039 (X'0FC7')	MQRCCF_USER_EXIT_NOT_AVAILABLE
4040 (X'0FC8')	MQRCCF_COMMIT_FAILED
4041 (X'0FC9')	MQRCCF_WRONG_CHANNEL_TYPE
4042 (X'0FCA')	MQRCCF_CHANNEL_ALREADY_EXISTS
4043 (X'0FCB')	MQRCCF_DATA_TOO_LARGE
4044 (X'0FCC')	MQRCCF_CHANNEL_NAME_ERROR
4045 (X'0FCD')	MQRCCF_XMIT_Q_NAME_ERROR
4047 (X'0FCF')	MQRCCF_MCA_NAME_ERROR
4048 (X'0FD0')	MQRCCF_SEND_EXIT_NAME_ERROR
4049 (X'0FD1')	MQRCCF_SEC_EXIT_NAME_ERROR
4050 (X'0FD2')	MQRCCF_MSG_EXIT_NAME_ERROR
4051 (X'0FD3')	MQRCCF_RCV_EXIT_NAME_ERROR

4052 (X'0FD4')	MQRCCF_XMIT_Q_NAME_WRONG_TYPE
4053 (X'0FD5')	MQRCCF_MCA_NAME_WRONG_TYPE
4054 (X'0FD6')	MQRCCF_DISC_INT_WRONG_TYPE
4055 (X'0FD7')	MQRCCF_SHORT_RETRY_WRONG_TYPE
4056 (X'0FD8')	MQRCCF_SHORT_TIMER_WRONG_TYPE
4057 (X'0FD9')	MQRCCF_LONG_RETRY_WRONG_TYPE
4058 (X'0FDA')	MQRCCF_LONG_TIMER_WRONG_TYPE
4059 (X'0FDB')	MQRCCF_PUT_AUTH_WRONG_TYPE
4061 (X'0FDD')	MQRCCF_MISSING_CONN_NAME
4062 (X'0FDE')	MQRCCF_CONN_NAME_ERROR
4063 (X'0FDF')	MQRCCF_MQSET_FAILED
4064 (X'0FE0')	MQRCCF_CHANNEL_NOT_ACTIVE
4065 (X'0FE1')	MQRCCF_TERMINATED_BY_SEC_EXIT
4067 (X'0FE3')	MQRCCF_DYNAMIC_Q_SCOPE_ERROR
4068 (X'0FE4')	MQRCCF_CELL_DIR_NOT_AVAILABLE
4069 (X'0FE5')	MQRCCF_MR_COUNT_ERROR
4070 (X'0FE6')	MQRCCF_MR_COUNT_WRONG_TYPE
4071 (X'0FE7')	MQRCCF_MR_EXIT_NAME_ERROR
4072 (X'0FE8')	MQRCCF_MR_EXIT_NAME_WRONG_TYPE
4073 (X'0FE9')	MQRCCF_MR_INTERVAL_ERROR
4074 (X'0FEA')	MQRCCF_MR_INTERVAL_WRONG_TYPE
4075 (X'0FEB')	MQRCCF_NPM_SPEED_ERROR
4076 (X'0FEC')	MQRCCF_NPM_SPEED_WRONG_TYPE
4077 (X'0FED')	MQRCCF_HB_INTERVAL_ERROR
4078 (X'0FEE')	MQRCCF_HB_INTERVAL_WRONG_TYPE
4079 (X'0FEF')	MQRCCF_CHAD_ERROR
4080 (X'0FF0')	MQRCCF_CHAD_WRONG_TYPE
4081 (X'0FF1')	MQRCCF_CHAD_EVENT_ERROR
4082 (X'0FF2')	MQRCCF_CHAD_EVENT_WRONG_TYPE
4083 (X'0FF3')	MQRCCF_CHAD_EXIT_ERROR
4084 (X'0FF4')	MQRCCF_CHAD_EXIT_WRONG_TYPE
4085 (X'0FF5')	MQRCCF_SUPPRESSED_BY_EXIT
4086 (X'0FF6')	MQRCCF_BATCH_INT_ERROR
4087 (X'0FF7')	MQRCCF_BATCH_INT_WRONG_TYPE
4088 (X'0FF8')	MQRCCF_NET_PRIORITY_ERROR
4089 (X'0FF9')	MQRCCF_NET_PRIORITY_WRONG_TYPE
4090 (X'0FFA')	MQRCCF_CHANNEL_CLOSED
4092 (X'0FFC')	MQRCCF_SSL_CIPHER_SPEC_ERROR
4093 (X'0FFD')	MQRCCF_SSL_PEER_NAME_ERROR
4094 (X'0FFE')	MQRCCF_SSL_CLIENT_AUTH_ERROR

