WebSphere MQ

Event Monitoring

SC34-6069-02

Note!
Before using this information and the product it supports, be sure to read the general information under "Notices" on page 185.

Third edition (December 2002)

This edition applies to the following WebSphere MQ V5.3 products:

- WebSphere MQ for AIX
- WebSphere MQ for HP-UX
- WebSphere MQ for iSeries
- WebSphere MQ for Linux for Intel
- WebSphere MQ for Linux for zSeries
- WebSphere MQ for Solaris
- WebSphere MQ for Windows
- WebSphere MQ for z/OS

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

Unless otherwise stated, the information also applies to these products:

- MQSeries for AT&T GIS (NCR) UNIX V2.2.1
- MQSeries for Compaq NonStop Kernel V5.1
- MQSeries for Compaq OpenVMS Alpha V5.1
- MQSeries for Compaq Tru64 UNIX, V5.1
- MQSeries for OS/2 Warp V5.1
- MQSeries for SINIX and DC/OSx V2.2.1

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

Contents

Figures

Tables

About this book

This book describes the facilities available on WebSphere® MQ products for monitoring instrumentation events in a network of connected systems that use IBM® WebSphere MQ products in different operating system environments.

Who this book is for

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

What you need to know to understand this book

You should have:

- 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.

Conventions used in this book

- z/OS™ means any release of z/OS or OS/390® that supports the current version of WebSphere MQ.
- Throughout this book, the term *object* refers to any WebSphere MQ queue manager, queue, namelist, channel, storage class, process, authentication information or CF structure.
- Throughout this book, the term *Windows*® refers to Windows NT® and Windows 2000.
- Throughout this book, there may be sections that do not specify which compliers and programming languages are supported on certain platforms. For information about which compilers and programming languages are supported on each platform see the *WebSphere MQ Application Programming Reference* manual.

How this book is organized

- Chapter 1, "An introduction to instrumentation events", on page 1 gives a general overview of instrumentation events.
- Chapter 2, "Understanding performance events", on page 15 goes into greater detail about performance events, specifically enabling and disabling them.
- Chapter 3, "Understanding configuration events (z/OS only)", on page 33 goes into greater detail about configuration events.
- Chapter 4, "Event message reference", on page 37 provides detailed reference information for specific events.
- Chapter 5, "Example of using instrumentation events", on page 133 contains an example of using events.

About this book

Summary of changes

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

Changes for this edition (SC34-6069-02)

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

Changes for the previous editions (SC34-6069-00 and -01)

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

- Changes have been made throughout the book to reflect the rebranding of MQSeries to WebSphere MQ.
- The configuration event has been introduced. The configuration event is a new type of instrumentation event and there are four types of configuration event:
  - Create object
  - Change object
  - Delete object
  - Refresh object
- The following constant types have been introduced:
  - MQBACF_*
  - MQEVO_*
  - MQOT_*
  - MQQSGD_*
- WebSphere MQ is now fully integrated with the Secure Sockets Layer (SSL) protocol. There is a new event that the SSL protocol can generate, it is called the Channel SSL Error event. For details of the SSL implementation on WebSphere MQ, see the *WebSphere MQ Security* book.
- The structure datatypes MQCFBS and MQCFSL have been introduced.
- RPG/ILE declaration examples have been introduced in the following structure datatypes:
  - MQCFIN
  - MQCFSL
  - MQCFST

Changes

Chapter 1. An introduction to instrumentation events

This chapter discusses:

What instrumentation events are

In WebSphere MQ, an instrumentation event is a logical combination of conditions that is detected by a queue manager or channel instance. Such an event causes the queue manager or channel instance to put a special message, called an *event message*, on an event queue.

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

Figure 1 on page 2 illustrates the concept of instrumentation events.

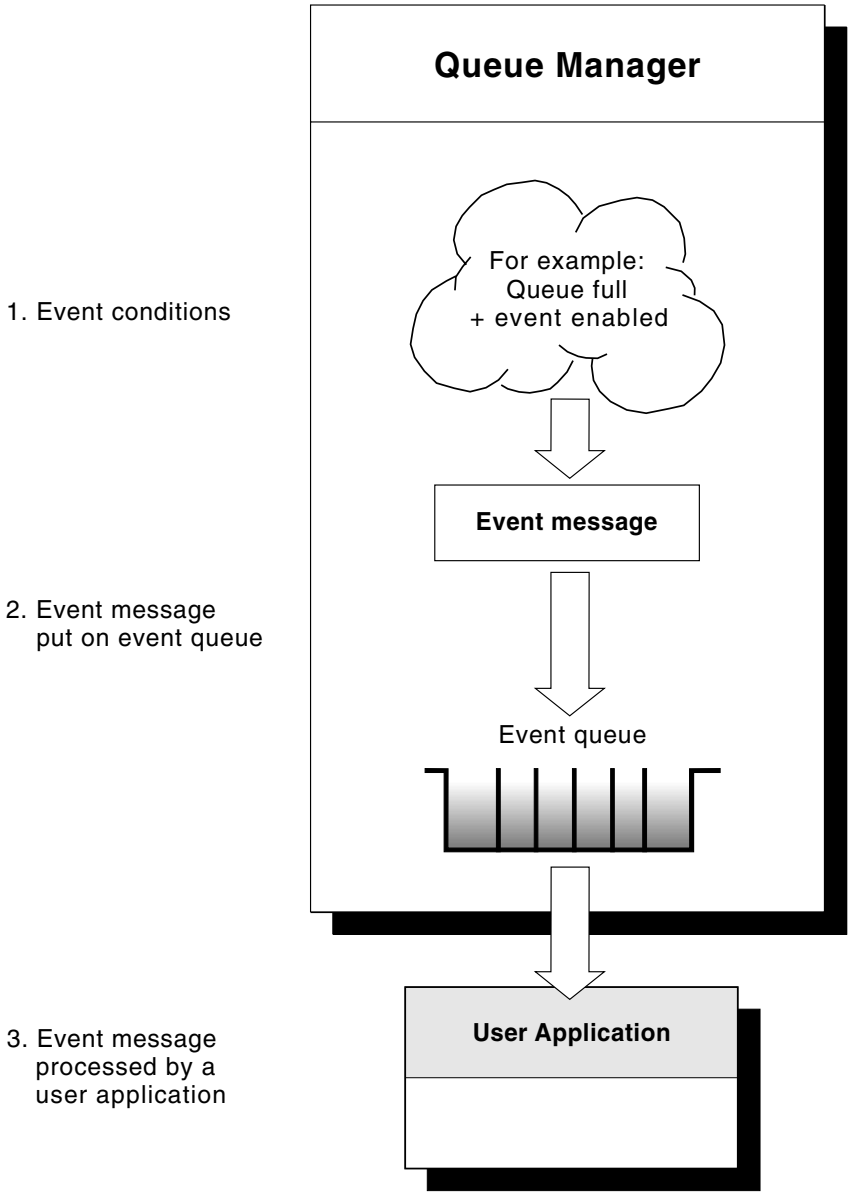

Figure 1. Understanding instrumentation events

Event notification through event queues

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

- Gets the message from the queue.
- Processes the message to extract the event data. For an overview of event message formats, see "Format of event messages" on page 12. For detailed descriptions of the format of each event message, see "Event message format" on page 38.

Types of event

WebSphere MQ instrumentation events come in the following types:

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

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

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

Configuration events
These events are notifications about the attributes of an object. They are generated automatically when the object is created, changed, or deleted, and are also generated by explicit requests. For example, when a namelist is created.

Note: Configuration events are available only with WebSphere MQ for z/OS.

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

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

By incorporating instrumentation events into your own system management application, you can monitor the activities across many queue managers, across many different nodes, for multiple WebSphere MQ applications. In particular, you can monitor all the nodes in your system from a single node (for those nodes that support WebSphere MQ events) as shown in Figure 2 on page 4.

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

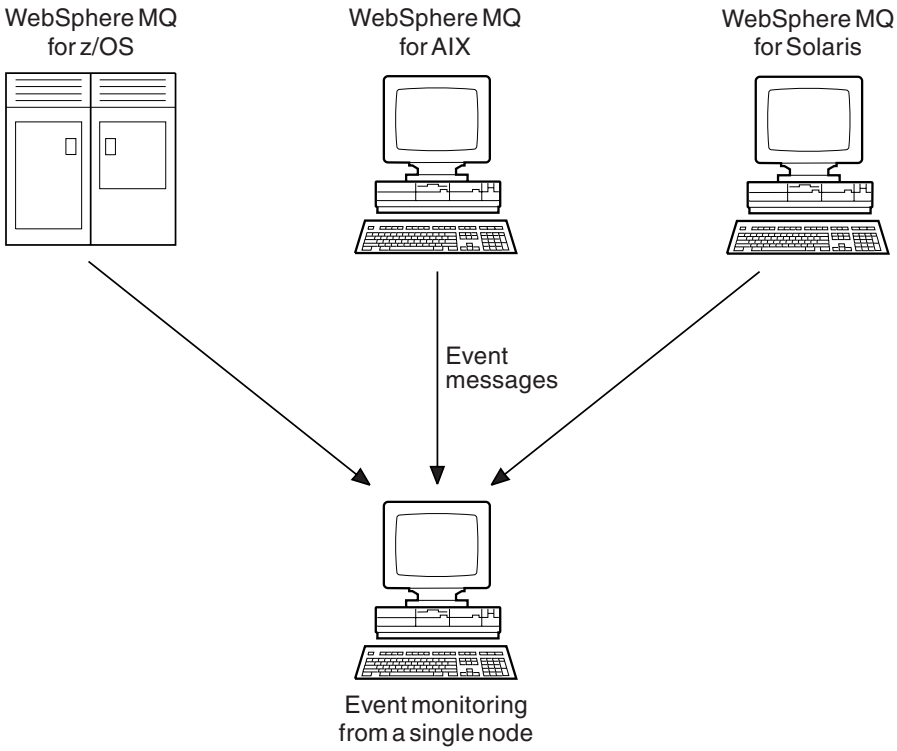

Figure 2. Monitoring queue managers across different platforms, on a single node

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

Queue manager events

Queue manager events are related to the use of resources within queue managers, such as an application trying to put a message to a queue that does not exist. The event messages for queue manager events are put on the SYSTEM.ADMIN.QMGR.EVENT queue. The following queue manager event types are supported:

- Authority (on Windows, Compaq OpenVMS Alpha, Compaq NonStop Kernel, and UNIX® systems only)
- Inhibit
- Local
- Remote
- Start and stop (z/OS supports only start)

For each event type in this list, there is a queue manager attribute that enables or disables the event type. See the *WebSphere MQ Script (MQSC) Command Reference* for more information.

The conditions that give rise to the event include:

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

  A similar condition can occur during the internal operation of a queue manager, for example, when generating a report message. The reason code in an event message might match an MQI reason code, even though it is not associated with any application. Do not assume that, because an event message reason code

looks like an MQI reason code, the event was necessarily caused by an unsuccessful MQI call from an application.

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

Authority events

Note

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

Authority events report an authorization, such as an application trying to open a queue for which it does not have the required authority, or a command being issued from a user ID that does not have the required authority.

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

"Not Authorized (type 1)" on page 91
"Not Authorized (type 2)" on page 92
"Not Authorized (type 3)" on page 94
"Not Authorized (type 4)" on page 96

Inhibit events

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

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

"Get Inhibited" on page 90
"Put Inhibited" on page 97

Local events

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

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

"Alias Base Queue Type Error" on page 50
"Unknown Alias Base Queue" on page 123
"Unknown Object Name" on page 127

Remote events

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

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

"Default Transmission Queue Type Error" on page 82
"Default Transmission Queue Usage Error" on page 84
"Queue Type Error" on page 111

Start and stop events

Start and stop events (z/OS supports only start) indicate that a queue manager has been started or has been requested to stop or quiesce.

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

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

Channel events

Channel events are reported by channels as a result of conditions detected during their operation, such as when a channel instance is stopped. Channel events are generated:

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

Notes:

1. No channel events are generated when using WebSphere MQ for z/OS with distributed queuing provided by CICS®.
2. Client connections on MQSeries for OS/390 Version 2, MQSeries Version 5, or later products, do not cause Channel Started or Channel Stopped events.

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

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

If a channel event is put onto an event queue, an error condition causes the queue manager to create an event as usual.

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

"Channel Conversion Error" on page 63
"Channel Not Activated" on page 66
"Channel Started" on page 71
"Channel Stopped" on page 73
"Channel Stopped By User" on page 76

IMS™ bridge events (z/OS only)

These events are reported when an IMS bridge starts or stops.

For more information about the event data returned in the messages specific to IMS bridge events, see

"Bridge Started (z/OS only)" on page 52
"Bridge Stopped (z/OS only)" on page 53

SSL events

The only SSL event is the Channel SSL Error event. This event is reported when a channel using the Secure Sockets Layer (SSL) fails to establish an SSL connection.

For more information about the event data returned in the message specific to the SSL event, see "Channel SSL Error" on page 68

Performance events

These events report that a resource has reached a threshold condition. For example, a queue depth limit might have been reached. For further details on performance events, see Chapter 2, "Understanding performance events", on page 15.

Performance events relate to conditions that can affect the performance of applications that use a specified queue. They are not generated for the event queues themselves.

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

If a queue manager tries to put a queue manager event or performance event message on an event queue and an error that would normally create an event is detected, another event is not created and no action is taken.

MQGET and MQPUT calls within a unit of work can generate performance events regardless of whether the unit of work is committed or backed out.

There are two types of performance event:

Queue depth events

Queue depth events relate to the number of messages on a queue; that is how full, or empty, the queue is. These event is supported for shared queues.

Queue service interval events

Queue service interval events relate to whether messages are processed within a user-specified time interval. These events are not supported for shared queues.

WebSphere MQ for z/OS supports queue depth events for QSGDISP (SHARED) queues, but not service interval events. Queue manager and channel events remain unaffected by shared queues.

Configuration events (z/OS only)

Configuration events report the attributes of an object that has been created or modified in some way. For example, a configuration event message is generated if

a namelist object is created. For more information see Chapter 3, "Understanding configuration events (z/OS only)", on page 33.

There are four types of configuration event:

Create object events
These events are generated when an object is created. For more information see "Create object (z/OS only)" on page 78.

Change object events
These events are generated when an object is changed. For more information see "Change object (z/OS only)" on page 55.

Delete object events
These events are generated when an object is deleted. For more information see "Delete object (z/OS only)" on page 86.

Refresh object events
These events are generated by an explicit request to refresh. For more information see "Refresh object (z/OS only)" on page 113.

Event message data summary

Table 1 is a full list of events. Use it to find information about a particular type of event message:

Table 1. Event message data summary

Event type	*Event name*	*page*
Authority events	Not authorized (type 1)	91
	Not authorized (type 2)	92
	Not authorized (type 3)	94
	Not authorized (type 4)	96
Channel events	Channel activated	59
	Channel auto-Definition Error	60
	Channel auto-Definition OK	62
	Channel conversion Error	63
	Channel not Activated	66
	Channel started	71
	Channel stopped	73
	Channel stopped by user	76
Configuration events	Create object	78
	Change object	55
	Delete object	86
	Refresh object	113
IMS Bridge events	Bridge started	52
	Bridge stopped	53
Inhibit events	Get inhibited	90
	Put inhibited	97
Local events	Alias base queue type error	50
	Unknown alias base queue	123
	Unknown object name	127
Performance events	Queue depth high	99
	Queue depth low	101
	Queue full	103
	Queue service interval high	107
	Queue service interval OK	109

Table 1. Event message data summary (continued)

Event type	*Event name*	*page*
Remote events	Default transmission queue type error	82
	Default transmission queue usage error	84
	Queue type error	111
	Remote queue name error	117
	Transmission queue type error	119
	Transmission queue usage error	121
	Unknown default transmission queue	125
	Unknown remote queue manager	129
	Unknown transmission queue	131
SSL events	Channel SSL error	68
Start and stop events	Queue manager active	105
	Queue manager not active	106

Enabling and disabling events

With the exception of channel events, all instrumentation events must be enabled before they can be generated. For example, the conditions giving rise to a *Queue Full* event are:

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

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

- WebSphere MQ script commands (MQSC). For more information, see the *WebSphere MQ Script (MQSC) Command Reference* manual.
- The corresponding WebSphere MQ PCF commands. For more information see the *WebSphere MQ Programmable Command Formats and Administration Interface*.
- The operations and control panels for queue managers on z/OS. For more information, see the *WebSphere MQ for z/OS System Administration Guide*.

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

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

Enabling and disabling queue manager events

Enable queue manager events by specifying the appropriate attribute on the MQSC command ALTER QMGR. For example, to enable inhibit events on the default queue manager, use this MQSC command:

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

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

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

Authority events

You enable authority events using:

- The AUTHOREV attribute on the MQSC command ALTER QMGR

Inhibit events
You enable inhibit events using:
- The INHIBTEV attribute on the MQSC command ALTER QMGR

Local events
You enable local events using:
- The LOCALEV attribute on the MQSC command ALTER QMGR

Remote events
You enable remote events using:
- The REMOTEEV attribute on the MQSC command ALTER QMGR

Start and stop events
You enable start and stop events using:
- The STRSTPEV attribute on the MQSC command ALTER QMGR

Enabling queue manager events summary
Table 2 summarizes how to enable queue manager events:

Table 2. Enabling queue manager events using MQSC commands

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

Enabling channel events
Most channel events are enabled automatically and you cannot enable or disable them by command. The exceptions are the two automatic channel definition events. However, you can suppress channel events by not defining the channel events queue, or by making it put-inhibited. Note that this could cause a queue to fill up if remote event queues point to a put-inhibited channel events queue.

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

Channel auto-definition
The generation of these events is controlled by the `ChannelAutoDefEvent` queue-manager attribute. Refer to the *WebSphere MQ Application Programming Reference* manual for further details of this attribute.

Enabling performance events
Performance events as a whole must be enabled on the queue manager, or no performance events can occur. You can then enable specific performance events by setting the appropriate queue attribute. You also have to specify the conditions that give rise to the event.

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

1. Enable performance events on the queue manager.
2. Enable the event on the required queue.

3. Set the limits, if required, to the appropriate levels, expressed as a percentage of the maximum queue depth. For more information, see "Enabling queue service interval events" on page 18.

Enabling queue service interval events

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

1. Enable performance events on the queue manager.
2. Set the control attribute for a Queue Service Interval High or OK event on the queue as required.
3. Specify the service interval time by setting the *QServiceInterval* attribute for the queue to the appropriate length of time. For more information, see "Understanding queue depth events" on page 25.

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

Enabling configuration events

Configuration events are only available on z/OS.

Enable configuration events specifying the CONFIGEV attribute on the MQSC command ALTER QMGR:

```
ALTER QMGR CONFIGEV (ENABLED)
```

To disable the events, set the CONFIGEV attribute to DISABLED:

```
ALTER QMGR CONFIGEV (DISABLED)
```

Conditions that cause events

Conditions that can give rise to instrumentation events include:

- A threshold limit for the number of messages on a queue is reached.
- A channel instance is started or stopped.
- A queue manager becomes active, or is requested to stop.
- An application tries to open a queue specifying a user ID that is not authorized on WebSphere MQ for iSeries™, Linux, Windows, and UNIX systems, and on MQSeries for Digital OpenVMS, Compaq NonStop Kernel, and Compaq OpenVMS Alpha.
- Objects are created, deleted, changed or refreshed (z/OS only).

Note: Putting a message on the dead-letter queue can cause an event to be generated if the event conditions are met.

Event queues

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

You must not define event queues as transmission queues, because event messages have formats that are incompatible with the format of messages required for transmission queues.

Event queues

Shared event queues are local queues defined with the QSGDISP(SHARED) value. For more information about defining shared queues, see the *WebSphere MQ for z/OS System Setup Guide*.

When an event queue is unavailable

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

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

An event queue might be unavailable for many different reasons including:
- The queue has not been defined.
- The queue has been deleted.
- The queue is full.
- The queue has been put-inhibited.

The absence of an event queue does not prevent the event from occurring. For example, after a performance event, the queue manager changes the queue attributes and resets the queue statistics. This happens whether the event message is put on the performance event queue or not. For more information about performance events changing queue attributes, see Chapter 2, "Understanding performance events", on page 15.

In the case of configuration events the same is true again. In the absence of an event queue, a configuration event message will **not** be generated, however the command or call will go ahead and the object will be created or manipulated in the desired way.

Using triggered event queues

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

Event queues can have trigger actions associated with them and can create trigger messages. However, if these trigger messages in turn cause conditions that would normally generate an event, no event is generated. This ensures that looping does not occur.

Format of event messages

Event messages contain information about the event and its origin. Typically, these messages are processed by a system management application program tailored to meet the requirements of the enterprise at which it runs. As with all WebSphere MQ messages, an event message has two parts: a message descriptor and the message data.

- The message descriptor is based on the MQMD structure, which is defined in the *WebSphere MQ Application Programming Reference* manual.
- The message data is also made up of an *event header* and the *event data*. The event header contains the reason code that identifies the event type. Putting the event message, and any subsequent actions following that, do not affect the reason code returned by the MQI call that caused the event. The event data provides further information about the event.

When the conditions are met to generate an event message to be generated for a shared queue, the queue managers in the queue sharing group decide whether to generate an event message. Several queue managers can generate an event message for one shared queue, resulting in several event messages being produced. To ensure that a system can correlate multiple event messages from different queue managers, these event messages have a unique correlation identifier *(CorrelId)* set in the message descriptor (MQMD). For further details of the MQMD see "Message descriptor (MQMD) in event messages" on page 39.

Using event monitoring in an WebSphere MQ network

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

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

See the *WebSphere MQ Application Programming Guide* for more information.

Monitoring performance on Windows

On Windows, performance data is stored using performance counters that can be accessed using the system registry. Within the registry, the counters are grouped according to the type of object to which they apply. For WebSphere MQ the type of object is `WebSphere MQ queues`.

For each queue the following performance counters are available:

- The current queue depth
- The queue depth as a percentage of the maximum queue depth
- The number of messages per second being placed on the queue
- The number of messages per second being removed from the queue

For messages sent to a distribution list, the performance monitor counts the number of messages put on to each queue.

In the case of large messages, the performance monitor counts the appropriate number of small messages. See the *WebSphere MQ System Administration Guide* guide for information on using the Windows performance monitor to view performance information. For details of how to access the performance counters in your own application, see the Microsoft® Web site at:

`http://msdn.microsoft.com/developer/`

Follow the links from this site to obtain online platform SDK information.

Chapter 2. Understanding performance events

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

In this chapter, the examples assume that you set queue attributes by using the appropriate WebSphere MQ commands (MQSC). See the *WebSphere MQ Script (MQSC) Command Reference* manual for more information. You can also set them using the operations and controls panels, for queue managers, on z/OS.

What performance events are

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

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

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

Every performance event message that is generated is placed on the queue, SYSTEM.ADMIN.PERFM.EVENT.

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

"Queue Depth High" on page 99
"Queue Depth Low" on page 101
"Queue Full" on page 103
"Queue Service Interval High" on page 107
"Queue Service Interval OK" on page 109

Performance event statistics

The event data in the event message contains information about the event for system management programs. For all performance events, the event data contains the names of the queue manager and the queue associated with the event. Also, the event data contains statistics related to the event. You can use these statistics to analyze the behavior of a specified queue. Table 3 summarizes the event statistics. All the statistics refer to what has happened since the last time the statistics were reset.

Table 3. Performance event statistics

Parameter	Description
TimeSinceReset	The elapsed time since the statistics were last reset.

Table 3. Performance event statistics (continued)

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

Performance event statistics are reset when any of the following occur:

- A performance event occurs (statistics are reset on all active queue managers).
- A queue manager stops and restarts.
- On z/OS only, the RESET QSTATS command is issued at the console.
- The PCF command, Reset Queue Statistics, is issued from a user-written administration program.

Understanding queue service interval events

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

Note: Queue service interval events are **not** supported on shared queues.

What queue service interval events are

The following are types of queue service interval events:

1. **Queue Service Interval OK** event indicates that after one of the following:
   - An MQPUT call
   - An MQGET call that leaves a non-empty queue

   an MQGET call was performed within a user-defined time period, known as the *service interval*.

   The Queue Service Interval OK event message can only be caused by an MQGET call.

   Note: In this section, Queue Service Interval OK events are referred to as OK events.
2. **Queue Service Interval High** event indicates that after one of the following:
   - An MQPUT call
   - An MQGET call that leaves a non-empty queue

   an MQGET call was **not** performed within a user-defined service interval.

   The Queue Service Interval High event message can be caused by an MQGET or a MQPUT call.

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

To enable both Queue Service Interval OK and Queue Service Interval High events you need to set the `QServiceIntervalEvent` control attribute to High. Queue Service Interval OK events are automatically enabled when a Queue Service Interval High event is generated. You do not need to enable Queue Service Interval OK events independently.

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

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

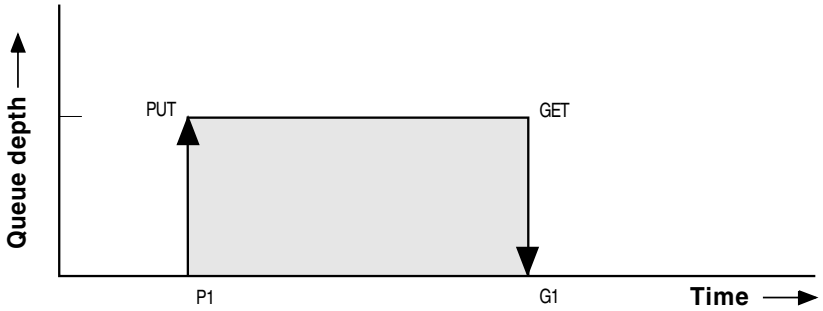

Figure 3. Understanding queue service interval events

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

- If the elapsed time between the put and get is less than or equal to the service interval:
  - A *Queue Service Interval OK* event is generated at G1, if queue service interval events are enabled
- If the elapsed time between the put and get is greater than the service interval:
  - A *Queue Service Interval High* event is generated at G1, if queue service interval events are enabled.

The actual algorithm for starting the service timer and generating events is described in "Queue service interval events algorithm" on page 18.

Understanding the service timer

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

What precisely does the service timer measure?

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

When is the service timer active?

The service timer is always active (running), if the queue has messages on it (depth is nonzero) and a queue service interval event is enabled. If the queue becomes empty (queue depth zero), the timer is put into an OFF state, to be restarted on the next put.

Queue service interval events

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

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

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

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

What about the *TimeSinceReset* parameter?
The *TimeSinceReset* parameter is returned as part of the event statistics in the event data. It specifies the time between successive queue service interval events, unless the event statistics are reset.

Queue service interval events algorithm

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

Service timer

The service timer is reset to zero and restarted:

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

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

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

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

Queue Service Interval High events

The Queue Service Interval event must be enabled (set to `HIGH`).

If the service time is greater than the service interval, an event is generated on the next MQPUT or MQGET call.

Queue Service Interval OK events

Queue Service Interval OK events are automatically enabled when a Queue Service Interval High event is generated.

If the service time (elapsed time) is less than or equal to the service interval, an event is generated on the next MQGET call.

Enabling queue service interval events

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

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

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

```
ALTER QMGR +
PERFMEV(ENABLED)

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

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

Automatic enabling of queue service interval events

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

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

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

Notes™:

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

Table 4. Enabling queue service interval events using MQSC

Queue service interval event	*Queue attributes*
Queue Service Interval High Queue Service Interval OK No queue service interval events	QSVCIEV (HIGH) QSVCIEV (OK) QSVCIEV (NONE)
Service interval	QSVCINT (*tt*) where *tt* is the service interval time in milliseconds.

Queue service interval events examples

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

The figures accompanying the examples have the same structure:

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

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

The following examples illustrate:

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

Example 1 (queue service interval events)

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

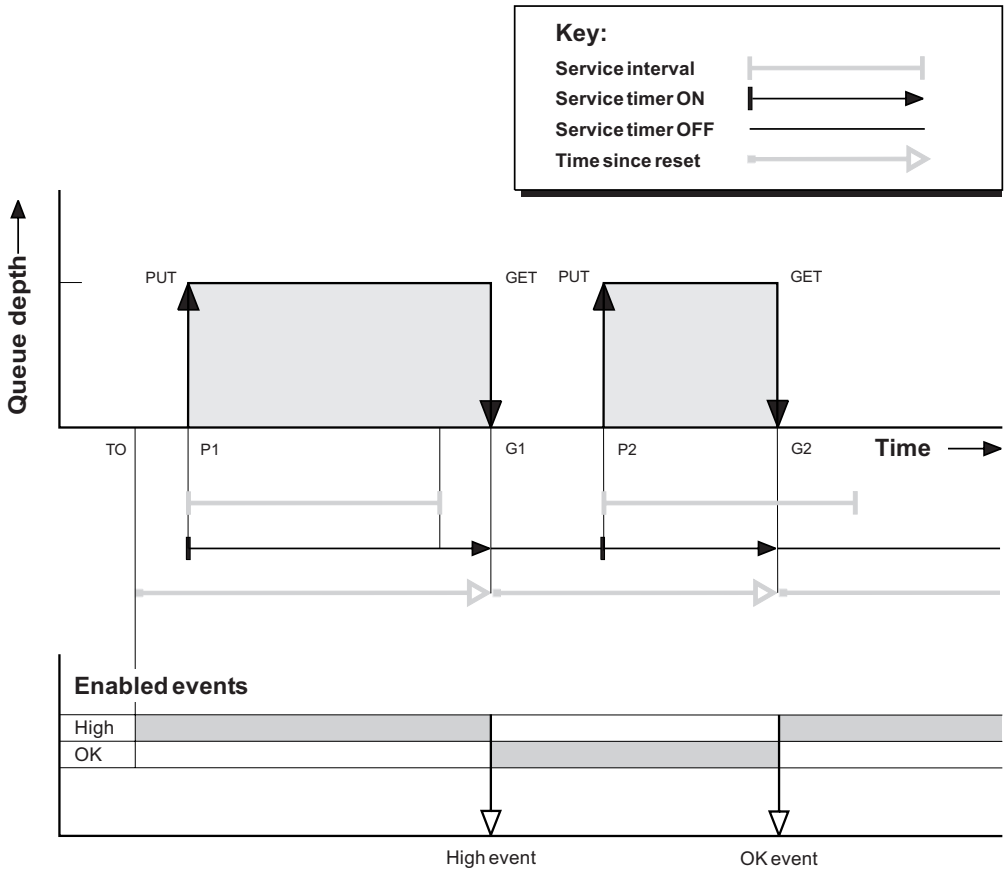

Figure 4. Queue service interval events - example 1

Commentary

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

   Note that T0 may be queue manager startup time.
2. At G1, another application gets the message from the queue. Because the elapsed time between P1 and G1 is greater than the service interval, a Queue

Service Interval High event is generated on the MQGET call at G1. When the high event is generated, the queue manager resets the event control attribute so that:

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

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

3. At P2, a second message is put onto the queue. This restarts the service timer.
4. At G2, the message is removed from the queue. However, because the elapsed time between P2 and G2 is less than the service interval, a Queue Service Interval OK event is generated on the MQGET call at G2. When the OK event is generated, the queue manager resets the control attribute so that:
   a. The high event is automatically enabled.
   b. The OK event is disabled.

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

Event statistics summary for example 1

Table 5 summarizes the event statistics for this example.

Table 5. Event statistics summary for example 1

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

The middle part of Figure 4 on page 20 shows the elapsed time as measured by the service timer compared to the service interval for that queue. To see whether a queue service interval event will occur, compare the length of the horizontal line representing the service timer (with arrow) to that of the line representing the service interval. If the service timer line is longer, and the Queue Service Interval High event is enabled, a Queue Service Interval High event will occur on the next get. If the timer line is shorter, and the Queue Service Interval OK event is enabled, a Queue Service Interval OK event will occur on the next get.

What queue service interval events tell you

You must exercise some caution when you look at queue statistics. Figure 4 on page 20 shows a simple case where the messages are intermittent and each message is removed from the queue before the next one arrives. From the event data, you know that the maximum number of messages on the queue was one. You can, therefore, work out how long each message was on the queue.

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

Example 2 (queue service interval events)

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

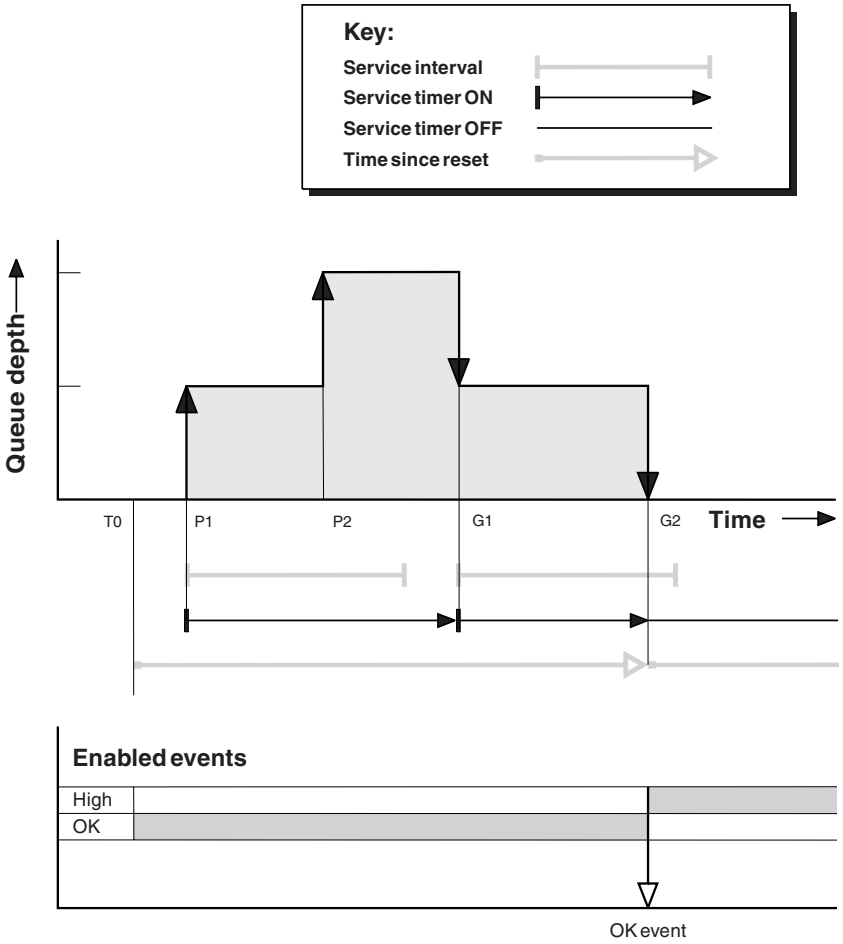

Figure 5. Queue service interval events - example 2

Commentary

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

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

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

Event statistics summary for example 2

Table 6 summarizes the event statistics for this example.

Table 6. Event statistics summary for example 2

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

Example 3 (queue service interval events)

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

Commentary

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

Queue service interval events

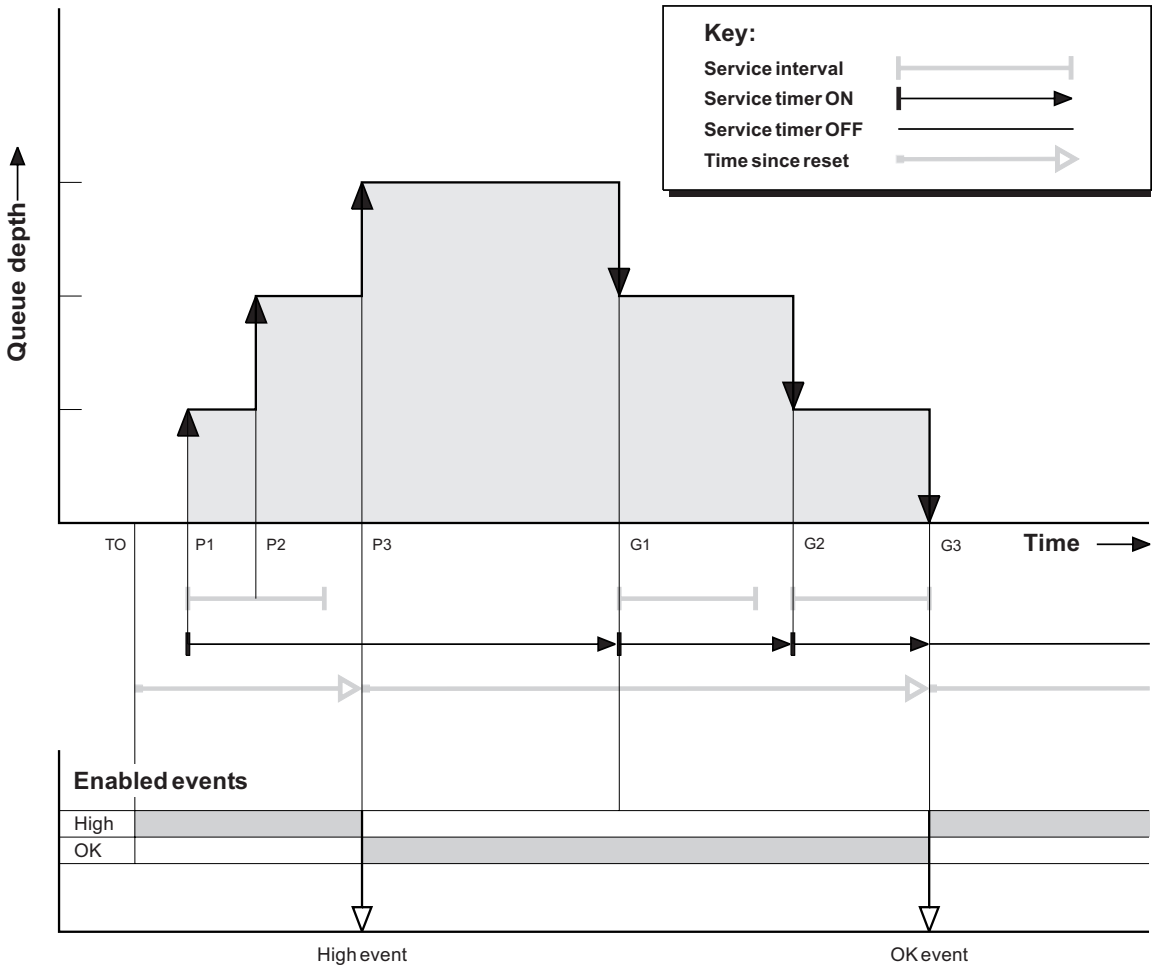

Figure 6. Queue service interval events - example 3

Event statistics summary for example 3

Table 7 summarizes the event statistics for this example.

Table 7. Event statistics summary for example 3

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

Understanding queue depth events

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

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

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

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

What queue depth events are

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

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

A Queue Full Event is generated when an application attempts to put a message on a queue that has reached its maximum depth. Queue Depth High events give advance warning that a queue is filling up. This means that having received this event, the system administrator should take some preventive action. If this action is successful and the queue depth drops to a 'safe' level, the queue manager can be configured to generate a Queue Depth Low event indicating an 'all clear' state.

Figure 8 on page 29 shows a graph of queue depth against time in such a case. The preventive action was (presumably) taken between T_2 and T_3 and continues to have effect until T_4 when the queue depth is well inside the 'safe' zone.

Shared queues and queue depth events (WebSphere MQ for z/OS)

When a queue depth event occurs on a shared queue, the queue managers in the queue-sharing group produce an event message, if the queue manager attribute *PerformanceEvent* (PERFMEV in MQSC) is set to ENABLED. If PERFMEV is set to DISABLED on some of the queue managers, event messages are not produced by those queue managers, making event monitoring from an application more difficult. To avoid this, give each queue manager the same setting for the *PerformanceEvent* attribute. This event message represents the individual usage of the shared queue by each queue manager. If a queue manager performs no activity on the shared queue, various values in the event message are null or zero. Null event messages:

- Allow you to ensure there is one event message for each active queue manager in a queue-sharing group

- Can highlight cases where there has been no activity on a shared queue for a queue manager that produced the event message

Enabling queue depth events

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

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

Enabling queue depth events on shared queues (WebSphere MQ for z/OS)

When a queue manager determines that an event should be issued, the shared queue object definition is updated to toggle the active performance event attributes. For example, depending on the definition of the queue attributes, a Queue Depth High event enables a Queue Depth Low and a Queue Full event. After the shared queue object has been updated successfully, the queue manager that detected the performance event initially becomes the *coordinating queue manager*.

The **coordinating queue manager**:

1. Determines if it has performance events enabled.
2. If it does, issues an event message that captures all shared queue performance data it has gathered since the last time an event message was created, or since the queue statistics were last reset. The message descriptor (MQMD) of this message contains a unique correlation identifier (*CorrelId*) created by the coordinating queue manager.
3. Broadcasts to all other **active** queue managers in the same queue-sharing group to request the production of an event message for the shared queue. The broadcast contains the correlation identifier created by the coordinating queue manager for the set of event messages.

After receiving a request from the coordinating queue manager, an **active queue manager in a queue-sharing group**:

1. Determines if its PERFMEV is ENABLED.
2. If it is, the active queue manager issues an event message for the shared queue, recording all operations performed by the receiving (active) queue manager since the last time an event message was created, or since the last statistics reset. The message descriptor (MQMD) of this event message contains the unique correlation identifier (*CorrelId*) specified by the coordinating queue manager.

When performance events occur on a shared queue, *n* event messages are produced, where *n* is 1 to the number of active queue managers in the queue-sharing group. Each event message contains data that relates to the shared queue activity for the queue manager where the event message was generated.

You can view event message data for a shared queue using the:

- Queue-sharing view.

  All data from event messages with the same correlation identifier is collected here.
- Queue manager view.

  Each event message shows how much it has been used by its originating queue manager.

Differences between shared and nonshared queues: Enabling queue depth events on shared queues differs from enabling events on nonshared queues. A key difference is that events are switched for shared queues even if PERFMEV is DISABLED on the queue manager. This is not the case for nonshared queues.

Consider the following example which illustrates this difference.

- QM1 is a queue manager with *PerformanceEvent* (PERFMEV in MQSC) set to DISABLED.
- SQ1 is a shared queue with QSGDISP set to (SHARED) QLOCAL in MQSC.
- LQ1 is a nonshared queue with QSGDISP set to (QMGR) QLOCAL in MQSC.

Both queues have the following attributes set on their definitions:

- QDPHIEV (ENABLED)
- QDPLOEV (DISABLED)
- QDPMAXEV (DISABLED)

If messages are placed on both queues so that the depth meets or exceeds the QDEPTHHI threshold, the QDPHIEV value on SQ1 switches to DISABLED. Also, QDPLOEV and QDPMAXEV are switched to ENABLED. SQ1's attributes are automatically switched for each performance event at the time the event criteria are met.

In contrast the attributes for LQ1 remain unchanged until PERFMEV on the queue manager is ENABLED. This means that if the queue manager's PERFMEV attribute is ENABLED, DISABLED and then re-ENABLED for instance, the performance event settings on shared queues might not be consistent with those of nonshared queues, even though they might have initially been the same.

Enabling Queue Depth High events

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

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

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

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

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

Enabling Queue Depth Low events

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

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

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

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

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

Enabling Queue Full events

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

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

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

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

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

Table 8. Enabling queue depth events using MQSC

Queue depth event	*Queue attributes*
Queue depth high	QDPHIEV (ENABLED) QDEPTHHI (*hh*) where *hh* is the queue depth high limit.
Queue depth low	QDPLOEV (ENABLED) QDEPTHLO (*ll*) where *ll* is the Queue depth low limit. (Both values are expressed as a percentage of the maximum queue depth, which is specified by the queue attribute MAXDEPTH.)
Queue full	QDPMAXEV (ENABLED)

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

Queue depth events examples

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

Example 1 (queue depth events)

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

The WebSphere MQ commands (MQSC) to configure this queue are:

```
ALTER QMGR PERFMEV(ENABLED)

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

Figure 7. Definition of MYQUEUE1

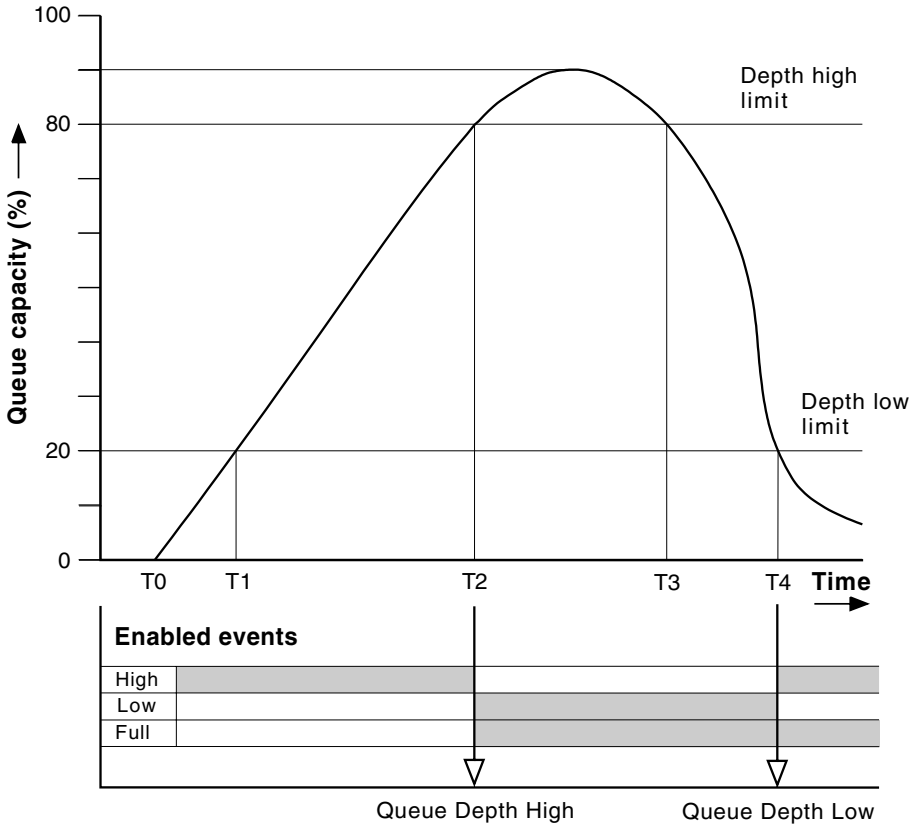

Figure 8. Queue depth events (1)

Commentary

Figure 8 shows how the queue depth changes over time:

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

   This enables both Queue Full and Queue Depth Low events.

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

   This enables both Queue Full and Queue Depth High events.

Table 9 summarizes the queue event statistics and Table 10 summarizes which events are enabled at different times for this example.

Table 9. Event statistics summary for queue depth events (example 1)

	Event 2	**Event 4**
Time of event	T_2	T_4
Type of event	Queue Depth High	Queue Depth Low
TimeSinceReset	T_2 - T_0	T_4 - T_2
HighQDepth (Maximum queue depth since reset)	800	900
MsgEnqCount	1157	1220
MsgDeqCount	357	1820

Table 10. Summary showing which events are enabled

Time period	**Queue Depth High event**	**Queue Depth Low event**	**Queue Full event**
Before T_1	ENABLED	-	-
T_1 to T_2	ENABLED	-	-
T_2 to T_3	-	ENABLED	ENABLED
T_3 to T_4	-	ENABLED	ENABLED
After T_4	ENABLED	-	ENABLED

Example 2 (queue depth events)

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

Table 11 on page 32 summarizes the queue event statistics and Table 12 on page 32 summarizes which events are enabled at different times for this example.

Figure 9 on page 31 shows the variation of queue depth over time.

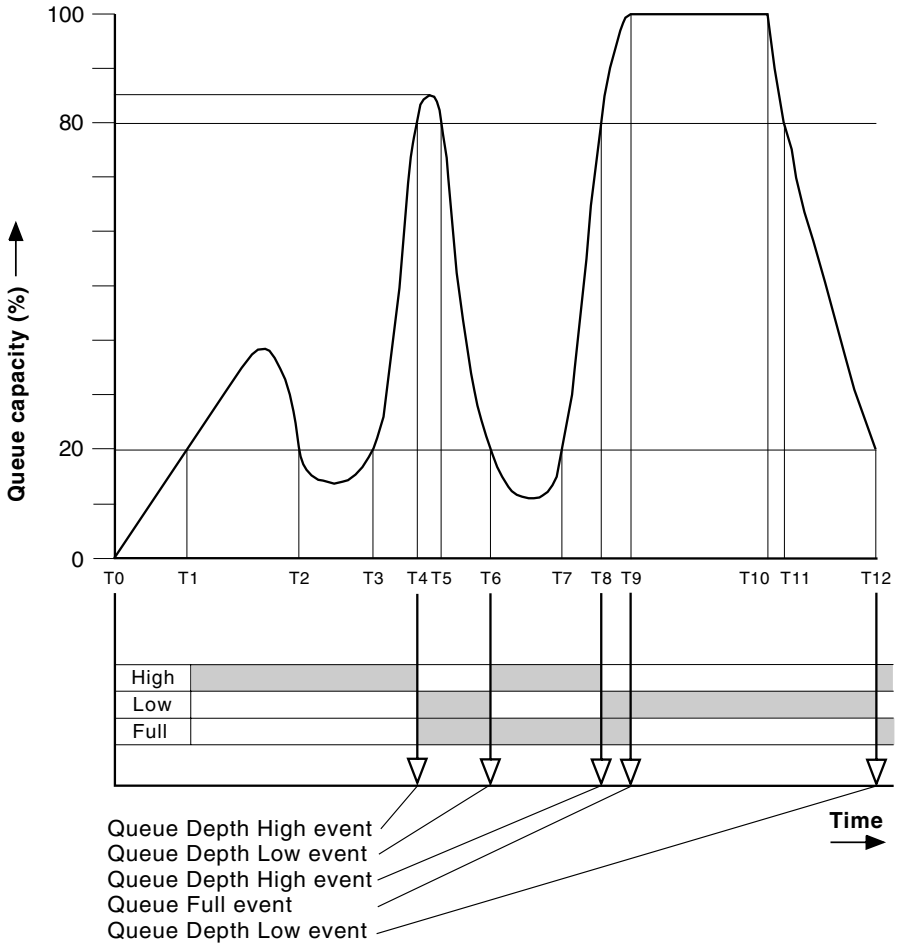

Figure 9. Queue depth events (2)

Commentary

Some points to note are:

1. No Queue Depth Low event is generated at:
   T_1 (Queue depth increasing, and not enabled)
   T_2 (Not enabled)
   T_3 (Queue depth increasing, and not enabled)
2. At T_4 a Queue Depth High event occurs. This enables both Queue Full and Queue Depth Low events.
3. At T_9 a Queue Full event occurs **after** the first message that cannot be put on the queue because the queue is full.
4. At T_{12} a Queue Depth Low event occurs.

Event statistics summary (example 2)

Table 11. Event statistics summary for queue depth events (example 2)

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

Table 12. Summary showing which events are enabled

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

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

Chapter 3. Understanding configuration events (z/OS only)

This chapter describes what configuration events are, when they are generated, when they are not generated, and how they are used. The chapter includes:

- "What configuration events are"
- "When configuration events are generated"
- "When configuration events are not generated" on page 34
- "How configuration events are used" on page 34

What configuration events are

Configuration events are notifications about the attributes of an object. They are generated when an object is created, changed, or deleted and are also generated by explicit requests. There are four types of configuration events:

- Create object events
- Change object events
- Delete object events
- Refresh object events

The event data contains the following information:

1. **Origin information**, describes the queue manager from where the change was made, the ID of the user that made the change, and how the change came about, for example by a console command.
2. **Context information**, is a replica of the context information in the message data from the command message.

   Note: Context information will only be included in the event data if the command was entered as a message on the SYSTEM.COMMAND.INPUT queue.
3. **Object identity**, describes the name, type and disposition of the object.
4. **Object attributes**, describes the values of all the attributes in the object.

In the case of change object events, two messages are generated, one with the information before the change, the other with the information after.

Every configuration event message that is generated is placed on the queue SYSTEM.ADMIN.CONFIG.EVENT.

For more information about the event data returned in configuration event messages see: Chapter 4, "Event message reference", on page 37

When configuration events are generated

A configuration event message is put to the configuration event queue when the CONFIGEV queue manager attribute is ENABLED and :

— any of the following commands are issued.

- DELETE AUTHINFO
- DELETE CFSTRUCT

- DELETE CHANNEL
- DELETE NAMELIST
- DELETE PROCESS
- DELETE QMODEL/QALIAS/QREMOTE
- DELETE STGCLASS
- REFRESH QMGR

— any of the following commands are issued, even if there is no change to the object.

- DEFINE/ALTER AUTHINFO
- DEFINE/ALTER CFSTRUCT
- DEFINE/ALTER CHANNEL
- DEFINE/ALTER NAMELIST
- DEFINE/ALTER PROCESS
- DEFINE/ALTER QMODEL/QALIAS/QREMOTE
- DEFINE/ALTER STGCLASS
- DEFINE MAXSMSGS
- ALTER QMGR unless the CONFIGEV attribute is DISABLED and is not changed to ENABLED

— any of the following commands are issued for a local queue that is not temporary dynamic, even if there is no change to the queue.

- DELETE QLOCAL
- DEFINE/ALTER QLOCAL

— an MQSET is issued, other than for a temporary dynamic queue, even if there is no change to the object.

When configuration events are not generated

Configuration events messages are not generated for:

- Command or MQSET calls that fail.
- A queue manager that encounters an error trying to put a configuration event on the event queue. In this situation, the command or MQSET call completes, but no event message is generated.
- Temporary dynamic queues.
- Internal changes to the TRIGGER queue attribute.
- The configuration event queue SYSTEM.ADMIN.CONFIG.EVENT, except by the REFRESH QMGR command.

How configuration events are used

Configuration events are used for two main purposes:

1. To produce and maintain a central configuration repository, from which reports can be produced and information about the structure of the system can be generated.
2. To generate an audit trail. For example, if an object is changed unexpectedly, information regarding who made the alteration and when it was done can be stored.

The Refresh Object configuration event

The Refresh Object configuration event is different from the other configuration events. The create, change, and delete events are generated by an MQSET call or by a command to change an object but the refresh object event occurs only when it is explicitly requested by the REFRESH QMGR command.

The REFRESH QMGR command is different from all the other commands that generate configuration events. All the other commands apply to a particular object and generate a single configuration event for that object. The REFRESH QMGR command can produce many configuration event messages potentially representing every object definition stored by a queue manager. One event message is generated for each object that is selected.

The REFRESH QMGR command uses a combination of three selection criteria to filter the number of objects involved:

- Object Name
- Object Type
- Refresh Interval

If you specify none of the selection criteria on the REFRESH QMGR command, the default values are used for each selection criteria and a refresh configuration event message is generated for every object definition stored by the queue manager. This may well involve excessive processing time and event message generation. It is recommended to always use some selection criteria.

The REFRESH QMGR command that generates the refresh events can be used in the following situations:

- When configuration data is wanted about all or some of the objects in a system regardless of whether the objects have been recently manipulated, for example, when configuration events are first enabled.

  It is recommended to use several commands each with a different selection of objects, but such that all are included.
- If there has been an error in the SYSTEM.ADMIN.CONFIG.EVENT queue. In this circumstance, no configuration event messages are generated for Create, Change, or Delete events. When the error on the queue has been corrected, the Refresh Queue Manager command can be used to request the generation of event messages, which were lost whilst there was an error in the queue. In this situation it is recommended that you use the refresh selection criteria, with the refresh interval being the time for which the queue was unavailable.

When the configuration event queue is not available

For an MQSET call or any of the following commands:

- DEFINE object
- ALTER object
- DELETE object

if the queue manager attribute CONFIGEV is enabled, but the configuration event message cannot be put on the configuration event queue, for example the event queue has not been defined, the command or MQSET call will NOT fail and the object WILL be created or manipulated in the intended way.

Effects of CMDSCOPE

For commands where CMDSCOPE is used, the configuration event message or messages will be generated on the queue manager or queue managers where the command is executed, not where the command is entered. However, all the origin and context information in the event data will relate to the original command as entered, even where the command using CMDSCOPE is one that has been generated by the source queue manager.

Where a queue sharing group includes queue managers that are not at the current version, events will be generated for any command that is executed by means of CMDSCOPE on a queue manager that is at the current version, but not on those that are at a previous version. This happens even if the queue manager where the command is entered is at the previous version, although in such a case no context information is included in the event data.

Chapter 4. Event message reference

This chapter provides an overview of the event message format. It describes the information returned in the event message for each instrumentation event, including returned parameters.

The chapter includes:

Event message format

Event messages are standard WebSphere MQ messages containing a message descriptor and message data.

Table 13 on page 39 shows the basic structure of these messages, and the names of the fields in an event message for queue service interval events.

Table 13. Event message structure for queue service interval events

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

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

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

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

Message descriptor (MQMD) in event messages

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

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

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

In an event message, many of these fields contain fixed data, which is supplied by the queue manager that generated the message. The fields that make up the MQMD structure are described in "MQMD (message descriptor)" below. The MQMD also specifies the name of the queue manager (truncated to 28 characters) that put the message, and the date and time when the event message was put on the event queue.

Message data in event messages

The event message data is based on the programmable command format (PCF), which is used in PCF command inquiries and responses.

The event message consists of two parts: the event header and the event data.

Event header (MQCFH)

The information in MQCFH specifies:

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

Reason codes have names that begin with the characters MQRC_. For example, the reason code MQRC_PUT_INHIBITED is generated when an application attempts to put a message on a queue that is not enabled for puts. MQCFH is described in "MQCFH (Event header)" on page 44.

Event data

See "Event message descriptions" on page 49.

MQMD (message descriptor)

The MQMD structure describes the information that accompanies the message data of an event message. For a full description of MQMD, including a description of the elementary datatype of each parameter, see the *WebSphere MQ Application Programming Reference* manual.

For an event message, the MQMD structure contains these values:

StrucId

Description: Structure identifier.
Datatype: MQCHAR4.
Value: MQMD_STRUC_ID

Version

Description: Structure version number.
Datatype: MQLONG.

Values:

MQMD_VERSION_1
: Version-1 message descriptor structure, supported in all environments.

MQMD_VERSION_2
: Version-2 message descriptor structure, supported on AIX®, DOS client, HP-UX, OS/2®, OS/400, Solaris, Windows NT client and Windows platforms.

Report

Description: Options for report messages.
Datatype: MQLONG.
Value:

MQRO_NONE
: No reports required.

MsgType

Description: Indicates type of message.
Datatype: MQLONG.
Value: MQMT_DATAGRAM.

Expiry

Description: Message lifetime.
Datatype: MQLONG.
Value:

MQEI_UNLIMITED
: The message does not have an expiry time.

Feedback

Description: Feedback or reason code.
Datatype: MQLONG.
Value: MQFB_NONE.

Encoding

Description: Numeric encoding of message data.
Datatype: MQLONG.
Value: MQENC_NATIVE.

CodedCharSetId

Description: Character set identifier of event message data.
Datatype: MQLONG.
Value:

MQCCSI_Q_MQR
: Coded character set ID (CCSID) of the queue manager generating the event.

Format

Description: Format name of message data.
Datatype: MQCHAR8.

Message descriptor

Value:

MQFMT_EVENT
: Event message.

  For the C programming language, the constant MQFMT_EVENT_ARRAY is also defined; this has the same value as MQFMT_EVENT, but is an array of characters not a string.

Priority

Description: Message priority.
Datatype: MQLONG.
Value:

MQPRI_PRIORITY_AS_Q_DEF
: Default priority of the event queue, if it is a local queue, or its local definition at the queue manager generating the event.

Persistence

Description: Message persistence.
Datatype: MQLONG.
Value:

MQPER_PERSISTENCE_AS_Q_DEF
: Default persistence of the event queue, if it is a local queue, or its local definition at the queue manager generating the event.

MsgId

Description: Message identifier.
Datatype: MQBYTE24.
Initial value: MQMI_NONE.
Valid values: A unique value generated by the queue manager.

CorrelId

Description: Correlation identifier.
Datatype: MQBYTE24.
Initial value: MQCI_NONE.
Valid values:

MQCI_NONE
: No correlation identifier is specified. This is for local queues only.

For shared queues: a nonzero correlation identifier is set. For events on a shared queue, this parameter is set, so you can track multiple event messages from different queue managers. The characters are specified below:

1–4 Product identifier ('CSQ ')

5–8 Queue-sharing group name

9 Queue manager identifier

10–17 Time stamp

18–24 Nulls

BackoutCount

Description: Backout counter.

Datatype: MQLONG.
Value: 0.

ReplyToQ

Description: Name of reply queue.
Datatype: MQCHAR48.
Values: Blank.

ReplyToQMgr

Description: Name of reply queue manager.
Datatype: MQCHAR48.
Value: The queue manager name at the originating system.

UserIdentifier

Description: Identifies the application that originated the message.
Datatype: MQCHAR12.
Value: Blank.

AccountingToken

Description: Accounting token that allows an application to charge for work done as a result of the message.
Datatype: MQBYTE32.
Value: MQACT_NONE.

ApplIdentityData

Description: Application data relating to identity.
Datatype: MQCHAR32.
Values: Blank.

PutApplType

Description: Type of application that put the message.
Datatype: MQLONG.
Value:

MQAT_QMGR
: Queue-manager-generated message.

PutApplName

Description: Name of application that put the message.
Datatype: MQCHAR28.
Value: The queue manager name at the originating system.

PutDate

Description: Date when message was put.
Datatype: MQCHAR8.
Value: As generated by the queue manager.

Message descriptor

PutTime

Description: Time when message was put.
Datatype: MQCHAR8.
Value: As generated by the queue manager.

ApplOriginData

Description: Application data relating to origin.
Datatype: MQCHAR4.
Value: Blank.

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

GroupId

Description: Identifies to which message group or logical message the physical message belongs.
Datatype: MQBYTE24.
Value:
MQGI_NONE
No group identifier specified.

MsgSeqNumber

Description: Sequence number of logical message within group.
Datatype: MQLONG.
Value: 1.

Offset

Description: Offset of data in physical message from start of logical message.
Datatype: MQLONG.
Value: 0.

MsgFlags

Description: Message flags that specify attributes of the message or control its processing.
Datatype: MQLONG.
Value: MQMF_NONE.

OriginalLength

Description: Length of original message.
Datatype: MQLONG.
Value: MQOL_UNDEFINED.

MQCFH (Event header)

The MQCFH structure is the header structure used for event messages and for PCF messages. When the structure is used for event messages, the message descriptor *Format* field is MQFMT_EVENT. The datatype of the following parameters (MQLONG) is described in the *WebSphere MQ Application Programming Reference* manual.

For an event, the MQCFH structure contains these values:

Type

Description: Structure type that identifies the content of the message.
Datatype: MQLONG.
Value:

MQCFT_EVENT
: Message is reporting an event.

StrucLength

Description: Structure length.
Datatype: MQLONG.
Value:

MQCFH_STRUC_LENGTH
: Length in bytes of MQCFH structure.

Version

Description: Structure version number.
Datatype: MQLONG.
Values:

MQCFH_VERSION_1
: Version-1 in all events except configuration events.