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'
MQRQ_SSL_HANDSHAKE_ERROR	13	X'0000000D'
MQRQ_SSL_CIPHER_SPEC_ERROR	14	X'0000000E'
MQRQ_SSL_CLIENT_AUTH_ERROR	15	X'0000000F'
MQRQ_SSL_PEER_NAME_ERROR	16	X'00000010'

MQSCA_* (SSL client authentication)

MQSCA_REQUIRED	0	X'00000000'
MQSCA_OPTIONAL	1	X'00000001'

MQSUS_* (Suspend status)

MQSUS_NO	0	X'00000000'
MQSUS_YES	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 PCF commands and responses These are described below for each of the supported programming languages. Not all of the files are available in all environments.

See:
- "C header files"
- "COBOL COPY files"
- "PL/I INCLUDE files" on page 378
- "System/390 Assembler COPY files" on page 378

C header files

The following header files are provided for the C programming language.

Table 14. 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

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 15. 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 z/OS)
CMQXV	–	Named constants for events and PCF commands relating to channels (available only on z/OS and OS/400)
CMQCFHV	CMQCFHL	Header structure for events and PCF commands (available only on z/OS)
CMQCFINV	CMQCFINL	Single-integer parameter structure for events and PCF commands (available only on z/OS)
CMQCFILV	CMQCFILL	Integer-list parameter structure for events and PCF commands (available only on z/OS)

Table 15. COBOL COPY files (continued)

File name (with initial values)	File name (without initial values)	Contents relating to this book
CMQCFSTV	CMQCFSTL	Single-string parameter structure for events and PCF commands (available only on z/OS)
CMQCFSLV	CMQCFSLL	String-list parameter structure for events and PCF commands (available only on z/OS)

PL/I INCLUDE files

The following INCLUDE files are provided for the PL/I programming language. These files are available only on z/OS, OS/2, and Windows.

Table 16. 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 z/OS.

Table 17. 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

RPG COPY files

The following COPY files are provided for the RPG programming language. These files are available only on iSeries.

Table 18. RPG COPY files

Filename	Contents relating to this book
CMQG	Some named constants for events and PCF commands
CMQCFG	Additional named constants for events and PCF commands

Table 18. RPG COPY files (continued)

Filename	Contents relating to this book
CMQXG	Named constants for events and PCF commands relating to channels
CMQCFHG	Header structure for events and PCF commands
CMQCFING	Single-integer parameter structure for events and PCF commands
CMQCFILG	Integer-list parameter structure for events and PCF commands
CMQCFSTG	Single-string parameter structure for events and PCF commands
CMQCFSLG	String-list parameter structure for events and PCF commands

Appendix D. MQAI Return codes

For each MQAI call, a completion code and a reason code are returned by the queue manager or by an exit routine, to indicate the success or failure of the call.

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.

Completion codes

The completion code parameter (*CompCode*) allows the caller to see quickly whether the call completed successfully, completed partially, or failed.

The following is a list of completion codes, with more detail than is given in the call descriptions:

MQCC_OK
Successful completion.

The call completed fully; all output parameters have been set. The *Reason* parameter always has the value MQRC_NONE in this case.

MQCC_WARNING
Warning (partial completion).

The call completed partially. Some output parameters may have been set in addition to the *CompCode* and *Reason* output parameters. The *Reason* parameter gives additional information about the partial completion.

MQCC_FAILED
Call failed.

The processing of the call did not complete, and the state of the queue manager is normally unchanged; exceptions are specifically noted. The *CompCode* and *Reason* output parameters have been set; other parameters are unchanged, except where noted.

The reason may be a fault in the application program, or it may be a result of some situation external to the program, for example the application's authority may have been revoked. The *Reason* parameter gives additional information about the error.

Reason codes

The reason code parameter (*Reason*) is a qualification to the completion code parameter (*CompCode*).

If there is no special reason to report, MQRC_NONE is returned. A successful call returns MQCC_OK and MQRC_NONE.

If the completion code is either MQCC_WARNING or MQCC_FAILED, the queue manager always reports a qualifying reason; details are given under each call description.