MQCFH_VERSION_2
: Version-2 for configuration events.

Command

Description: Command identifier. This identifies the event category.
Datatype: MQLONG.
Values:

MQCMD_Q_MGR_EVENT
: Queue manager event.

MQCMD_PERFM_EVENT
: Performance event.

MQCMD_CHANNEL_EVENT
: Channel event.

MQCMD_CONFIG_EVENT
: Configuration event.

MsgSeqNumber

Description: Message sequence number. This is the sequence number of the message within a group of related messages.
Datatype: MQLONG.
Values:

1
: For change object configuration events with attribute values before the changes, and for all other types of events.

2
: For change object configuration events with the attribute values after the changes

Control

Description: Control options.
Datatype: MQLONG.

Values:

MQCFC_LAST
: For change object configuration events with attribute values after the changes, and for all other types of events.

MQCFC_NOT_LAST
: For Change Object configurations events only, with the attribute values from before the changes.

CompCode

Description: Completion code.
Datatype: MQLONG.
Values:

MQCC_OK
: Event reporting OK condition.

MQCC_WARNING
: Event reporting warning condition. All events have this completion code, unless otherwise specified.

Reason

Description: Reason code qualifying completion code.
Datatype: MQLONG.
Values: MQRC_* Dependent on the event being reported.
Note: Events with the same reason code are further identified by the *ReasonQualifier* parameter in the event data.

ParameterCount

Description: Count of parameter structures. This is the number of parameter structures (MQCFIN, MQCFST, MQCFSL and MQCFBS) that follow the MQCFH structure.
Datatype: MQLONG.
Values: 0 or greater.

C language declaration (MQCFH)

```
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)

```
**   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 (MQCFH)

```
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 */
```

RPG/ILE declaration (MQCFH) (OS/400 only)

```
     D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
     D* MQCFH Structure
     D*
     D* Structure type
     D  FHTYP                  1      4I 0
     D* Structure length
     D  FHLEN                  5      8I 0
     D* Structure version number
     D  FHVER                  9     12I 0
     D* Command identifier
     D  FHCMD                 13     16I 0
     D* Message sequence number
     D  FHSEQ                 17     20I 0
     D* Control options
     D  FHCTL                 21     24I 0
     D* Completion code
     D  FHCMP                 25     28I 0
     D* Reason code qualifying completion code
     D  FHREA                 29     32I 0
     D* Count of parameter structures
     D  FHCNT                 33     36I 0
```

System/390® assembler-language declaration (MQCFH) (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
```

Event header

```
MQCFH_LENGTH                  EQU  *-MQCFH  Length of structure
                              ORG  MQCFH
MQCFH_AREA                    DS   CL(MQCFH_LENGTH)
```

Visual Basic® language declaration (MQCFH) (Windows platforms 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
```

Event message descriptions

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

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

Notes

1. The event structures in the event data are not returned in a defined order. They must be identified from the parameter identifiers shown in the description.
2. The events described in the reference section are available on all platforms, unless specific limitations are shown at the start of an event.
3. The structure datatypes of each parameter are described in Appendix A, “Structure datatypes MQCFBS, MQCFIN, MQCFSL and MQCFST”, on page 145.

Alias Base Queue Type Error

Event name:	Alias Base Queue Type Error.
Reason code in MQCFH:	MQRC_ALIAS_BASE_Q_TYPE_ERROR (2001, X'7D1'). Alias base queue not a valid type.
Event description:	An MQOPEN or MQPUT1 call was issued specifying an alias queue as the destination, but the *BaseQName* in the alias queue definition resolves to a queue that is not a local queue, or local definition of a remote queue.
Event type:	Local.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

QName

Description: Queue name from object descriptor (MQOD).
Identifier: MQCA_Q_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

BaseQName

Description: Queue name to which the alias resolves.
Identifier: MQCA_BASE_Q_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

QType

Description: Type of queue to which the alias resolves.
Identifier: MQIA_Q_TYPE.
Datatype: MQCFIN.
Values:

MQQT_ALIAS
: Alias queue definition.

MQQT_MODEL
: Model queue definition.

Returned: Always.

ApplType

Description: Type of the application making the call that caused the event.

Identifier: MQIA_APPL_TYPE.
Datatype: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application making the call that caused the event.
Identifier: MQCACF_APPL_NAME.
Datatype: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

ObjectQMgrName

Description: Name of the object queue manager.
Identifier: MQCACF_OBJECT_Q_MGR_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: If the *ObjectName* in the object descriptor (MQOD), when the object was opened, is not the queue manager currently connected.

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

Bridge Started (z/OS only)

Event name:	Bridge Started.
Reason code in MQCFH:	MQRC_BRIDGE_STARTED (2125, X'84D'). Bridge started.
Event description:	The IMS bridge has been started.
Event type:	IMS Bridge.
Platforms:	WebSphere MQ for z/OS only.
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Data type:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

BridgeType

Description:	Bridge type.
Identifier:	MQIACF_BRIDGE_TYPE.
Data type:	MQCFIN.
Values:	MQBT_OTMA OTMA bridge.
Returned:	Always.

BridgeName

Description:	Bridge name. For bridges of type MQBT_OTMA, the name is of the form `XCFgroupXCFmember`, where `XCFgroup` is the XCF group name to which both IMS and WebSphere MQ belong. `XCFmember` is the XCF member name of the IMS system.
Identifier:	MQCACF_BRIDGE_NAME.
Data type:	MQCFST.
Maximum length:	MQ_BRIDGE_NAME_LENGTH.
Returned:	Always.

Bridge Stopped (z/OS only)

Event name:	Bridge Stopped.
Reason code in MQCFH:	MQRC_BRIDGE_STOPPED (2126, X'84E'). Bridge stopped.
Event description:	The IMS bridge has been stopped.
Event type:	IMS Bridge.
Platforms:	WebSphere MQ for z/OS only.
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

ReasonQualifier

Description: Identifier that qualifies the reason code in MQCFH.
Identifier: MQIACF_REASON_QUALIFIER.
Datatype: MQCFIN.
Values:

MQRQ_BRIDGE_STOPPED_OK
: Bridge has been stopped with either a zero return code or a warning return code. For MQBT_OTMA bridges, one side or the other issued a normal IXCLEAVE request.

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

Returned: Always.

BridgeType

Description: Bridge type.
Identifier: MQIACF_BRIDGE_TYPE.
Datatype: MQCFIN.
Value:

MQBT_OTMA
: OTMA bridge.

Returned: Always.

BridgeName

Description: Bridge name. For bridges of type MQBT_OTMA, the name is of the form `XCFgroupXCFmember`, where `XCFgroup` is the XCF group name to which both IMS and WebSphere MQ belong. `XCFmember` is the XCF member name of the IMS system.
Identifier: MQCACF_BRIDGE_NAME.
Datatype: MQCFST.
Maximum length: MQ_BRIDGE_NAME_LENGTH.
Returned: Always.

Bridge Stopped

ErrorIdentifier

Description:	When a bridge is stopped due to an error, this code identifies the error. If the event reports a bridge stop failure, the IMS sense code is set.
Identifier:	MQIACF_ERROR_IDENTIFIER.
Datatype:	MQCFIN.
Returned:	If *ReasonQualifier* is MQRQ_BRIDGE_STOPPED_ERROR.

Change object (z/OS only)

Event name:	Change object.
Reason code in MQCFH:	MQRC_CONFIG_CHANGE_OBJECT (2368, X'940'). Existing object changed.
Event description:	An ALTER or DEFINE REPLACE command or an MQSET call was issued that successfully changed an existing object.
Event type:	Configuration.
Platforms:	WebSphere MQ for z/OS.
Event queue:	SYSTEM.ADMIN.CONFIG.EVENT.

Note: Two event messages are generated for the change object event. The first has the object attribute values **before** the change, the second has the attribute values **after** the change.

Event data

EventUserId

Description: The user id that issued the command or call that generated the event. (This is the same user id that is used to check the authority to issue the command or call; for commands received from a queue, this is also the user identifier (UserIdentifier) from the MD of the command message).
Identifier: MQCACF_EVENT_USER_ID.
Datatype: MQCFST.
Maximum length: MQ_USER_ID_LENGTH.
Returned: Always.

EventOrigin

Description: The origin of the action causing the event.
Identifier: MQIACF_EVENT_ORIGIN.
Datatype: MQCFIN.
Values:

MQEVO_CONSOLE
: Console.

MQEVO_INIT
: Initialization input data set.

MQEVO_MSG
: Message on SYSTEM.COMMAND.INPUT.

MQEVO_MQSET
: MQSET call.

MQEVO_INTERNAL
: Directly by queue manager.

MQEVO_OTHER
: None of the above.

Returned: Always.

Change object

EventQMgr

Description: The queue manager where the command or call was entered. (The queue manager where the command is executed and that generates the event is in the MD of the event message).
Identifier: MQCACF_EVENT_Q_MGR.
Datatype: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

EventAccountingToken

Description: For commands received as a message (MQEVO_MSG), the accounting token (AccountingToken) from the MD of the command message.
Identifier: MQBACF_EVENT_ACCOUNTING_TOKEN.
Datatype: MQCFBS.
Maximum length: MQ_ACCOUNTING_TOKEN_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplIdentity

Description: For commands received as a message (MQEVO_MSG), application identity data (ApplIdentityData) from the MD of the command message.
Identifier: MQCACF_EVENT_APPL_IDENTITY.
Datatype: MQCFST.
Maximum length: MQ_APPL_IDENTITY_DATA_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplType

Description: For commands received as a message (MQEVO_MSG), the type of application (PutApplType) from the MD of the command message.
Identifier: MQIACF_EVENT_APPL_TYPE.
Datatype: MQCFIN.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplName

Description: For commands received as a message (MQEVO_MSG), the name of the application (PutApplName) from the MD of the command message.
Identifier: MQCACF_EVENT_APPL_NAME.
Datatype: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplOrigin

Description: For commands received as a message (MQEVO_MSG), the application origin data (ApplOriginData) from the MD of the command message.
Identifier: MQCACF_EVENT_APPL_ORIGIN.
Datatype: MQCFST.
Maximum length: MQ_APPL_ORIGIN_DATA_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

ObjectType

Description: Object type:
Identifier: MQIACF_OBJECT_TYPE.
Datatype: MQCFIN.
Values:

MQOT_CHANNEL
: Channel.

MQOT_NAMELIST
: Namelist.

MQOT_PROCESS
: Process.

MQOT_Q
: Queue.

MQOT_STORAGE_CLASS
: Storage class.

MQOT_Q_MGR
: Queue manager.

MQOT_AUTH_INFO
: Authentication information.

MQOT_CF_STRUC
: CF structure.

Returned: Always.

ObjectName

Description: Object name:
Identifier : Identifier will be according to object type.

- MQCACH_CHANNEL_NAME
- MQCA_NAMELIST_NAME
- MQCA_PROCESS_NAME
- MQCA_Q_NAME
- MQCA_STORAGE_CLASS
- MQCA_Q_MGR_NAME
- MQCA_AUTH_INFO_NAME
- MQCA_CF_STRUC_NAME

Datatype: MQCFST.
Maximum length: MQ_OBJECT_NAME_LENGTH.
Returned: Always

Disposition

Description: Object disposition:
Identifier: MQIA_QSG_DISP.
Datatype: MQCFIN.

Values:

MQQSGD_Q_MGR
: Object resides on page set of queue manager.

MQQSGD_SHARED
: Object resides in shared repository and messages are shared in coupling facility.

MQQSGD_GROUP
: Object resides in shared repository.

MQQSGD_COPY
: Object resides on page set of queue manager and is a local copy of a GROUP object.

Returned: Always, except for queue manager and CF structure objects.

Object attributes

A parameter structure is returned for each attribute of the object. The attributes returned depend on the object type. For more information see Appendix D, “Event data for object attributes”, on page 167

Channel Activated

Event name:	Channel Activated.
Reason code in MQCFH:	MQRC_CHANNEL_ACTIVATED (2295, X'8F7'). Channel activated.
Event description:	This condition is detected when a channel that has been waiting to become active, and for which a Channel Not Activated event has been generated, is now able to become active, because an active slot has been released by another channel. This event is not generated for a channel that is able to become active without waiting for an active slot to be released.
Event type:	Channel.
Platforms:	All, except WebSphere MQ for z/OS if CICS is used for distributed queue management.
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

ChannelName

Description:	Channel Name.
Identifier:	MQCACH_CHANNEL_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_CHANNEL_NAME_LENGTH.
Returned:	Always.

XmitQName

Description:	Transmission queue name.
Identifier:	MQCACH_XMIT_Q_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	For sender, server, cluster-sender, and cluster-receiver channels only.

ConnectionName

Description:	If the channel has successfully established a TCP connection, this is the Internet address. Otherwise it is the contents of the *ConnectionName* field in the channel definition.
Identifier:	MQCACH_CONNECTION_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_CONN_NAME_LENGTH.
Returned:	Only for commands that do not contain a generic name.

Channel Auto-definition Error

Event name:	Channel Auto-definition Error.
Reason code in MQCFH:	MQRC_CHANNEL_AUTO_DEF_ERROR (2234, X'8BA'). Automatic channel definition failed.
Event description:	This condition is detected when the automatic definition of a channel fails; this may be because an error occurred during the definition process, or because the channel automatic-definition exit inhibited the definition. Additional information indicating the reason for the failure is returned in the event message.
Event type:	Channel.
Platforms:	Any MQSeries Version 5 product or later release, except WebSphere MQ for z/OS when using CICS for distributed queuing.
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

ChannelName

Description:	Name of the channel for which the auto-definiton has failed.
Identifier:	MQCACH_CHANNEL_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_CHANNEL_NAME_LENGTH.
Returned:	Always.

ChannelType

Description:	Channel Type. This specifies the type of channel for which the auto-definition has failed.
Identifier:	MQIACH_CHANNEL_TYPE.
Datatype:	MQCFIN.
Values:	MQCHT_RECEIVER Receiver. MQCHT_SVRCONN Server-connection (for use by clients). MQCHT_CLUSSDR Cluster-sender.
Returned:	Always.

ErrorIdentifier

Description:	Identifier of the cause of the error. This contains either the reason code (MQRC_* or MQRCCF_*) resulting from the channel definition attempt or the value MQRCCF_SUPPRESSED_BY_EXIT if the attempt to create the definition was disallowed by the exit.
Identifier:	MQIACF_ERROR_IDENTIFIER.

Datatype: MQCFIN.
Returned: Always.

ConnectionName

Description: Name of the partner attempting to establish connection.
Identifier: MQCACH_CONNECTION_NAME.
Datatype: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: Always.

AuxErrorDataInt1

Description: Auxiliary error data. This contains the value returned by the exit in the *Feedback* field of the MQCXP to indicate why the auto definition has been disallowed.
Identifier: MQIACF_AUX_ERROR_DATA_INT_1.
Datatype: MQCFIN.
Returned: Only if *ErrorIdentifier* contains MQRCCF_SUPPRESSED_BY_EXIT.

Channel Auto-definition OK

Event name:	Channel Auto-definition OK.
Reason code in MQCFH:	MQRC_CHANNEL_AUTO_DEF_OK (2233, X'8B9'). Automatic channel definition succeeded.
Event description:	This condition is detected when the automatic definition of a channel is successful. The channel is defined by the MCA.
Event type:	Channel.
Platforms:	Any MQSeries Version 5 product or later release, except WebSphere MQ for z/OS when using CICS for distributed queuing.
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

ChannelName

Description: Name of the channel being defined.
Identifier: MQCACH_CHANNEL_NAME.
Datatype: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH.
Returned: Always.

ChannelType

Description: Type of channel being defined.
Identifier: MQIACH_CHANNEL_TYPE.
Datatype: MQCFIN.
Values:

MQCHT_RECEIVER
: Receiver.

MQCHT_SVRCONN
: Server-connection (for use by clients).

MQCHT_CLUSSDR
: Cluster-sender.

Returned: Always.

ConnectionName

Description: Name of the partner attempting to establish connection.
Identifier: MQCACH_CONNECTION_NAME.
Datatype: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: Always.

Channel Conversion Error

Event name:	Channel Conversion Error.
Reason code in MQCFH:	MQRC_CHANNEL_CONV_ERROR (2284, X'8EC'). Channel conversion error.
Event description:	This condition is detected when a channel is unable to carry out data conversion and the MQGET call to get a message from the transmission queue resulted in a data conversion error. The reason for the failure is identified by *ConversionReasonCode*.
Event type:	Channel.
Platforms:	All, except WebSphere MQ for z/OS if CICS is used for distributed queue management.
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT.

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

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

ConversionReasonCode

Description:	Identifier of the cause of the conversion error.
Identifier:	MQIACF_CONV_REASON_CODE.
Datatype:	MQCFIN.

Channel Conversion Error

Values:

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

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

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

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

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

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

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

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

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

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

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

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

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

Returned: Always.

ChannelName

Description: Channel name.
Identifier: MQCACH_CHANNEL_NAME.
Datatype: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH.
Returned: Always.

Format

Description: Format name.
Identifier: MQCACH_FORMAT_NAME.
Datatype: MQCFST.
Maximum length: MQ_FORMAT_LENGTH.
Returned: Always.

XmitQName

Description: Transmission queue name.
Identifier: MQCACH_XMIT_Q_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

ConnectionName

Description:	If the channel has successfully established a TCP connection, this is the Internet address. Otherwise it is the contents of the *ConnectionName* field in the channel definition.
Identifier:	MQCACH_CONNECTION_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_CONN_NAME_LENGTH.
Returned:	Always.

Channel Not Activated

Event name:	Channel Not Activated.
Reason code in MQCFH:	MQRC_CHANNEL_NOT_ACTIVATED (2296, X'8F8'). Channel cannot be activated.
Event description:	This condition is detected when a channel is required to become active, either because it is starting, or because it is about to make another attempt to establish connection with its partner. However, it is unable to do so because the limit on the number of active channels has been reached. See the: • MaxActiveChannels parameter in the qm.ini file for OS/2, AIX, HP-UX, and Solaris • MaxActiveChannels parameter in the Registry for Windows NT • ACTCHL parameter in CSQXPARM for z/OS The channel waits until it is able to take over an active slot released when another channel ceases to be active. At that time a Channel Activated event is generated.
Event type:	Channel.
Platforms:	All, except WebSphere MQ for z/OS if CICS is used for distributed queue management.
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT.

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

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

ChannelName

Description:	Channel name.
Identifier:	MQCACH_CHANNEL_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_CHANNEL_NAME_LENGTH.
Returned:	Always.

XmitQName

Description:	Transmission queue name.
Identifier:	MQCACH_XMIT_Q_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.

Returned: For sender, server, cluster-sender, and cluster-receiver channel types only.

ConnectionName

Description: If the channel has successfully established a TCP connection, this is the Internet address. Otherwise it is the contents of the *ConnectionName* field in the channel definition.
Identifier: MQCACH_CONNECTION_NAME.
Datatype: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: Only for commands that do not contain a generic name.

Channel SSL Error

Event name:	Channel SSL Error.
Reason code in MQCFH:	MQRC_CHANNEL_SSL_ERROR (2371, X'943'). Channel SSL Error.
Event description:	This condition is detected when a channel using Secure Sockets Layer (SSL) fails to establish an SSL connection. *ReasonQualifier* identifies the nature of the error.
Event type:	SSL.
Platforms:	• WebSphere MQ for AIX • WebSphere MQ for HP-UX • WebSphere MQ for iSeries • WebSphere MQ for Linux • WebSphere MQ for Solaris • WebSphere MQ for Windows • WebSphere MQ for z/OS
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

ReasonQualifier

Description:	Identifier that qualifies the reason code.
Identifier:	MQIACF_REASON_QUALIFIER.
Datatype:	MQCFIN.
Values:	MQRQ_SSL_HANDSHAKE_ERROR The key exchange / authentication failure arose during the SSL handshake. MQRQ_SSL_CIPHER_SPEC_ERROR This error can mean any one of the following: • The SSL client CipherSpec does not match that on the SSL server channel definition. • An invalid CipherSpec has been specified. • A CipherSpec has only been specified on one end of the SSL channel. MQRQ_SSL_PEER_NAME_ERROR The Distinguished Name in the certificate sent by one end of the SSL channel does not match the peer name on the end of the channel definition at the other end of the SSL channel. MQRQ_SSL_CLIENT_AUTH_ERROR The SSL server channel definition specified either SSLCAUTH(REQUIRED) or a SSLPEER value that was not blank, but the SSL client did not provide a certificate.
Returned:	Always.

ChannelName

Description: Channel Name.
Identifier: MQCACH_CHANNEL_NAME.
Datatype: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH.
Returned: The *ChannelName* may not be available if the channel has not yet got far enough through its start-up process, in this case the channel name will not be returned. Otherwise always.

XmitQName

Description: Transmission queue name.
Identifier: MQCACH_XMIT_Q_NAME.
Datatype: MQCFST.
Returned: For sender, server, cluster-sender and cluster-receiver channels only.

ConnectionName

Description: If the channel has successfully established a TCP connection, this is the Internet address. Otherwise it is the contents of the ConnectionName field in the channel definition.
Identifier: MQCACH_CONNECTION_NAME.
Datatype: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: The *ConnectionName* may not be available if the channel has not yet got far enough through its start-up process, in this case the connection name will not be returned. Otherwise always.

SSLHandshakeStage

Description: The name of the SSL function call giving the error, or on Windows, a description of the error is given in some cases. Details of these function names for specific platforms can be found as follows:

- For z/OS, see the *System Secure Sockets Layer Programming Guide and Reference*, SC24-5877.
- For other platforms, see *WebSphere MQ Messages*, GC34-6057.
- For Windows, when the text consists of a description of the error, no further information is available.

Identifier: MQCACH_SSL_HANDSHAKE_STAGE.
Datatype: MQCFST.
Maximum length: MQ_SSL_HANDSHAKE_STAGE_LENGTH.
Returned: This field is only present if *ReasonQualifier* is set to MQRQ_SSL_HANDSHAKE_ERROR.

Channel SSL Error

SSLReturnCode

Description: A numeric return code from a failing SSL call.

On Windows, if the SSLHandshakeStage contains a description of the error, the numeric return code will be an MQRC_* value.

Details of SSL Return Codes for specific platforms can be found as follows:

- For z/OS, see *WebSphere MQ for z/OS Messages and Codes*, GC34-6056.
- For other platforms, see *WebSphere MQ Messages*, GC34-6057.
- For Windows, if the return code is an MQRC_* value see the *WebSphere MQ Application Programming Reference*, SC34-6062.

Identifier: MQIACH_SSL_RETURN_CODE.
Datatype: MQCFIN.
Returned: This field is only present if *ReasonQualifier* is set to MQRQ_SSL_HANDSHAKE_ERROR.

SSLPeerName

Description: The Distinguished Name in the certificate sent from the remote system.
Identifier: MQCACH_SSL_PEER_NAME.
Datatype: MQCFST.
Maximum length: MQ_DISTINGUISHED_NAME_LENGTH.
Returned: This field is only present if *ReasonQualifier* is set to MQRQ_SSL_PEER_NAME_ERROR and is not always present for this reason qualifier.

Channel Started

Event name:	Channel Started.
Reason code in MQCFH:	MQRC_CHANNEL_STARTED (2282, X'8EA'). Channel started.
Event description:	Either an operator has issued a Start Channel command, or an instance of a channel has been successfully established. This condition is detected when Initial Data negotiation is complete and resynchronization has been performed where necessary, such that message transfer can proceed.
Event type:	Channel.
Platforms:	All, except WebSphere MQ for z/OS if CICS is used for distributed queue management. Client connections do not produce this event.
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT.

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

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

ChannelName

Description:	Channel name.
Identifier:	MQCACH_CHANNEL_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_CHANNEL_NAME_LENGTH.
Returned:	Always.

XmitQName

Description:	Transmission queue name.
Identifier:	MQCACH_XMIT_Q_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	For sender, server, cluster-sender, and cluster-receiver channels only.

ConnectionName

Description:	If the channel has successfully established a TCP connection, this is the Internet address. Otherwise it is the contents of the *ConnectionName* field in the channel definition.
Identifier:	MQCACH_CONNECTION_NAME.

Channel Started

Datatype:	MQCFST.
Maximum length:	MQ_CONN_NAME_LENGTH.
Returned:	Only for commands that do not contain a generic name.

Channel Stopped

Event name:	Channel Stopped.
Reason code in MQCFH:	MQRC_CHANNEL_STOPPED (2283, X'8EB'). Channel stopped.
Event description:	This is issued when a channel instance stops. It will only be issued if the channel instance previously issued a channel started event.
Event type:	Channel.
Platforms:	All, except WebSphere MQ for z/OS if CICS is used for distributed queue management. Client connections on WebSphere MQ for z/OS, or MQSeries Version 5 products do not produce this event.
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT.

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

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

ReasonQualifier

Description:	Identifier that qualifies the reason code.
Identifier:	MQIACF_REASON_QUALIFIER.
Datatype:	MQCFIN.
Values:	MQRQ_CHANNEL_STOPPED_OK Channel has been closed with either a zero return code or a warning return code. MQRQ_CHANNEL_STOPPED_ERROR Channel has been closed but there is an error reported and the channel is not in stopped or retry state. MQRQ_CHANNEL_STOPPED_RETRY Channel has been closed and it is in retry state. MQRQ_CHANNEL_STOPPED_DISABLED Channel has been closed and it is in a stopped state.
Returned:	Always.

ChannelName

Description:	Channel name.
Identifier:	MQCACH_CHANNEL_NAME.
Datatype:	MQCFST.

Maximum length: MQ_CHANNEL_NAME_LENGTH.
Returned: Always.

ErrorIdentifier

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

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

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

- For z/OS, see the section "Distributed queuing message codes" in the *WebSphere MQ for z/OS Messages and Codes* book.
- For other platforms, the last four digits of *ErrorIdentifier* when displayed in hexadecimal notation indicate the decimal code number of the error message.

  For example, if *ErrorIdentifier* has the value X'xxxxyyyy', the message code of the error message explaining the error is AMQyyyy. See the *WebSphere MQ Messages* book for a description of these error messages.

Identifier: MQIACF_ERROR_IDENTIFIER.
Datatype: MQCFIN.
Returned: Always.

AuxErrorDataInt1

Description: First integer of auxiliary error data for channel errors. If a channel is stopped due to an error, this is the first integer parameter that qualifies the error. This information is for use by IBM service personnel; include it in any problem report that you submit to IBM regarding this event message.
Identifier: MQIACF_AUX_ERROR_DATA_INT_1.
Datatype: MQCFIN.
Returned: Always.

AuxErrorDataInt2

Description: Second integer of auxiliary error data for channel errors. If a channel is stopped due to an error, this is the second integer parameter that qualifies the error. This information is for use by IBM service personnel; include it in any problem report that you submit to IBM regarding this event message.
Identifier: MQIACF_AUX_ERROR_DATA_INT_2.
Datatype: MQCFIN.
Returned: Always.

AuxErrorDataStr1

Description: First string of auxiliary error data for channel errors. If a channel is stopped due to an error, this is the first string parameter that qualifies the error. This information is for use by IBM service personnel; include it in any problem report that you submit to IBM regarding this event message.
Identifier: MQCACF_AUX_ERROR_DATA_STR_1.
Datatype: MQCFST.
Returned: Always.

AuxErrorDataStr2

Description: Second string of auxiliary error data for channel errors. If a channel is stopped due to an error, this is the second string parameter that qualifies the error. This information is for use by IBM service personnel; include it in any problem report that you submit to IBM regarding this event message.
Identifier: MQCACF_AUX_ERROR_DATA_STR_2.
Datatype: MQCFST.
Returned: Always.

AuxErrorDataStr3

Description: Third string of auxiliary error data for channel errors. If a channel is stopped due to an error, this is the third string parameter that qualifies the error. This information is for use by IBM service personnel; include it in any problem report that you submit to IBM regarding this event message.
Identifier: MQCACF_AUX_ERROR_DATA_STR_3.
Datatype: MQCFST.
Returned: Always.

XmitQName

Description: Transmission queue name.
Identifier: MQCACH_XMIT_Q_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: For sender, server, cluster-sender, and cluster-receiver channels only.

ConnectionName

Description: If the channel has successfully established a TCP connection, this is the Internet address. Otherwise it is the contents of the *ConnectionName* field in the channel definition.
Identifier: MQCACH_CONNECTION_NAME.
Datatype: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: Only for commands that do not contain a generic name.

Channel Stopped By User

Event name:	Channel Stopped By User.
Reason code in MQCFH:	MQRC_CHANNEL_STOPPED_BY_USER (2279, X'8E7'). Channel stopped by user.
Event description:	This is issued when a user issues a STOP CHL command. *ReasonQualifier* identifies the reasons for stopping.
Event type:	Channel.
Platforms:	All, except MQSeries for Compaq NonStop Kernel, MQSeries for Compaq Tru64 UNIX, or WebSphere MQ for z/OS if CICS is used for distributed queue management.
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

ReasonQualifier

Description: Identifier that qualifies the reason code.
Identifier: MQIACF_REASON_QUALIFIER.
Datatype: MQCFIN.
Values:

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

Returned: Always.

ChannelName

Description: Channel name.
Identifier: MQCACH_CHANNEL_NAME.
Datatype: MQCFST.
Maximum length: MQ_CHANNEL_NAME_LENGTH.
Returned: Always.

ErrorIdentifier

Description: Identifier of the cause of the error. As the event message is generated by a Stop Channel command and not a channel error, the following fields are set:

1. *ReasonQualifier*, containing the same value as in the *ReasonQualifier*(MQCFIN) field.
2. *AuxErrorDataInt1*, containing zeros
3. *AuxErrorDataInt2*, containing zeros
4. *AuxErrorDataStr1*, containing zeros
5. *AuxErrorDataStr2*, containing zeros
6. *AuxErrorDataStr3*, containing zeros

Identifier: MQIACF_ERROR_IDENTIFIER.
Datatype: MQCFIN.

Returned: Always.

XmitQName

Description: Transmission queue name.
Identifier: MQCACH_XMIT_Q_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: For sender, server, cluster-sender, and cluster-receiver channels only.

ConnectionName

Description: If the channel has successfully established a TCP connection, this is the Internet address. Otherwise it is the contents of the *ConnectionName* field in the channel definition.
Identifier: MQCACH_CONNECTION_NAME.
Datatype: MQCFST.
Maximum length: MQ_CONN_NAME_LENGTH.
Returned: Only for commands that do not contain a generic name.

Create object (z/OS only)

Event name:	Create object.
Reason code in MQCFH:	MQRC_CONFIG_CREATE_OBJECT (2367, X'93F'). New object created.
Event description:	A DEFINE or DEFINE REPLACE command was issued which successfully created a new object.
Event type:	Configuration.
Platforms:	WebSphere MQ for z/OS.
Event queue:	SYSTEM.ADMIN.CONFIG.EVENT.

Event data

EventUserId

Description: The user id that issued the command or call that generated the event. (This is the same user id that is used to check the authority to issue the command or call; for commands received from a queue, this is also the user identifier (UserIdentifier) from the MD of the command message).
Identifier: MQCACF_EVENT_USER_ID.
Datatype: MQCFST.
Maximum length: MQ_USER_ID_LENGTH.
Returned: Always.

EventOrigin

Description: The origin of the action causing the event.
Identifier: MQIACF_EVENT_ORIGIN.
Datatype: MQCFIN.
Values:

MQEVO_CONSOLE
: Console.

MQEVO_INIT
: Initialization input data set.

MQEVO_MSG
: Message on SYSTEM.COMMAND.INPUT.

MQEVO_MQSET
: MQSET call.

MQEVO_INTERNAL
: Directly by queue manager.

MQEVO_OTHER
: None of the above.

Returned: Always.

EventQMgr

Description: The queue manager where the command or call was entered. (The queue manager where the command is executed and that generates the event is in the MD of the event message).
Identifier: MQCACF_EVENT_Q_MGR.
Datatype: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.

Returned: Always.

EventAccountingToken

Description: For commands received as a message (MQEVO_MSG), the accounting token (AccountingToken) from the MD of the command message.
Identifier: MQBACF_EVENT_ACCOUNTING_TOKEN.
Datatype: MQCFBS.
Maximum length: MQ_ACCOUNTING_TOKEN_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplIdentity

Description: For commands received as a message (MQEVO_MSG), application identity data (ApplIdentityData) from the MD of the command message.
Identifier: MQCACF_EVENT_APPL_IDENTITY.
Datatype: MQCFST.
Maximum length: MQ_APPL_IDENTITY_DATA_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplType

Description: For commands received as a message (MQEVO_MSG), the type of application (PutApplType) from the MD of the command message.
Identifier: MQIACF_EVENT_APPL_TYPE.
Datatype: MQCFIN.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplName

Description: For commands received as a message (MQEVO_MSG), the name of the application (PutApplName) from the MD of the command message.
Identifier: MQCACF_EVENT_APPL_NAME.
Datatype: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplOrigin

Description: For commands received as a message (MQEVO_MSG), the application origin data (ApplOriginData) from the MD of the command message.
Identifier: MQCACF_EVENT_APPL_ORIGIN.
Datatype: MQCFST.
Maximum length: MQ_APPL_ORIGIN_DATA_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

ObjectType

Description: Object type:
Identifier: MQIACF_OBJECT_TYPE.
Datatype: MQCFIN.

Create object

Values:

MQOT_CHANNEL
: Channel.

MQOT_NAMELIST
: Namelist.

MQOT_PROCESS
: Process.

MQOT_Q
: Queue.

MQOT_STORAGE_CLASS
: Storage class.

MQOT_Q_MGR
: Queue manager.

MQOT_AUTH_INFO
: Authentication information.

MQOT_CF_STRUC
: CF structure.

Returned: Always.

ObjectName

Description: Object name:

Identifier : Identifier will be according to object type.

- MQCACH_CHANNEL_NAME
- MQCA_NAMELIST_NAME
- MQCA_PROCESS_NAME
- MQCA_Q_NAME
- MQCA_STORAGE_CLASS
- MQCA_Q_MGR_NAME
- MQCA_AUTH_INFO_NAME
- MQCA_CF_STRUC_NAME

Datatype: MQCFST.

Maximum length: MQ_OBJECT_NAME_LENGTH.

Returned: Always

Disposition

Description: Object disposition:

Identifier: MQIA_QSG_DISP.

Datatype: MQCFIN.

Values:

MQQSGD_Q_MGR
: Object resides on page set of queue manager.

MQQSGD_SHARED
: Object resides in shared repository and messages are shared in coupling facility.

MQQSGD_GROUP
: Object resides in shared repository.

MQQSGD_COPY
: Object resides on page set of queue manager and is a local copy of a GROUP object.

Returned: Always, except for queue manager and CF structure objects.

Object attributes

A parameter structure is returned for each attribute of the object. The attributes returned depend on the object type. For more information see Appendix D, "Event data for object attributes", on page 167

Default Transmission Queue Type Error

Event name:	Default Transmission Queue Type Error.
Reason code in MQCFH:	MQRC_DEF_XMIT_Q_TYPE_ERROR (2198, X'896'). Default transmission queue not local.
Event description:	An MQOPEN or MQPUT1 call was issued specifying a remote queue as the destination. Either a local definition of the remote queue was specified, or a queue-manager alias was being resolved, but in either case the *XmitQName* attribute in the local definition is blank. No transmission queue is defined with the same name as the destination queue manager, so the local queue manager has attempted to use the default transmission queue. However, although there is a queue defined by the *DefXmitQName* queue-manager attribute, it is not a local queue. See the *WebSphere MQ Application Programming Guide* for more information.
Event type:	Remote.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

QName

Description: Queue name from object descriptor (MQOD).
Identifier: MQCA_Q_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

XmitQName

Description: Default transmission queue name.
Identifier: MQCA_XMIT_Q_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

QType

Description: Type of default transmission queue.
Identifier: MQIA_Q_TYPE.
Datatype: MQCFIN.

Values:
- MQQT_ALIAS
  Alias queue definition.
- MQQT_REMOTE
  Local definition of a remote queue.

Returned: Always.

ApplType

Description: Type of application making the MQI call that caused the event.
Identifier: MQIA_APPL_TYPE.
Datatype: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application making the MQI call that caused the event.
Identifier: MQCACF_APPL_NAME.
Datatype: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

ObjectQMgrName

Description: Name of the object queue manager.
Identifier: MQCACF_OBJECT_Q_MGR_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: If the *ObjectName* in the object descriptor (MQOD), when the object was opened, is not the queue manager currently connected.

Note: If the application is a server for clients, the *ApplType* and *ApplName* parameters identify the server, not the client.

Default Transmission Queue Usage Error

Event name:	Default Transmission Queue Usage Error.
Reason code in MQCFH:	MQRC_DEF_XMIT_Q_USAGE_ERROR (2199, X'897'). Default transmission queue usage error.
Event description:	An MQOPEN or MQPUT1 call was issued specifying a remote queue as the destination. Either a local definition of the remote queue was specified, or a queue-manager alias was being resolved, but in either case the *XmitQName* attribute in the local definition is blank. No transmission queue is defined with the same name as the destination queue manager, so the local queue manager has attempted to use the default transmission queue. However, the queue defined by the *DefXmitQName* queue-manager attribute does not have a *Usage* attribute of MQUS_TRANSMISSION. See the *WebSphere MQ Application Programming Guide* for more information.
Event type:	Remote.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

QName

Description: Queue name from object descriptor (MQOD).
Identifier: MQCA_Q_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

XmitQName

Description: Default transmission queue name.
Identifier: MQCA_XMIT_Q_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

ApplType

Description: Type of application making the MQI call that caused the event.
Identifier: MQIA_APPL_TYPE.
Datatype: MQCFIN.
Returned: Always.

ApplName

Description:	Name of the application making the MQI call that caused the event.
Identifier:	MQCACF_APPL_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_APPL_NAME_LENGTH.
Returned:	Always.

ObjectQMgrName

Description:	Name of the object queue manager.
Identifier:	MQCACF_OBJECT_Q_MGR_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	If the *ObjectName* in the object descriptor (MQOD), when the object was opened, is not the queue manager currently connected.

Note: If the application is a server for clients, the *ApplType* and *ApplName* parameters identify the server, not the client.

Delete object (z/OS only)

Event name:	Delete object.
Reason code in MQCFH:	MQRC_CONFIG_DELETE_OBJECT (2369, X'941'). Object deleted.
Event description:	A DELETE command or MQCLOSE call was issued that successfully deleted an object.
Event type:	Configuration.
Platforms:	WebSphere MQ for z/OS.
Event queue:	SYSTEM.ADMIN.CONFIG.EVENT.

Event data

EventUserId

Description: The user id that issued the command or call that generated the event. (This is the same user id that is used to check the authority to issue the command or call; for commands received from a queue, this is also the user identifier (UserIdentifier) from the MD of the command message).
Identifier: MQCACF_EVENT_USER_ID.
Datatype: MQCFST.
Maximum length: MQ_USER_ID_LENGTH.
Returned: Always.

EventOrigin

Description: The origin of the action causing the event.
Identifier: MQIACF_EVENT_ORIGIN.
Datatype: MQCFIN.
Values:

MQEVO_CONSOLE
: Console.

MQEVO_INIT
: Initialization input data set.

MQEVO_MSG
: Message on SYSTEM.COMMAND.INPUT.

MQEVO_MQSET
: MQSET call.

MQEVO_INTERNAL
: Directly by queue manager.

MQEVO_OTHER
: None of the above.

Returned: Always.

EventQMgr

Description: The queue manager where the command or call was entered. (The queue manager where the command is executed and that generates the event is in the MD of the event message).
Identifier: MQCACF_EVENT_Q_MGR.
Datatype: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.

Returned: Always.

EventAccountingToken

Description: For commands received as a message (MQEVO_MSG), the accounting token (AccountingToken) from the MD of the command message.
Identifier: MQBACF_EVENT_ACCOUNTING_TOKEN.
Datatype: MQCFBS.
Maximum length: MQ_ACCOUNTING_TOKEN_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplIdentity

Description: For commands received as a message (MQEVO_MSG), application identity data (ApplIdentityData) from the MD of the command message.
Identifier: MQCACF_EVENT_APPL_IDENTITY.
Datatype: MQCFST.
Maximum length: MQ_APPL_IDENTITY_DATA_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplType

Description: For commands received as a message (MQEVO_MSG), the type of application (PutApplType) from the MD of the command message.
Identifier: MQIACF_EVENT_APPL_TYPE.
Datatype: MQCFIN.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplName

Description: For commands received as a message (MQEVO_MSG), the name of the application (PutApplName) from the MD of the command message.
Identifier: MQCACF_EVENT_APPL_NAME.
Datatype: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplOrigin

Description: For commands received as a message (MQEVO_MSG), the application origin data (ApplOriginData) from the MD of the command message.
Identifier: MQCACF_EVENT_APPL_ORIGIN.
Datatype: MQCFST.
Maximum length: MQ_APPL_ORIGIN_DATA_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

ObjectType

Description: Object type:
Identifier: MQIACF_OBJECT_TYPE.
Datatype: MQCFIN.

Delete object

Values:

MQOT_CHANNEL
: Channel.

MQOT_NAMELIST
: Namelist.

MQOT_PROCESS
: Process.

MQOT_Q
: Queue.

MQOT_STORAGE_CLASS
: Storage class.

MQOT_Q_MGR
: Queue manager.

MQOT_AUTH_INFO
: Authentication information.

MQOT_CF_STRUC
: CF structure.

Returned: Always.

ObjectName

Description: Object name:

Identifier : Identifier will be according to object type.

- MQCACH_CHANNEL_NAME
- MQCA_NAMELIST_NAME
- MQCA_PROCESS_NAME
- MQCA_Q_NAME
- MQCA_STORAGE_CLASS
- MQCA_Q_MGR_NAME
- MQCA_AUTH_INFO_NAME
- MQCA_CF_STRUC_NAME

Datatype: MQCFST.

Maximum length: MQ_OBJECT_NAME_LENGTH.

Returned: Always

Disposition

Description: Object disposition:

Identifier: MQIA_QSG_DISP.

Datatype: MQCFIN.

Values:

MQQSGD_Q_MGR
: Object resides on page set of queue manager.