MQAI Return codes

For complete descriptions of reason codes see:

- *WebSphere MQ for z/OS Messages and Codes* for WebSphere MQ for z/OS
- *WebSphere MQ Messages* for all other WebSphere MQ platforms

Appendix E. MQAI Constants in C

This appendix specifies the values of the named constants that apply to MQAI calls. For MQI constants, refer to the *WebSphere MQ Intercommunication* book.

List of constants

The following sections list all of the named constants mentioned in part 2 of this book, and shows their values.

```
/* Create-bag options for mqCreateBag */

#define MQCBO_NONE                          (0x00000000)
#define MQCBO_USER_BAG                      (0x00000000)
#define MQCBO_ADMIN_BAG                     (0x00000001)
#define MQCBO_COMMAND_BAG                   (0x00000010)
#define MQCBO_SYSTEM_BAG                    (0x00000020)
#define MQCBO_LIST_FORM_ALLOWED             (0x00000002)
#define MQCBO_LIST_FORM_INHIBITED           (0x00000000)
#define MQCBO_REORDER_AS_REQUIRED           (0x00000004)
#define MQCBO_DO_NOT_REORDER                (0x00000000)
#define MQCBO_CHECK_SELECTORS               (0x00000008)
#define MQCBO_DO_NOT_CHECK_SELECTORS        (0x00000000)
```

```
/* Special selector values */

#define MQSEL_ANY_SELECTOR                  (-30001)
#define MQSEL_ANY_USER_SELECTOR             (-30002)
#define MQSEL_ANY_SYSTEM_SELECTOR           (-30003)
#define MQSEL_ALL_SELECTORS                 (-30001)
#define MQSEL_ALL_USER_SELECTORS            (-30002)
#define MQSEL_ALL_SYSTEM_SELECTORS          (-30003)
```

```
/* Integer user selectors */

#define MQIACF_ALL                          1009
#define MQIACF_INQUIRY                      1074
#define MQIACF_WAIT_INTERVAL                1075
```

```
/* Handle user selectors */

#define MQHA_BAG_HANDLE                     4001
```

```
/* Limits for handle user selectors */

#define MQHA_FIRST                          4001
#define MQHA_LAST_USED                      4001
#define MQHA_LAST                           6000
```

```
/* Limits for selectors for object attributes */

#define MQOA_FIRST                                   1
#define MQOA_LAST                                 6000


/* Integer system selectors */

#define MQIASY_FIRST                              (-1)
#define MQIASY_CODED_CHAR_SET_ID                  (-1)
#define MQIASY_TYPE                               (-2)
#define MQIASY_COMMAND                            (-3)
#define MQIASY_MSG_SEQ_NUMBER                     (-4)
#define MQIASY_CONTROL                            (-5)
#define MQIASY_COMP_CODE                          (-6)
#define MQIASY_REASON                             (-7)
#define MQIASY_BAG_OPTIONS                        (-8)
#define MQIASY_LAST_USED                          (-8)
#define MQIASY_LAST                            (-2000)


/* Limits for integer system selectors */

#define MQIASY_FIRST                              (-1)
#define MQIASY_LAST_USED                          (-7)
#define MQIASY_LAST                            (-2000)


/* Special index values */

#define MQIND_NONE                                (-1)
#define MQIND_ALL                                 (-2)


/* Bag handles */

#define MQHB_UNUSABLE_HBAG                        (-1)
#define MQHB_NONE                                 (-2)


/* Queue handles */

#define MQHO_NONE                                 (-2)


/* Values for "BufferLength" parameter on mqAddString/mqSetString */

#define MQBL_NULL_TERMINATED                      (-1)


/* Values for "ItemType" parameter on mqInquireItemInfo */

#define MQIT_INTEGER                                 1
#define MQIT_STRING                                  2
#define MQIT_BAG                                     3
```

```
/* Coded character set identifiers */

#define MQCCSI_DEFAULT                                    0


/* Character-attribute selectors */

#define MQCA_Q_NAME                                    2016


/* Integer-attribute selectors */

#define MQIA_Q_TYPE                                      20
#define MQIA_SCOPE                                       45


/* Queue types */

#define MQQT_LOCAL                                        1


/* Queue definition scope */

#define MQSCO_Q_MGR                                       1


/* Control options */

#define MQCFC_LAST                                        1


/* Formats */

#define MQFMT_EVENT                              “MQEVENT ”
#define MQFMT_PCF                                   “MQPCF ”
#define MQFMT_ADMIN                              “MQADMIN ”


/* Reason codes */

#define MQRC_STORAGE_NOT_AVAILABLE                     2071
#define MQRC_COMMAND_TYPE_ERROR                        2300
#define MQRC_BUFFER_ERROR                              2004
#define MQRC_BUFFER_LENGTH_ERROR                       2005
#define MQRC_DATA_LENGTH_ERROR                         2010
#define MQRC_OPTIONS_ERROR                             2046
#define MQRC_MULTIPLE_INSTANCE_ERROR                   2301
#define MQRC_SYSTEM_ITEM_NOT_ALTERABLE                 2302
#define MQRC_BAG_CONVERSION_ERROR                      2303
#define MQRC_SELECTOR_OUT_OF_RANGE                     2304
#define MQRC_SELECTOR_NOT_UNIQUE                       2305
#define MQRC_INDEX_NOT_PRESENT                         2306
#define MQRC_STRING_ERROR                              2307
#define MQRC_ENCODING_NOT_SUPPORTED                    2308
#define MQRC_SELECTOR_NOT_PRESENT                      2309
#define MQRC_OUT_SELECTOR_ERROR                        2310
#define MQRC_DATA_TRUNCATED                            2311
#define MQRC_STRING_TRUNCATED                          2311
#define MQRC_SELECTOR_WRONG_TYPE                       2312
```