MQQSGD_SHARED
: Object resides in shared repository and messages are shared in coupling facility.

MQQSGD_GROUP
: Object resides in shared repository.

MQQSGD_COPY
: Object resides on page set of queue manager and is a local copy of a GROUP object.

Returned: Always, except for queue manager and CF structure objects.

Object attributes

A parameter structure is returned for each attribute of the object. The attributes returned depend on the object type. For more information see Appendix D, “Event data for object attributes”, on page 167

Get Inhibited

Event name:	Get Inhibited.
Reason code in MQCFH:	MQRC_GET_INHIBITED (2016, X'7E0'). Gets inhibited for the queue.
Event description:	MQGET calls are currently inhibited for the queue (see the *InhibitGet* queue attribute in the *WebSphere MQ Application Programming Reference* manual) or for the queue to which this queue resolves.
Event type:	Inhibit.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

QName

Description:	Queue name from object descriptor (MQOD).
Identifier:	MQCA_Q_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

ApplType

Description:	Type of application that issued the get.
Identifier:	MQIA_APPL_TYPE.
Datatype:	MQCFIN.
Returned:	Always.

ApplName

Description:	Name of the application that issued the get.
Identifier:	MQCACF_APPL_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_APPL_NAME_LENGTH.
Returned:	Always.

Note: If the application is a server for clients, the *ApplType* and *ApplName* parameters identify the server, not the client.

Not Authorized (type 1)

Event name:	Not Authorized (type 1).
Reason code in MQCFH:	MQRC_NOT_AUTHORIZED (2035, X'7F3'). Not authorized for access.
Event description:	On an MQCONN call, the user is not authorized to connect to the queue manager.
Event type:	Authority.
Platforms:	All, except WebSphere MQ for z/OS , MQSeries for OS/2 Warp, and MQSeries for Windows Version 2.1.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

ReasonQualifier

Description: Identifier for type 1 authority events.
Identifier: MQIACF_REASON_QUALIFIER.
Datatype: MQCFIN.
Values:
MQRQ_CONN_NOT_AUTHORIZED
Connection not authorized.
Returned: Always.

UserIdentifier

Description: User identifier that caused the authorization check.
Identifier: MQCACF_USER_IDENTIFIER.
Datatype: MQCFST.
Maximum length: MQ_USER_ID_LENGTH.
Returned: Always.

ApplType

Description: Type of application causing the event.
Identifier: MQIA_APPL_TYPE.
Datatype: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application causing the event.
Identifier: MQCACF_APPL_NAME.
Datatype: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

Not Authorized (type 2)

Event name:	Not Authorized (type 2).
Reason code in MQCFH:	MQRC_NOT_AUTHORIZED (2035, X'7F3'). Not authorized for access.
Event description:	On an MQOPEN or MQPUT1 call, the user is not authorized to open the object for the options specified.
Event type:	Authority.
Platforms:	All, except WebSphere MQ for z/OS, MQSeries for OS/2 Warp, MQSeries for Compaq NonStop Kernel, and MQSeries for Windows Version 2.1.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

ReasonQualifier

Description: Identifier for type 2 authority events.
Identifier: MQIACF_REASON_QUALIFIER.
Datatype: MQCFIN.
Values:
MQRQ_OPEN_NOT_AUTHORIZED
Open not authorized.
Returned: Always.

Options

Description: Options specified on the MQOPEN call.
Identifier: MQIACF_OPEN_OPTIONS.
Datatype: MQCFIN.
Returned: Always.

UserIdentifier

Description: User identifier that caused the authorization check.
Identifier: MQCACF_USER_IDENTIFIER.
Datatype: MQCFST.
Maximum length: MQ_USER_ID_LENGTH.
Returned: Always.

ApplType

Description: Type of application that caused the authorization check.
Identifier: MQIA_APPL_TYPE.
Datatype: MQCFIN.
Returned: Always.

ApplName

Description:	Name of the application that caused the authorization check.
Identifier:	MQCACF_APPL_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_APPL_NAME_LENGTH.
Returned:	Always.

ObjectQMgrName

Description:	Object queue manager name from object descriptor (MQOD).
Identifier:	MQCACF_OBJECT_Q_MGR_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	If the *ObjectQMgrName* in the object descriptor (MQOD) when the object was opened is not the queue manager currently connected.

QName

Description:	Object name from object descriptor (MQOD).
Identifier:	MQCA_Q_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	If the object opened is not a process object.

ProcessName

Description:	Name of process object from object descriptor (MQOD).
Identifier:	MQCA_PROCESS_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_PROCESS_NAME_LENGTH.
Returned:	If the object opened is a process object.

Note: If the application is a server for clients, the *ApplType* and *ApplName* parameters identify the server, not the client.

Not Authorized (type 3)

Event name:	Not Authorized (type 3).
Reason code in MQCFH:	MQRC_NOT_AUTHORIZED (2035, X'7F3'). Not authorized for access.
Event description:	On an MQCLOSE call, the user is not authorized to delete the object, which is a permanent dynamic queue, and the *Hobj* parameter specified on the MQCLOSE call is not the handle returned by the MQOPEN call that created the queue.
Event type:	Authority.
Platforms:	All, except WebSphere MQ for z/OS, MQSeries for OS/2 Warp, MQSeries for Compaq NonStop Kernel, and MQSeries for Windows Version 2.1.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

ReasonQualifier

Description: Identifier for type 3 authority events.
Identifier: MQIACF_REASON_QUALIFIER.
Datatype: MQCFIN.
Values:
MQRQ_CLOSE_NOT_AUTHORIZED
Close not authorized.
Returned: Always.

QName

Description: Object name from object descriptor (MQOD).
Identifier: MQCA_Q_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

UserIdentifier

Description: User identifier that caused the authorization check.
Identifier: MQCACF_USER_IDENTIFIER.
Datatype: MQCFST.
Maximum length: MQ_USER_ID_LENGTH.
Returned: Always.

ApplType

Description: Type of application causing the authorization check.
Identifier: MQIA_APPL_TYPE.
Datatype: MQCFIN.

Returned: Always.

ApplName

Description: Name of the application causing the authorization check.
Identifier: MQCACF_APPL_NAME.
Datatype: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

Note: If the application is a server for clients, the *ApplType* and *ApplName* parameters identify the server, not the client.

Not Authorized (type 4)

Event name:	Not Authorized (type 4).
Reason code in MQCFH:	MQRC_NOT_AUTHORIZED (2035, X'7F3'). Not authorized for access.
Event description:	Indicates that a command has been issued from a user ID that is not authorized to access the object specified in the command.
Event type:	Authority.
Platforms:	All, except WebSphere MQ for z/OS, MQSeries for OS/2 Warp, MQSeries for Compaq NonStop Kernel, and MQSeries for Windows Version 2.1.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

ReasonQualifier

Description: Identifier for type 4 authority events.
Identifier: MQIACF_REASON_QUALIFIER.
Datatype: MQCFIN.
Values:
MQRQ_CMD_NOT_AUTHORIZED
Command not authorized.
Returned: Always.

Command

Description: Command identifier. See the MQCFH header structure, described in "MQCFH (Event header)" on page 44.
Identifier: MQIACF_COMMAND.
Datatype: MQCFIN.
Returned: Always.

UserIdentifier

Description: User identifier that caused the authorization check.
Identifier: MQCACF_USER_IDENTIFIER.
Datatype: MQCFST.
Maximum length: MQ_USER_ID_LENGTH.
Returned: Always.

Put Inhibited

Event name:	Put Inhibited.
Reason code in MQCFH:	MQRC_PUT_INHIBITED (2051, X'803'). Put calls inhibited for the queue.
Event description:	MQPUT and MQPUT1 calls are currently inhibited for the queue (see the *InhibitPut* queue attribute in in the *WebSphere MQ Application Programming Reference* manual) or for the queue to which this queue resolves.
Event type:	Inhibit.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

QName

Description:	Queue name from object descriptor (MQOD).
Identifier:	MQCA_Q_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

ApplType

Description:	Type of application that issued the put.
Identifier:	MQIA_APPL_TYPE.
Datatype:	MQCFIN.
Returned:	Always.

ApplName

Description:	Name of the application that issued the put.
Identifier:	MQCACF_APPL_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_APPL_NAME_LENGTH.
Returned:	Always.

ObjectQMgrName

Description:	Name of queue manager from object descriptor (MQOD).
Identifier:	MQCACF_OBJECT_Q_MGR_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.

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

Note: If the application is a server for clients, the *ApplType* and *ApplName* parameters identify the server, not the client.

Queue Depth High

Event name:	Queue Depth High.
Reason code in MQCFH:	MQRC_Q_DEPTH_HIGH (2224, X'8B0'). Queue depth high limit reached or exceeded.
Event description:	An MQPUT or MQPUT1 call has caused the queue depth to be incremented to or above the limit specified in the *QDepthHighLimit* attribute.
Corrective action:	None. This reason code is used only to identify the corresponding event message.
Event type:	Performance.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.PERFM.EVENT.

Notes:

1. WebSphere MQ for z/OS supports queue depth events on shared queues. You might receive a NULL event message for a shared queue if a queue manager has performed no activity on that shared queue.
2. For shared queues, the correlation identifier, *CorrelId* in the message descriptor (MQMD) is set. See "MQMD (message descriptor)" on page 40 for more information.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

QName

Description:	Name of the queue on which the limit has been reached.
Identifier:	MQCA_BASE_Q_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

TimeSinceReset

Description:	Time, in seconds, since the statistics were last reset. The value recorded by this timer is also used as the *interval time* in queue service interval events.
Identifier:	MQIA_TIME_SINCE_RESET.
Datatype:	MQCFIN.
Returned:	Always.

HighQDepth

Description:	Maximum number of messages on the queue since the queue statistics were last reset.
Identifier:	MQIA_HIGH_Q_DEPTH.
Datatype:	MQCFIN.

Returned: Always.

MsgEnqCount

Description: Number of messages enqueued. This is the number of messages put on the queue since the queue statistics were last reset.
Identifier: MQIA_MSG_ENQ_COUNT.
Datatype: MQCFIN.
Returned: Always.

MsgDeqCount

Description: Number of messages removed from the queue since the queue statistics were last reset.
Identifier: MQIA_MSG_DEQ_COUNT.
Datatype: MQCFIN.
Returned: Always.

Queue Depth Low

Event name:	Queue Depth Low.
Reason code in MQCFH:	MQRC_Q_DEPTH_LOW (2225, X'8B1'). Queue depth low limit reached or exceeded.
Event description:	An MQGET call has caused the queue depth to be decremented to or below the limit specified in the *QDepthLowLimit* attribute.
Corrective action:	None. This reason code is used only to identify the corresponding event message.
Event type:	Performance.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.PERFM.EVENT.

Notes:

1. WebSphere MQ for z/OS supports queue depth events on shared queues. You might receive a NULL event message for a shared queue if a queue manager has performed no activity on that shared queue.
2. For shared queues, the correlation identifier, *CorrelId* in the message descriptor (MQMD) is set. See "MQMD (message descriptor)" on page 40 for more information.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

QName

Description:	Name of the queue on which the limit has been reached.
Identifier:	MQCA_BASE_Q_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

TimeSinceReset

Description:	Time, in seconds, since the statistics were last reset. The value recorded by this timer is also used as the *interval time* in queue service interval events.
Identifier:	MQIA_TIME_SINCE_RESET.
Datatype:	MQCFIN.
Returned:	Always.

HighQDepth

Description:	Maximum number of messages on the queue since the queue statistics were last reset.
Identifier:	MQIA_HIGH_Q_DEPTH.
Datatype:	MQCFIN.

Returned: Always.

MsgEnqCount

Description: Number of messages enqueued. This is the number of messages put on the queue since the queue statistics were last reset.
Identifier: MQIA_MSG_ENQ_COUNT.
Datatype: MQCFIN.
Returned: Always.

MsgDeqCount

Description: Number of messages removed from the queue since the queue statistics were last reset.
Identifier: MQIA_MSG_DEQ_COUNT.
Datatype: MQCFIN.
Returned: Always.

Queue Full

Event name:	Queue Full.
Reason code in MQCFH:	MQRC_Q_FULL (2053, X'805'). Queue already contains maximum number of messages.
Event description:	On an MQPUT or MQPUT1 call, the call failed because the queue is full. That is, it already contains the maximum number of messages possible (see the *MaxQDepth* local-queue attribute in the *WebSphere MQ Application Programming Reference* manual). This reason code can also occur in the *Feedback* field in the message descriptor of a report message; in this case it indicates that the error was encountered by a message channel agent when it attempted to put the message on a remote queue.
Corrective action:	Retry the operation later. Consider increasing the maximum depth for this queue, or arranging for more instances of the application to service the queue.
Event type:	Performance.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.PERFM.EVENT.

Notes:

1. WebSphere MQ for z/OS supports queue depth events on shared queues. You might receive a NULL event message for a shared queue if a queue manager has performed no activity on that shared queue.
2. For shared queues, the correlation identifier, *CorrelId* in the message descriptor (MQMD) is set. See "MQMD (message descriptor)" on page 40 for more information.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

QName

Description:	Name of the queue on which the put was rejected.
Identifier:	MQCA_BASE_Q_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

TimeSinceReset

Description:	Time, in seconds, since the statistics were last reset.
Identifier:	MQIA_TIME_SINCE_RESET.
Datatype:	MQCFIN.
Returned:	Always.

Queue Full

HighQDepth

Description: Maximum number of messages on a queue.
Identifier: MQIA_HIGH_Q_DEPTH.
Datatype: MQCFIN.
Returned: Always.

MsgEnqCount

Description: Number of messages enqueued. This is the number of messages put on the queue since the queue statistics were last reset.
Identifier: MQIA_MSG_ENQ_COUNT.
Datatype: MQCFIN.
Returned: Always.

MsgDeqCount

Description: Number of messages removed from the queue since the queue statistics were last reset.
Identifier: MQIA_MSG_DEQ_COUNT.
Datatype: MQCFIN.
Returned: Always.

Queue Manager Active

Event name:	Queue Manager Active.
Reason code in MQCFH:	MQRC_Q_MGR_ACTIVE (2222, X'8AE'). Queue manager created.
Event description:	This condition is detected when a queue manager becomes active.
Event type:	Start And Stop.
Platforms:	All, except the first start of a WebSphere MQ for z/OS queue manager. In this case it is produced only on subsequent restarts.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

Queue Manager Not Active

Event name:	Queue Manager Not Active.
Reason code in MQCFH:	MQRC_Q_MGR_NOT_ACTIVE (2223, X'8AF'). Queue manager unavailable.
Event description:	This condition is detected when a queue manager is requested to stop or quiesce.
Event type:	Start And Stop.
Platforms:	All, except WebSphere MQ for z/OS.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

ReasonQualifier

Description: Identifier of causes of this reason code. This specifies the type of stop that was requested.
Identifier: MQIACF_REASON_QUALIFIER.
Datatype: MQCFIN.
Values:

MQRQ_Q_MGR_STOPPING
: Queue manager stopping.

MQRQ_Q_MGR_QUIESCING
: Queue manager quiescing.

Returned: Always.

Queue Service Interval High

Event name:	Queue Service Interval High.
Reason code in MQCFH:	MQRC_Q_SERVICE_INTERVAL_HIGH (2226, X'8B2'). Queue service interval high.
Event description:	No successful gets or puts have been detected within an interval greater than the limit specified in the *QServiceInterval* attribute.
Corrective action:	None. This reason code is used only to identify the corresponding event message.
Event type:	Performance.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.PERFM.EVENT.

Note: WebSphere MQ for z/OS does not support service interval events on shared queues.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

QName

Description:	Name of the queue specified on the command that caused this queue service interval event to be generated.
Identifier:	MQCA_BASE_Q_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

TimeSinceReset

Description:	Time, in seconds, since the statistics were last reset. For a service interval high event, this value is greater than the service interval.
Identifier:	MQIA_TIME_SINCE_RESET.
Datatype:	MQCFIN.
Returned:	Always.

HighQDepth

Description:	Maximum number of messages on the queue since the queue statistics were last reset.
Identifier:	MQIA_HIGH_Q_DEPTH.
Datatype:	MQCFIN.
Returned:	Always.

Queue Service Interval High

MsgEnqCount

Description: Number of messages enqueued. This is the number of messages put on the queue since the queue statistics were last reset.
Identifier: MQIA_MSG_ENQ_COUNT.
Datatype: MQCFIN.
Returned: Always.

MsgDeqCount

Description: Number of messages removed from the queue since the queue statistics were last reset.
Identifier: MQIA_MSG_DEQ_COUNT.
Datatype: MQCFIN.
Returned: Always.

Queue Service Interval OK

Event name:	Queue Service Interval OK.
Reason code in MQCFH:	MQRC_Q_SERVICE_INTERVAL_OK (2227, X'8B3'). Queue service interval OK.
Event description:	A successful get has been detected within an interval less than or equal to the limit specified in the *QServiceInterval* attribute.
Corrective action:	None. This reason code is used only to identify the corresponding event message.
Event type:	Performance.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.PERFM.EVENT.

Note: WebSphere MQ for z/OS does not support service interval events on shared queues.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

QName

Description:	Queue name specified on the command that caused this queue service interval event to be generated.
Identifier:	MQCA_BASE_Q_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

TimeSinceReset

Description:	Time, in seconds, since the statistics were last reset.
Identifier:	MQIA_TIME_SINCE_RESET.
Datatype:	MQCFIN.
Returned:	Always.

HighQDepth

Description:	Maximum number of messages on the queue since the queue statistics were last reset.
Identifier:	MQIA_HIGH_Q_DEPTH.
Datatype:	MQCFIN.
Returned:	Always.

MsgEnqCount

Description:	Number of messages enqueued. This is the number of messages put on the queue since the queue statistics were last reset.

Queue Service Interval OK

Identifier: MQIA_MSG_ENQ_COUNT.
Datatype: MQCFIN.
Returned: Always.

MsgDeqCount

Description: Number of messages removed from the queue since the queue statistics were last reset.
Identifier: MQIA_MSG_DEQ_COUNT.
Datatype: MQCFIN.
Returned: Always.

Queue Type Error

Event name:	Queue Type Error.
Reason code in MQCFH:	MQRC_Q_TYPE_ERROR (2057, X'809'). Queue type not valid.
Event description:	On an MQOPEN call, the *ObjectQMgrName* field in the object descriptor specifies the name of a local definition of a remote queue (in order to specify a queue-manager alias). In that local definition the *RemoteQMgrName* attribute is the name of the local queue manager. However, the *ObjectName* field specifies the name of a model queue on the local queue manager, which is not allowed. See the *WebSphere MQ Application Programming Guide* for more information.
Event type:	Remote.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

QName

Description: Queue name from object descriptor (MQOD).
Identifier: MQCA_Q_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

ApplType

Description: Type of application making the MQI call that caused the event.
Identifier: MQIA_APPL_TYPE.
Datatype: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application making the MQI call that caused the event.
Identifier: MQCACF_APPL_NAME.
Datatype: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

ObjectQMgrName

Description: Name of the object queue manager.
Identifier: MQCACF_OBJECT_Q_MGR_NAME.
Datatype: MQCFST.

Queue Type Error

Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

Note: If the application is a server for clients, the *ApplType* and *ApplName* parameters identify the server, not the client.

Refresh object (z/OS only)

Event name:	Refresh object.
Reason code in MQCFH:	MQRC_CONFIG_REFRESH_OBJECT (2370, X'942'). Refresh queue manager configuration.
Event description:	A REFRESH QMGR command specifying TYPE (CONFIGEV) was issued.
Event type:	Configuration.
Platforms:	WebSphere MQ for z/OS.
Event queue:	SYSTEM.ADMIN.CONFIG.EVENT.

Note: The REFRESH QMGR command can produce many configuration events; one event is generated for each object that is selected by the command.

Event data

EventUserId

Description: The user id that issued the command or call that generated the event. (This is the same user id that is used to check the authority to issue the command or call; for commands received from a queue, this is also the user identifier (UserIdentifier) from the MD of the command message).
Identifier: MQCACF_EVENT_USER_ID.
Datatype: MQCFST.
Maximum length: MQ_USER_ID_LENGTH.
Returned: Always.

EventOrigin

Description: The origin of the action causing the event.
Identifier: MQIACF_EVENT_ORIGIN.
Datatype: MQCFIN.
Values:

MQEVO_CONSOLE
: Console.

MQEVO_INIT
: Initialization input data set.

MQEVO_MSG
: Message on SYSTEM.COMMAND.INPUT.

MQEVO_MQSET
: MQSET call.

MQEVO_INTERNAL
: Directly by queue manager.

MQEVO_OTHER
: None of the above.

Returned: Always.

EventQMgr

Description: The queue manager where the command or call was entered. (The queue manager where the command is executed and that generates the event is in the MD of the event message).
Identifier: MQCACF_EVENT_Q_MGR.