#define MQRC_INCONSISTENT_ITEM_TYPE	2313
#define MQRC_INDEX_ERROR	2314
#define MQRC_SYSTEM_BAG_NOT_ALTERABLE	2315
#define MQRC_ITEM_COUNT_ERROR	2316
#define MQRC_FORMAT_NOT_SUPPORTED	2317
#define MQRC_SELECTOR_NOT_SUPPORTED	2318
#define MQRC_ITEM_VALUE_ERROR	2319
#define MQRC_HBAG_ERROR	2320
#define MQRC_PARAMETER_MISSING	2321
#define MQRC_CMD_SERVER_NOT_AVAILABLE	2322
#define MQRC_STRING_LENGTH_ERROR	2323
#define MQRC_INQUIRY_COMMAND_ERROR	2324
#define MQRC_NESTED_BAG_NOT_SUPPORTED	2325
#define MQRC_BAG_WRONG_TYPE	2326
#define MQRC_ITEM_TYPE_ERROR	2327
#define MQRC_SYSTEM_BAG_NOT_DELETABLE	2328
#define MQRC_SYSTEM_ITEM_NOT_DELETABLE	2329
#define MQRC_CODED_CHAR_SET_ID_ERROR	2330
#define MQRCCF_COMMAND_FAILED	3008

```
/* Function names */
```

#define mqAddInquiry	MQADDIQ
#define mqAddInteger	MQADDIN
#define mqAddString	MQADDST
#define mqBagToBuffer	MQBG2BF
#define mqBufferToBag	MQBF2BG
#define mqClearBag	MQCLRBG
#define mqCountItems	MQCNTIT
#define mqCreateBag	MQCRTBG
#define mqDeleteBag	MQDELBG
#define mqDeleteItem	MQDELIT
#define mqExecute	MQEXEC
#define mqGetBag	MQGETBG
#define mqInquireBag	MQINQBG
#define mqInquireInteger	MQINQIN
#define mqInquireItemInfo	MQINQII
#define mqInquireString	MQINQST
#define mqPad	MQPAD
#define mqPutBag	MQPUTBG
#define mqSetInteger	MQSETIN
#define mqSetString	MQSETST
#define mqTrim	MQTRIM
#define mqTruncateBag	MQTRNBG

Elementary datatypes in C

```
typedef MQLONG MQHBAG;
typedef MQHBAG MQPOINTER PMQHBAG;
```

Appendix F. MQAI Header files

WebSphere MQ provides C and Visual Basic header files to help you write your MQAI applications:

Table 19. Header files

C	Visual Basic	Description
cmqbc.h	CMQBB.BAS	Contains prototypes, datatypes (MQHBAG), and named constants for the MQAI.
cmqcfc.h	CMQCFB.BAS	Contains elementary datatypes and named constants for events and PCF commands.
cmqc.h	CMQB.BAS	Contains prototypes, data types, and named constants for the main MQI.

Appendix G. MQAI Selectors

Items in bags are identified by a *selector* that acts as an identifier for the item. There are two types of selector, *user selector* and *system selector*.

User selectors

User selectors have values that are zero or positive. For the administration of MQSeries objects, valid user selectors are already defined by the following constants:
- MQCA_* and MQIA_* (object attributes)
- MQCACF_* and MQIACF_* (items relating specifically to PCF)
- MQCACH_* and MQIACH_* (channel attributes)

For user messages, the meaning of a user selector is defined by the application.

The following additional user selectors are introduced by the MQAI:

MQIACF_INQUIRY
Identifies an WebSphere MQ object attribute to be returned by an Inquire command.

MQHA_BAG_HANDLE
Identifies a bag handle residing within another bag.

MQHA_FIRST
Lower limit for handle selectors.

MQHA_LAST
Upper limit for handle selectors.

MQHA_LAST_USED
Upper limit for last handle selector allocated.

MQCA_USER_LIST
Default user selector. Supported on Visual Basic only. This selector supports character type and represents the default value used if the `Selector` parameter is omitted on the mqAdd*, mqSet*, or mqInquire* calls.

MQIA_USER_LIST
Default user selector. Supported on Visual Basic only. This selector supports integer type and represents the default value used if the `Selector` parameter is omitted on the mqAdd*, mqSet*, or mqInquire* calls.

System selectors

System selectors have negative values. The following system selectors are included in the bag when it is created:

MQIASY_BAG_OPTIONS
Bag-creation options. A summation of the options used to create the bag. This selector cannot be changed by the user.

MQIASY_CODED_CHAR_SET_ID
Character-set identifier for the character data items in the bag. The initial value is the queue-manager’s character set.

The value in the bag is used on entry to the mqExecute call and set on exit from the mqExecute call. This also applies when character strings are added to or modified in the bag.

MQIASY_COMMAND
PCF command identifier. Valid values are the MQCMD_* constants. For user messages, the value MQCMD_NONE should be used. The initial value is MQCMD_NONE.

The value in the bag is used on entry to the mqPutBag and mqBagToBuffer calls, and set on exit from the mqExecute, mqGetBag and mqBufferToBag calls.

MQIASY_COMP_CODE
Completion code. Valid values are the MQCC_* constants. The initial value is MQCC_OK.

The value in the bag is used on entry to the mqExecute, mqPutBag, and mqBagToBuffer calls, and set on exit from the mqExecute, mqGetBag, and mqBufferToBag calls.

MQIASY_CONTROL
PCF control options. Valid values are the MQCFC_* constants. The initial value is MQCFC_LAST.

The value in the bag is used on entry to the mqExecute, mqPutBag, and mqBagToBuffer calls, and set on exit from the mqExecute, mqGetBag, and mqBufferToBag calls.

MQIASY_MSG_SEQ_NUMBER
PCF message sequence number. Valid values are 1 or greater. The initial value is 1.

The value in the bag is used on entry to the mqExecute, mqPutBag, and mqBagToBuffer calls, and set on exit from the mqExecute, mqGetBag, and mqBufferToBag calls.

MQIASY_REASON
Reason code. Valid values are the MQRC_* constants. The initial value is MQRC_NONE.

The value in the bag is used on entry to the mqExecute, mqPutBag, and mqBagToBuffer calls, and set on exit from the mqExecute, mqGetBag, and mqBufferToBag calls.

MQIASY_TYPE
PCF command type. Valid values are the MQCFT_* constants. For user messages, the value MQCFT_USER should be used. The initial value is MQCFT_USER for bags created as user bags and MQCFT_COMMAND for bags created as administration or command bags.

The value in the bag is used on entry to the mqExecute, mqPutBag, and mqBagToBuffer calls, and set on exit from the mqExecute, mqGetBag, and mqBufferToBag calls.

Appendix H. Notices

This information was developed for products and services offered in the United States. IBM may not offer the products, services, or features discussed in this information in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this information. The furnishing of this information does not give you any license to these patents. You can send license inquiries, in writing, to:
IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:
IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Notices

Trademarks

Index

Sending your comments to IBM

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book.

Please limit your comments to the information in this book and the way in which the information is presented.

To make comments about the functions of IBM products or systems, talk to your IBM representative or to your IBM authorized remarketer.

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

You can send your comments to IBM in any of the following ways:

- By mail, to this address:

  User Technologies Department (MP095)
  IBM United Kingdom Laboratories
  Hursley Park
  WINCHESTER,
  Hampshire
  SO21 2JN
  United Kingdom
- By fax:
  - From outside the U.K., after your international access code use 44–1962–816151
  - From within the U.K., use 01962–816151
- Electronically, use the appropriate network ID:
  - IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
  - IBMLink™: HURSLEY(IDRCF)
  - Internet: idrcf@hursley.ibm.com

Whichever method you use, ensure that you include:

- The publication title and order number
- The topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.

Printed in U.S.A.

SC34-6060-03