Refresh object

Datatype: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

EventAccountingToken

Description: For commands received as a message (MQEVO_MSG), the accounting token (AccountingToken) from the MD of the command message.
Identifier: MQBACF_EVENT_ACCOUNTING_TOKEN.
Datatype: MQCFBS.
Maximum length: MQ_ACCOUNTING_TOKEN_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplIdentity

Description: For commands received as a message (MQEVO_MSG), application identity data (ApplIdentityData) from the MD of the command message.
Identifier: MQCACF_EVENT_APPL_IDENTITY.
Datatype: MQCFST.
Maximum length: MQ_APPL_IDENTITY_DATA_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplType

Description: For commands received as a message (MQEVO_MSG), the type of application (PutApplType) from the MD of the command message.
Identifier: MQIACF_EVENT_APPL_TYPE.
Datatype: MQCFIN.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplName

Description: For commands received as a message (MQEVO_MSG), the name of the application (PutApplName) from the MD of the command message.
Identifier: MQCACF_EVENT_APPL_NAME.
Datatype: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

EventApplOrigin

Description: For commands received as a message (MQEVO_MSG), the application origin data (ApplOriginData) from the MD of the command message.
Identifier: MQCACF_EVENT_APPL_ORIGIN.
Datatype: MQCFST.
Maximum length: MQ_APPL_ORIGIN_DATA_LENGTH.
Returned: Only if EventOrigin is MQEVO_MSG.

ObjectType

Description: Object type:
Identifier: MQIACF_OBJECT_TYPE.
Datatype: MQCFIN.

Values:

MQOT_CHANNEL
: Channel.

MQOT_NAMELIST
: Namelist.

MQOT_PROCESS
: Process.

MQOT_Q
: Queue.

MQOT_STORAGE_CLASS
: Storage class.

MQOT_Q_MGR
: Queue manager.

MQOT_AUTH_INFO
: Authentication information.

MQOT_CF_STRUC
: CF structure.

Returned: Always.

ObjectName

Description: Object name:

Identifier : Identifier will be according to object type.

- MQCACH_CHANNEL_NAME
- MQCA_NAMELIST_NAME
- MQCA_PROCESS_NAME
- MQCA_Q_NAME
- MQCA_STORAGE_CLASS
- MQCA_Q_MGR_NAME
- MQCA_AUTH_INFO_NAME
- MQCA_CF_STRUC_NAME

Datatype: MQCFST.

Maximum length: MQ_OBJECT_NAME_LENGTH.

Returned: Always

Disposition

Description: Object disposition:

Identifier: MQIA_QSG_DISP.

Datatype: MQCFIN.

Values:

MQQSGD_Q_MGR
: Object resides on page set of queue manager.

MQQSGD_SHARED
: Object resides in shared repository and messages are shared in coupling facility.

MQQSGD_GROUP
: Object resides in shared repository.

MQQSGD_COPY
: Object resides on page set of queue manager and is a local copy of a GROUP object.

Returned: Always, except for queue manager and CF structure objects.

Object attributes

A parameter structure is returned for each attribute of the object. The attributes returned depend on the object type. For more information see Appendix D, “Event data for object attributes”, on page 167

Remote Queue Name Error

Event name:	Remote Queue Name Error.
Reason code in MQCFH:	MQRC_REMOTE_Q_NAME_ERROR (2184, X'888'). Remote queue name not valid.
Event description:	On an MQOPEN or MQPUT1 call either: • A local definition of a remote queue (or an alias to one) was specified, but the *RemoteQName* attribute in the remote queue definition is blank. Note that this error occurs even if the *XmitQName* in the definition is not blank. or • The *ObjectQMgrName* field in the object descriptor is not blank and not the name of the local queue manager, but the *ObjectName* field is blank.
Event type:	Remote.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

QName

Description: Queue name from object descriptor (MQOD).
Identifier: MQCA_Q_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

ApplType

Description: Type of application making the MQI call that caused the event.
Identifier: MQIA_APPL_TYPE.
Datatype: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application making the MQI call that caused the event.
Identifier: MQCACF_APPL_NAME.
Datatype: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

ObjectQMgrName

Description: Name of the object queue manager.

Remote Queue Name Error

Identifier:	MQCACF_OBJECT_Q_MGR_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	If the *ObjectName* in the object descriptor (MQOD), when the object was opened, is not the queue manager currently connected.

Note: If the application is a server for clients the *ApplType* and *ApplName* parameters identify the server not the client.

Transmission Queue Type Error

Event name:	Transmission Queue Type Error.
Reason code in MQCFH:	MQRC_XMIT_Q_TYPE_ERROR (2091, X'82B'). Transmission queue not local.
Event description:	On an MQOPEN or MQPUT1 call, a message is to be sent to a remote queue manager. The *ObjectName* or *ObjectQMgrName* field in the object descriptor specifies the name of a local definition of a remote queue but one of the following applies to the *XmitQName* attribute of the definition. Either: • *XmitQName* is not blank, but specifies a queue that is not a local queue or • *XmitQName* is blank, but *RemoteQMgrName* specifies a queue that is not a local queue This also occurs if the queue name is resolved through a cell directory, and the remote queue manager name obtained from the cell directory is the name of a queue, but this is not a local queue.
Event type:	Remote.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

QName

Description:	Queue name from object descriptor (MQOD).
Identifier:	MQCA_Q_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

XmitQName

Description:	Transmission queue name.
Identifier:	MQCA_XMIT_Q_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

QType

Description:	Type of transmission queue.
Identifier:	MQIA_Q_TYPE.
Datatype:	MQCFIN.

Transmission Queue Type Error

Values:
- MQQT_ALIAS
  Alias queue definition.
- MQQT_REMOTE
  Local definition of a remote queue.

Returned: Always.

ApplType

Description: Type of application making the MQI call that caused the event.
Identifier: MQIA_APPL_TYPE.
Datatype: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application making the MQI call that caused the event.
Identifier: MQCACF_APPL_NAME.
Datatype: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

ObjectQMgrName

Description: Name of the object queue manager.
Identifier: MQCACF_OBJECT_Q_MGR_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: If the *ObjectName* in the object descriptor (MQOD), when the object was opened, is not the queue manager currently connected.

Note: If the application is a server for clients, the *ApplType* and *ApplName* parameters identify the server, not the client.

Transmission Queue Usage Error

Event name:	Transmission Queue Usage Error.
Reason code in MQCFH:	MQRC_XMIT_Q_USAGE_ERROR (2092, X'82C'). Transmission queue with wrong usage.
Event description:	On an MQOPEN or MQPUT1 call, a message is to be sent to a remote queue manager, but one of the following occurred. Either: • *ObjectQMgrName* specifies the name of a local queue, but it does not have a *Usage* attribute of MQUS_TRANSMISSION. • The *ObjectName* or *ObjectQMgrName* field in the object descriptor specifies the name of a local definition of a remote queue but one of the following applies to the *XmitQName* attribute of the definition: – *XmitQName* is not blank, but specifies a queue that does not have a *Usage* attribute of MQUS_TRANSMISSION – *XmitQName* is blank, but *RemoteQMgrName* specifies a queue that does not have a *Usage* attribute of MQUS_TRANSMISSION • The queue name is resolved through a cell directory, and the remote queue manager name obtained from the cell directory is the name of a local queue, but it does not have a *Usage* attribute of MQUS_TRANSMISSION.
Event type:	Remote.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

QName

Description:	Queue name from object descriptor (MQOD).
Identifier:	MQCA_Q_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

XmitQName

Description:	Transmission queue name.
Identifier:	MQCA_XMIT_Q_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

ApplType

Description:	Type of application making the MQI call that caused the event.
Identifier:	MQIA_APPL_TYPE.

Transmission Queue Usage Error

Datatype: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application making the MQI call that caused the event.
Identifier: MQCACF_APPL_NAME.
Datatype: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

ObjectQMgrName

Description: Name of the object queue manager.
Identifier: MQCACF_OBJECT_Q_MGR_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: If the *ObjectName* in the object descriptor (MQOD), when the object was opened, is not the queue manager currently connected.

Note: If the application is a server for clients, the *ApplType* and *ApplName* parameters identify the server, not the client.

Unknown Alias Base Queue

Event name:	Unknown Alias Base Queue.
Reason code in MQCFH:	MQRC_UNKOWN_ALIAS_BASE_Q (2082, X'822'). Unknown alias base queue.
Event description:	An MQOPEN or MQPUT1 call was issued specifying an alias queue as the destination, but the *BaseQName* in the alias queue attributes is not recognized as a queue name.
Event type:	Local.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

QName

Description: Queue name from object descriptor (MQOD).
Identifier: MQCA_Q_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

BaseQName

Description: Queue name to which the alias resolves.
Identifier: MQCA_BASE_Q_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

ApplType

Description: Type of application making the MQI call that caused the event.
Identifier: MQIA_APPL_TYPE.
Datatype: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application making the MQI call that caused the event.
Identifier: MQCACF_APPL_NAME.
Datatype: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

Unknown Alias Base Queue

ObjectQMgrName

Description:	Name of the object queue manager.
Identifier:	MQCACF_OBJECT_Q_MGR_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	If the *ObjectName* in the object descriptor (MQOD), when the object was opened, is not the queue manager currently connected.

Note: If the application is a server for clients, the *ApplType* and *ApplName* parameters identify the server, not the client.

Unknown Default Transmission Queue

Event name:	Unknown Default Transmission Queue.
Reason code in MQCFH:	MQRC_UNKNOWN_DEF_XMIT_Q (2197, X'895'). Unknown default transmission queue.
Event description:	An MQOPEN or MQPUT1 call was issued specifying a remote queue as the destination. If a local definition of the remote queue was specified, or if a queue-manager alias is being resolved, the *XmitQName* attribute in the local definition is blank. No queue is defined with the same name as the destination queue manager. The queue manager has therefore attempted to use the default transmission queue. However, the name defined by the *DefXmitQName* queue-manager attribute is not the name of a locally-defined queue.
Event type:	Remote.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

QName

Description: Queue name from object descriptor (MQOD).
Identifier: MQCA_Q_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

XmitQName

Description: Default transmission queue name.
Identifier: MQCA_XMIT_Q_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

ApplType

Description: Type of application attempting to open the remote queue.
Identifier: MQIA_APPL_TYPE.
Datatype: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application attempting to open the remote queue.

Unknown Default Transmission Queue

Identifier:	MQCACF_APPL_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_APPL_NAME_LENGTH.
Returned:	Always.

ObjectQMgrName

Description:	Name of the object queue manager.
Identifier:	MQCACF_OBJECT_Q_MGR_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	If the *ObjectName* in the object descriptor (MQOD), when the object was opened, is not the queue manager currently connected.

Note: If the application is a server for clients, the *ApplType* and *ApplName* parameters identify the server, not the client.

Unknown Object Name

Event name:	Unknown Object Name.
Reason code in MQCFH:	MQRC_UNKNOWN_OBJECT_NAME (2085, X'825'). Unknown object name.
Event description:	On an MQOPEN or MQPUT1 call, the *ObjectQMgrName* field in the object descriptor MQOD is set to one of the following. It is either: • Blank • The name of the local queue manager • The name of a local definition of a remote queue (a queue-manager alias) in which the *RemoteQMgrName* attribute is the name of the local queue manager However, the *ObjectName* in the object descriptor is not recognized for the specified object type. See also MQRC_Q_DELETED.
Event type:	Local.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

ApplType

Description:	Type of application making the MQI call that caused the event.
Identifier:	MQIA_APPL_TYPE.
Datatype:	MQCFIN.
Returned:	Always.

ApplName

Description:	Name of the application making the MQI call that caused the event.
Identifier:	MQCACF_APPL_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_APPL_NAME_LENGTH.
Returned:	Always.

QName

Description:	Queue name from object descriptor (MQOD).
Identifier:	MQCA_BASE_Q_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always unless *ProcessName* is returned. Either *ProcessName* or *QName* is returned.

Unknown Object Name

ProcessName

Description:	Name of the process (application) making the MQI call that caused the event.
Identifier:	MQCA_PROCESS_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_PROCESS_NAME_LENGTH.
Returned:	Always, unless *QName* is returned. Either *ProcessName* or *QName* is returned.

ObjectQMgrName

Description:	Name of the object queue manager.
Identifier:	MQCACF_OBJECT_Q_MGR_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	If the *ObjectName* in the object descriptor (MQOD), when the object was opened, is not the queue manager currently connected.

Note: If the application is a server for clients, the *ApplType* and *ApplName* parameters identify the server, not the client.

Unknown Remote Queue Manager

Event name:	Unknown Remote Queue Manager.
Reason code in MQCFH:	MQRC_UNKNOWN_REMOTE_Q_MGR (2087, X'827'). Unknown remote queue manager.
Event description:	On an MQOPEN or MQPUT1 call, an error occurred with the queue-name resolution, for one of the following reasons: • *ObjectQMgrName* is either blank or the name of the local queue manager, and *ObjectName* is the name of a local definition of a remote queue that has a blank *XmitQName*. However, there is no (transmission) queue defined with the name of *RemoteQMgrName*, and the *DefXmitQName* queue-manager attribute is blank. • *ObjectQMgrName* is the name of a queue-manager alias definition (held as the local definition of a remote queue) that has a blank *XmitQName*. However, there is no (transmission) queue defined with the name of *RemoteQMgrName*, and the *DefXmitQName* queue-manager attribute is blank. • *ObjectQMgrName* specified is not: – Blank – The name of the local queue manager – The name of a local queue – The name of a queue-manager alias definition (that is, a local definition of a remote queue with a blank *RemoteQName*) and the *DefXmitQName* queue-manager attribute is blank. • *ObjectQMgrName* is blank or is the name of the local queue manager, and *ObjectName* is the name of a local definition of a remote queue (or an alias to one), for which *RemoteQMgrName* is either blank or is the name of the local queue manager. Note that this error occurs even if the *XmitQName* is not blank. • *ObjectQMgrName* is the name of a local definition of a remote queue. In this case, it should be a queue-manager alias definition, but the *RemoteQName* in the definition is not blank. • *ObjectQMgrName* is the name of a model queue. • The queue name is resolved through a cell directory. However, there is no queue defined with the same name as the remote queue manager name obtained from the cell directory. Also, the *DefXmitQName* queue-manager attribute is blank.
Event type:	Remote.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description:	Name of the queue manager generating the event.
Identifier:	MQCA_Q_MGR_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

QName

Description:	Queue name from object descriptor (MQOD).
Identifier:	MQCA_Q_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.

Unknown Remote Queue Manager

Returned: Always.

ApplType

Description: Type of application attempting to open the remote queue.
Identifier: MQIA_APPL_TYPE.
Datatype: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application attempting to open the remote queue.
Identifier: MQCACF_APPL_NAME.
Datatype: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.
Returned: Always.

ObjectQMgrName

Description: Name of the object queue manager.
Identifier: MQCACF_OBJECT_Q_MGR_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: If the *ObjectName* in the object descriptor (MQOD), when the object was opened, is not the queue manager currently connected.

Note: If the application is a server for clients, the *ApplType* and *ApplName* parameters identify the server, not the client.

Unknown Transmission Queue

Event name:	Unknown Transmission Queue.
Reason code in MQCFH:	MQRC_UNKNOWN_XMIT_Q (2196, X'894'). Unknown transmission queue.
Event description:	On an MQOPEN or MQPUT1 call, a message is to be sent to a remote queue manager. The *ObjectName* or the *ObjectQMgrName* in the object descriptor specifies the name of a local definition of a remote queue (in the latter case queue-manager aliasing is being used). However, the *XmitQName* attribute of the definition is not blank and not the name of a locally-defined queue.
Event type:	Remote.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.QMGR.EVENT.

Event data

QMgrName

Description: Name of the queue manager generating the event.
Identifier: MQCA_Q_MGR_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: Always.

QName

Description: Queue name from object descriptor (MQOD).
Identifier: MQCA_Q_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

XmitQName

Description: Transmission queue name.
Identifier: MQCA_XMIT_Q_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_NAME_LENGTH.
Returned: Always.

ApplType

Description: Type of application making the MQI call that caused the event.
Identifier: MQIA_APPL_TYPE.
Datatype: MQCFIN.
Returned: Always.

ApplName

Description: Name of the application making the MQI call that caused the event.
Identifier: MQCACF_APPL_NAME.
Datatype: MQCFST.
Maximum length: MQ_APPL_NAME_LENGTH.

Unknown Transmission Queue

Returned: Always.

ObjectQMgrName

Description: Name of the object queue manager.
Identifier: MQCACF_OBJECT_Q_MGR_NAME.
Datatype: MQCFST.
Maximum length: MQ_Q_MGR_NAME_LENGTH.
Returned: If the *ObjectName* in the object descriptor (MQOD), when the object was opened, is not the queue manager currently connected.

Note: If the application is a server for clients, the *ApplType* and *ApplName* parameters identify the server, not the client.

Chapter 5. Example of using instrumentation events

This example shows how to write a program for instrumentation events. It is written for queue managers in C, for information about which platforms support C see the *WebSphere MQ Application Programming Reference* manual. It is not part of any WebSphere MQ product and is therefore supplied as source only. The example is incomplete in that it does not enumerate all the possible outcomes of specified actions. Bearing this in mind, you can use this sample as a basis for your own programs that use events, in particular, the PCF formats used in event messages. However, you will need to modify this program to get it to run on your systems.

```
/*********************************************************************/
/*                                                                   */
/* Program name: EVMON                                               */
/*                                                                   */
/* Description: C program that acts as an event monitor              */
/*                                                                   */
/*                                                                   */
/*********************************************************************/
/*                                                                   */
/* Function:                                                         */
/*                                                                   */
/*                                                                   */
/*   EVMON is a C program that acts as an event monitor - reads an   */
/*   event queue and tells you if anything appears on it             */
/*                                                                   */
/*   Its first parameter is the queue manager name, the second is    */
/*   the event queue name. If these are not supplied it uses the     */
/*   defaults.                                                       */
/*                                                                   */
/*********************************************************************/
#include <time.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#ifndef min
  #define min(a,b)        (((a) < (b)) ? (a) : (b))
#endif
#ifdef OS2
  /*******************************************************************/
  /* for beep                                                        */
  /*******************************************************************/
  #define INCL_DOSPROCESS
  #include <os2.h>
#endif
```

Figure 10. Event monitoring sample program (Part 1 of 11)

Example using events

```
/*************************************************************************/
/* includes for MQI                                                      */
/*************************************************************************/
#include <cmqc.h>
#include <cmqcfc.h>
void printfmqcfst(MQCFST* pmqcfst);
void printfmqcfin(MQCFIN* pmqcfst);
void printreas(MQLONG reason);

 #define PRINTREAS(param)                                              \
    case param:                                                        \
      printf("Reason = %s\n",#param);                                  \
      break;

/*************************************************************************/
/* global variable                                                       */
/*************************************************************************/
MQCFH    *evtmsg;                    /* evtmsg message buffer       */

int main(int argc, char **argv)
{
  /***********************************************************************/
  /* declare variables                                                   */
  /***********************************************************************/
  int  i;                             /* auxiliary counter          */
  /***********************************************************************/
  /* Declare MQI structures needed                                       */
  /***********************************************************************/
  MQOD     od = {MQOD_DEFAULT};      /* Object Descriptor           */
  MQMD     md = {MQMD_DEFAULT};      /* Message Descriptor          */
  MQGMO   gmo = {MQGMO_DEFAULT};     /* get message options         */
  /***********************************************************************/
  /* note, uses defaults where it can                                    */
  /***********************************************************************/
```

Figure 10. Event monitoring sample program (Part 2 of 11)

```
MQHCONN  Hcon;                   /* connection handle          */
MQHOBJ   Hobj;                   /* object handle              */
MQLONG   O_options;              /* MQOPEN options             */
MQLONG   C_options;              /* MQCLOSE options            */
MQLONG   CompCode;               /* completion code            */
MQLONG   OpenCode;               /* MQOPEN completion code     */
MQLONG   Reason;                 /* reason code                */
MQLONG   CReason;                /* reason code for MQCONN     */
MQLONG   buflen;                 /* buffer length              */
MQLONG   evtmsglen;              /* message length received    */
MQCHAR   command[1100];          /* call command string ...    */
MQCHAR   p1[600];                /* ApplId insert              */
MQCHAR   p2[900];                /* evtmsg insert              */
MQCHAR   p3[600];                /* Environment insert         */
MQLONG   mytype;                 /* saved application type     */
char     QMName[50];             /* queue manager name         */
MQCFST  *paras;                  /* the parameters             */
int      counter;                /* loop counter               */
time_t   ltime;

/*****************************************************************/
/* Connect to queue manager                                      */
/*****************************************************************/
QMName[0] = 0;                    /* default queue manager      */
if (argc > 1)
  strcpy(QMName, argv[1]);
MQCONN(QMName,                    /* queue manager              */
       &Hcon,                  /* connection handle          */
       &CompCode,              /* completion code            */
       &CReason);              /* reason code                */

/*****************************************************************/
/* Initialize object descriptor for subject queue                */
/*****************************************************************/
strcpy(od.ObjectName, "SYSTEM.ADMIN.QMGR.EVENT");
if (argc > 2)
  strcpy(od.ObjectName, argv[2]);

/*****************************************************************/
/* Open the event queue for input; exclusive or shared.  Use of  */
/* the queue is controlled by the queue definition here          */
/*****************************************************************/
```

Figure 10. Event monitoring sample program (Part 3 of 11)

Example using events

```
O_options = MQOO_INPUT_AS_Q_DEF     /* open queue for input       */
      + MQOO_FAIL_IF_QUIESCING      /* but not if qmgr stopping   */
      + MQOO_BROWSE;
MQOPEN(Hcon,                        /* connection handle          */
       &od,                    /* object descriptor for queue*/
       O_options,                   /* open options               */
       &Hobj,                  /* object handle              */
       &CompCode,              /* completion code            */
       &Reason);               /* reason code                */

/*****************************************************************/
/*   Get messages from the message queue                         */
/*****************************************************************/
while (CompCode != MQCC_FAILED)
{
  /***************************************************************/
  /* I don't know how big this message is so just get the        */
  /* descriptor first                                            */
  /***************************************************************/
  gmo.Options = MQGMO_WAIT + MQGMO_LOCK
     + MQGMO_BROWSE_FIRST + MQGMO_ACCEPT_TRUNCATED_MSG;
                                  /* wait for new messages      */
  gmo.WaitInterval = MQWI_UNLIMITED;/* no time limit             */
  buflen = 0;                     /* amount of message to get   */

  /***************************************************************/
  /* clear selectors to get messages in sequence                 */
  /***************************************************************/
  memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
  memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));

  /***************************************************************/
  /* wait for event message                                      */
  /***************************************************************/
  printf("...>\n");
  MQGET(Hcon,                         /* connection handle          */
        Hobj,                         /* object handle              */
        &md,                    /* message descriptor         */
        &gmo,                   /* get message options        */
        buflen,                       /* buffer length              */
        evtmsg,                       /* evtmsg message buffer      */
        &evtmsglen,             /* message length             */
        &CompCode,              /* completion code            */
        &Reason);               /* reason code                */

  /***************************************************************/
  /* report reason, if any                                       */
  /***************************************************************/
```

Figure 10. Event monitoring sample program (Part 4 of 11)

```
if (Reason != MQRC_NONE && Reason != MQRC_TRUNCATED_MSG_ACCEPTED)
{
  printf("MQGET ==> %ld\n", Reason);
}
else
{
  gmo.Options = MQGMO_NO_WAIT + MQGMO_MSG_UNDER_CURSOR;
  buflen = evtmsglen;             /* amount of message to get   */
  evtmsg = malloc(buflen);
  if (evtmsg != NULL)
  {
    /*************************************************************/
    /* clear selectors to get messages in sequence               */
    /*************************************************************/
    memcpy(md.MsgId, MQMI_NONE, sizeof(md.MsgId));
    memcpy(md.CorrelId, MQCI_NONE, sizeof(md.CorrelId));

    /*************************************************************/
    /* get the event message                                     */
    /*************************************************************/
    printf("...>\n");
    MQGET(Hcon,                     /* connection handle        */
          Hobj,                     /* object handle            */
          &md,                 /* message descriptor       */
          &gmo,                /* get message options      */
          buflen,                   /* buffer length            */
          evtmsg,                   /* evtmsg message buffer    */
          &evtmsglen,          /* message length           */
          &CompCode,           /* completion code          */
          &Reason);            /* reason code              */

    /*************************************************************/
    /* report reason, if any                                     */
    /*************************************************************/
    if (Reason != MQRC_NONE)
    {
      printf("MQGET ==> %ld\n", Reason);
    }
  }
  else
  {
    CompCode = MQCC_FAILED;
  }
}
/*****************************************************************/
/* . . . process each message received                           */
/*****************************************************************/
```

Figure 10. Event monitoring sample program (Part 5 of 11)

Example using events

```
if (CompCode != MQCC_FAILED)
{
  /************************************************************/
  /* announce a message                                       */
  /************************************************************/
  #ifdef OS2
    {
      unsigned short tone;
      for (tone = 1; tone < 8000; tone = tone * 2)
      {
        DosBeep(tone,50);
      }
    }
  #else
    printf("\a\a\a\a\a\a\a");
  #endif
  time(&ltime);
  printf(ctime(&ltime));

  if (evtmsglen != buflen)
    printf("DataLength = %ld?\n", evtmsglen);
  else
  {
    /************************************************************/
    /* right let's look at the data                             */
    /************************************************************/
    if (evtmsg->Type != MQCFT_EVENT)
    {
      printf("Something's wrong this isn't an event message,"
             " its type is %ld\n",evtmsg->Type);
    }
    else
    {
      if (evtmsg->Command == MQCMD_Q_MGR_EVENT)
      {
        printf("Queue Manager event: ");
      }
      else
        if (evtmsg->Command == MQCMD_CHANNEL_EVENT)
        {
          printf("Channel event: ");
        }
        else

.
.
.
```

Figure 10. Event monitoring sample program (Part 6 of 11)

```
  {
    printf("Unknown Event message, %ld.",
            evtmsg->Command);
  }

if      (evtmsg->CompCode == MQCC_OK)
  printf("CompCode(OK)\n");
else if (evtmsg->CompCode == MQCC_WARNING)
  printf("CompCode(WARNING)\n");
else if (evtmsg->CompCode == MQCC_FAILED)
  printf("CompCode(FAILED)\n");
else
  printf("* CompCode wrong * (%ld)\n",
            evtmsg->CompCode);

if (evtmsg->StrucLength != MQCFH_STRUC_LENGTH)
{
  printf("it's the wrong length, %ld\n",evtmsg->StrucLength);
}

if (evtmsg->Version != MQCFH_VERSION_1)
{
  printf("it's the wrong version, %ld\n",evtmsg->Version);
}

if (evtmsg->MsgSeqNumber != 1)
{
  printf("it's the wrong sequence number, %ld\n",
          evtmsg->MsgSeqNumber);
}

if (evtmsg->Control != MQCFC_LAST)
{
  printf("it's the wrong control option, %ld\n",
          evtmsg->Control);
}

printreas(evtmsg->Reason);
printf("parameter count is %ld\n", evtmsg->ParameterCount);
/*********************************************************/
/* get a pointer to the start of the parameters          */
/*********************************************************/
```

Figure 10. Event monitoring sample program (Part 7 of 11)

Example using events

```
        paras = (MQCFST *)(evtmsg + 1);
        counter = 1;
        while (counter <= evtmsg->ParameterCount)
        {
          switch (paras->Type)
          {
            case MQCFT_STRING:
              printfmqcfst(paras);
              paras = (MQCFST *)((char *)paras
                                 + paras->StrucLength);
              break;
            case MQCFT_INTEGER:
              printfmqcfin((MQCFIN*)paras);
              paras = (MQCFST *)((char *)paras
                                 + paras->StrucLength);
              break;
            default:
              printf("unknown parameter type, %ld\n",
                     paras->Type);
              counter = evtmsg->ParameterCount;
              break;
          }
          counter++;
        }
      }
    }   /* end evtmsg action             */
    free(evtmsg);
  }     /* end process for successful GET */
}       /* end message processing loop    */

/*****************************************************************/
/* close the event queue - if it was opened                      */
/*****************************************************************/
if (OpenCode != MQCC_FAILED)
{
  C_options = 0;                  /* no close options            */
  MQCLOSE(Hcon,                   /* connection handle           */
          &Hobj,              /* object handle               */
          C_options,
          &CompCode,          /* completion code             */
          &Reason);           /* reason code                 */
/*****************************************************************/
/* Disconnect from queue manager (unless previously connected)   */
/*****************************************************************/
if (CReason != MQRC_ALREADY_CONNECTED)
{
  MQDISC(&Hcon,               /* connection handle           */
          &CompCode,          /* completion code             */
          &Reason);           /* reason code                 */
```

Figure 10. Event monitoring sample program (Part 8 of 11)

```
 /*********************************************************************/
 /*                                                                   */
 /* END OF EVMON                                                      */
 /*                                                                   */
 /*********************************************************************/
 }

#define PRINTPARAM(param)                                             \
   case param:                                                        \
     {                                                                \
       char *p = #param;                                            \
     strncpy(thestring,pmqcfst->String,min(sizeof(thestring),       \
             pmqcfst->StringLength));                               \
     printf("%s %s\n",p,thestring);                                 \
     }                                                              \
     break;

#define PRINTAT(param)                                                \
   case param:                                                        \
     printf("MQIA_APPL_TYPE = %s\n",#param);                          \
     break;


void printfmqcfst(MQCFST* pmqcfst)
{
  char thestring[100];

  switch (pmqcfst->Parameter)
  {
    PRINTPARAM(MQCA_BASE_Q_NAME)
    PRINTPARAM(MQCA_PROCESS_NAME)
    PRINTPARAM(MQCA_Q_MGR_NAME)
    PRINTPARAM(MQCA_Q_NAME)
    PRINTPARAM(MQCA_XMIT_Q_NAME)
    PRINTPARAM(MQCACF_APPL_NAME)

   .
   .
   .
     default:
      printf("Invalid parameter, %ld\n",pmqcfst->Parameter);
      break;
  }
}
```

Figure 10. Event monitoring sample program (Part 9 of 11)

Example using events

```
void printfmqcfin(MQCFIN* pmqcfst)
{
  switch (pmqcfst->Parameter)
  {
    case MQIA_APPL_TYPE:
      switch (pmqcfst->Value)
      {
        PRINTAT(MQAT_UNKNOWN)
        PRINTAT(MQAT_OS2)
        PRINTAT(MQAT_DOS)
        PRINTAT(MQAT_UNIX)
        PRINTAT(MQAT_QMGR)
        PRINTAT(MQAT_OS400)
        PRINTAT(MQAT_WINDOWS)
        PRINTAT(MQAT_CICS_VSE)
        PRINTAT(MQAT_VMS)
        PRINTAT(MQAT_GUARDIAN)
        PRINTAT(MQAT_VOS)
      }
      break;
    case MQIA_Q_TYPE:
      if (pmqcfst->Value == MQQT_ALIAS)
      {
        printf("MQIA_Q_TYPE is MQQT_ALIAS\n");
      }
      else

   .
   .
   .
{
        if (pmqcfst->Value == MQQT_REMOTE)
        {
          printf("MQIA_Q_TYPE is MQQT_REMOTE\n");
          if (evtmsg->Reason == MQRC_ALIAS_BASE_Q_TYPE_ERROR)
          {
            printf("but remote is not valid here\n");
          }
        }
        else
        {
          printf("MQIA_Q_TYPE is wrong, %ld\n",pmqcfst->Value);
        }
      }
      break;
```

Figure 10. Event monitoring sample program (Part 10 of 11)

```
          case MQIACF_REASON_QUALIFIER:
      printf("MQIACF_REASON_QUALIFIER %ld\n",pmqcfst->Value);
      break;

    case MQIACF_ERROR_IDENTIFIER:
      printf("MQIACF_ERROR_INDENTIFIER %ld (X'%lX')\n",
              pmqcfst->Value,pmqcfst->Value);
      break;

    case MQIACF_AUX_ERROR_DATA_INT_1:
      printf("MQIACF_AUX_ERROR_DATA_INT_1 %ld (X'%lX')\n",
              pmqcfst->Value,pmqcfst->Value);
      break;

    case MQIACF_AUX_ERROR_DATA_INT_2:
      printf("MQIACF_AUX_ERROR_DATA_INT_2 %ld (X'%lX')\n",
              pmqcfst->Value,pmqcfst->Value);
      break;
   .
   .
   .
default :
      printf("Invalid parameter, %ld\n",pmqcfst->Parameter);
      break;
  }
}

   void printreas(MQLONG reason)
{
  switch (reason)
  {
    PRINTREAS(MQRCCF_CFH_TYPE_ERROR)
    PRINTREAS(MQRCCF_CFH_LENGTH_ERROR)
    PRINTREAS(MQRCCF_CFH_VERSION_ERROR)
    PRINTREAS(MQRCCF_CFH_MSG_SEQ_NUMBER_ERR)

   .
   .
   .
    PRINTREAS(MQRC_NO_MSG_LOCKED)
    PRINTREAS(MQRC_CONNECTION_NOT_AUTHORIZED)
    PRINTREAS(MQRC_MSG_TOO_BIG_FOR_CHANNEL)
    PRINTREAS(MQRC_CALL_IN_PROGRESS)
    default:
      printf("It's an unknown reason, %ld\n",
              reason);
      break;
  }
}
```

Figure 10. Event monitoring sample program (Part 11 of 11)

Appendix A. Structure datatypes MQCFBS, MQCFIN, MQCFSL and MQCFST

In this appendix, the structures MQCFBS, MQCFIN, MQCFSL and MQCFST are described in a language-independent form. The declarations are shown in the following programming languages:

- C
- COBOL
- PL/I
- RPG (ILE) (OS/400 only)
- S/390® assembler (z/OS only)
- Visual Basic (Windows platforms only)

Where the platform is not declared in the list above, see the *WebSphere MQ Application Programming Reference* manual for information about which compilers and programming languages are supported on each platform.

The elementary data types of the fields in MQCFBS, MQCFIN, MQCFSL and MQCFST are described in the *WebSphere MQ Application Programming Reference* manual.

The *initial value* of each field is shown under its description. This is the value of the field in the *default structure*.

MQCFBS - Byte string parameter

The MQCFBS structure describes an byte string parameter in an event message.

Type

Description: This indicates that the structure is an MQCFBS structure describing a byte string parameter.
Datatype: MQLONG.
Initial value: MQCFT_BYTE_STRING.
Valid value:

MQCFT_BYTE_STRING
: Structure defining a byte string.

StrucLength

Description: 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.
Datatype: MQLONG.
Initial value: MQCFBS_STRUC_LENGTH_FIXED.
Valid value:

MQCFBS_STRUC_LENGTH_FIXED
: Length of fixed part of the MQCFBS structure, that is the length excluding the *String* field.

Parameter

Description: 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.
Datatype: MQLONG.
Initial value: 0.

StringLength

Description: 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.
Datatype: MQLONG.
Initial value: 0.

String

Description: 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 byte in the string is treated as normal data, and does not act as a delimiter for the string.
Datatype: MQBYTE ×*StringLength*.
Initial value: Null string. (Only present in C)

C language declaration (MQCFBS)

```
typedef struct tagMQCFBS MQCFBS;
struct tagMQCFBS {
  MQLONG  Type;          /* Structure type */
  MQLONG  StrucLength;   /* Structure length */
  MQLONG  Parameter;     /* Parameter identifier */
  MQLONG  StringLength;  /* Length of string */
  MQBYTE  String[1];     /* String value -- first character */
};
```

COBOL language declaration (MQCFBS)

```
**   MQCFBS structure
  10 MQCFBS.
**    Structure type
   15 MQCFBS-TYPE         PIC S9(9) BINARY.
**    Structure length
   15 MQCFBS-STRUCLENGTH  PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFBS-PARAMETER    PIC S9(9) BINARY.
**    Length of string
   15 MQCFBS-STRINGLENGTH PIC S9(9) BINARY.
```

PL/I language declaration (MQCFBS) (z/OS only)

```
dcl
 1 MQCFBS based,
  3 Type         fixed bin(31), /* Structure type */
  3 StrucLength  fixed bin(31), /* Structure length */
  3 Parameter    fixed bin(31), /* Parameter identifier */
  3 StringLength fixed bin(31); /* Length of string */
```

System/390 assembler-language declaration (MQCFBS) (z/OS only)

```
MQCFBS               DSECT
MQCFBS_TYPE          DS   F  Structure type
MQCFBS_STRUCLENGTH   DS   F  Structure length
MQCFBS_PARAMETER     DS   F  Parameter identifier
MQCFBS_STRINGLENGTH  DS   F  Length of string
*
MQCFBS_LENGTH        EQU  *-MQCFBS
                     ORG  MQCFBS
MQCFBS_AREA          DS   CL(MQCFBS_LENGTH)
```

MQCFIN - Integer parameter

The MQCFIN structure describes an integer parameter in an event message.

Type

Description: Indicates that the structure type is MQCFIN and describes an integer parameter.
Datatype: MQLONG.
Initial value: MQCFT_INTEGER.
Valid value:

MQCFT_INTEGER
: Structure defining an integer.

StrucLength

Description: Length in bytes of the MQCFIN structure.
Datatype: MQLONG.
Initial value: MQCFIN_STRUC_LENGTH.
Valid value:

MQCFIN_STRUC_LENGTH
: Length of MQCFIN structure.

Parameter

Description: Identifies the parameter whose value is contained in the structure.
Datatype: MQLONG.
Initial value: 0.
Valid values: Dependent on the event message.

Value

Description: Value of parameter identified by the *Parameter* field.
Datatype: MQLONG.
Initial value: 0.

C language declaration (MQCFIN)

```
typedef struct tagMQCFIN {
  MQLONG  Type;         /* Structure type */
  MQLONG  StrucLength;  /* Structure length */
  MQLONG  Parameter;    /* Parameter identifier */
  MQLONG  Value;        /* Parameter value */
 } MQCFIN;
```

COBOL language declaration (MQCFIN)

```
**   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 (MQCFIN)

```
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 */
```

RPG/ILE declaration (MQCFIN) (OS/400 only)

```
     D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
     D* MQCFIN Structure
     D*
     D* Structure type
     D  INTYP                  1      4I 0
     D* Structure length
     D  INLEN                  5      8I 0
     D* Parameter identifier
     D  INPRM                  9     12I 0
     D* Parameter value
     D  INVAL                 13     16I 0
```

System/390 assembler-language declaration (MQCFIN)

```
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 (MQCFIN)

```
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
```

MQCFSL - String list parameter

The MQCFSL structure describes a string list parameter in an event message.

Type

Description: This indicates that the structure is an MQCFSL structure describing a string-list parameter.
Datatype: MQLONG.
Initial value: MQCFT_STRING_LIST.
Valid value:

MQCFT_STRING_LIST
: Structure defining a string list.

StrucLength

Description: 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.
Datatype: MQLONG.
Initial value: MQCFSL_STRUC_LENGTH_FIXED.
Valid value:

MQCFSL_STRUC_LENGTH_FIXED
: Length of fixed part, that is the length excluding the *Strings* field, of the command format string-list parameter structure.

Parameter

Description: 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.
Datatype: MQLONG.
Initial value: 0.

CodedCharSetId

Description: This specifies the coded character set identifier of the data in the *Strings* field.
Datatype: MQLONG.
Initial value: MQCCSI_DEFAULT.
Valid value:

MQCFSI_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.

Count

Description: This is the number of strings present in the *Strings* field; it must be zero or greater.
Datatype: MQLONG.
Initial value: 0.

StringLength

Description: 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.
Datatype: MQLONG.

Initial value: 0.

String

Description: 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_EVENT messages, trailing blanks are omitted from string parameters (that is, the string may be shorter than the defined length of the parameter). *StringLength* gives the length of the string actually present in the message.
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_EVENT message, the receiving application receives all of the data specified by the sending application. The data may, of course, have been converted between character sets (for example, by the receiving application specifying the MQGMO_CONVERT option on the MQGET call).

In contrast, when the queue manager reads an MQFMT_ADMIN message from the command input queue, the queue manager processes the data as though it had been specified on an MQI call. This means that within each string, the first null and the characters following it (up to the end of the string) are treated as blanks.

Datatype: MQCHAR × *StringLength*×*Count*.
Initial value: Null string. (Present only in C).

COBOL language declaration (MQCFSL)

```
**   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 (MQCFSL)

```
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 */
```

RPG/ILE declaration (MQCFSL) (OS/400 only)

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQCFSL Structure
D*
D* Structure type
D  SLTYP                  1      4I 0
D* Structure length
D  SLLEN                  5      8I 0
D* Parameter identifier
D  SLPRM                  9     12I 0
D* Coded character set identifier
D  SLCSI                 13     16I 0
D* Count of parameter values
D  SLCNT                 17     20I 0
D* Length of one string
D  SLSTL                 21     24I 0
```

System/390 assembler-language declaration (MQCFSL) (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
                       ORG  MQCFSL
MQCFSL_AREA            DS   CL(MQCFSL_LENGTH)
```

Visual Basic language declaration (MQCFSL) (Windows systems 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
```

MQCFST - String parameter

The MQCFST structure describes a string parameter in an event message.

The structure ends with a variable-length character string; see the *String* field below for further details.

Type

Description: Indicates that the structure type is MQCFST and describes a string parameter.

Datatype: MQLONG.

Initial value: MQCFT_STRING.

Valid value:

MQCFT_STRING
: Structure defining a string.

StrucLength

Description: 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.

Datatype: MQLONG.

Initial value:

MQCFST_STRUC_LENGTH_FIXED.
Length of the *fixed* part of the MQCFST structure, excluding the *String* field.

Parameter

Description: Identifies the parameter whose value is contained in the structure.

Datatype: MQLONG.

Initial value: 0.

Valid values: Dependent on the event message.

CodedCharSetId

Description: Coded character set identifier of the data in the *String* field.

Datatype: MQLONG.

Initial value:

MQCCSI_DEFAULT.
Default coded character set identifier, indicating that character 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.

Valid values:

- If all of the strings in an event message have the same coded character-set identifier, the *CodedCharSetId* field in the message descriptor MQMD or in the MQ header structure preceding MQCFH should be set to that identifier when the message is put, and the *CodedCharSetId* fields in the MQCFST structure 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 or in the MQ header structure preceding MQCFH should be set to MQCCSI_EMBEDDED when the message is put, and the *CodedCharSetId* fields in the MQCFST structure within the message should be set to the identifiers that apply.

Do not specify MQCCSI_EMBEDDED in MQMD or in the MQ header structure preceding MQCFH when the message is put, with MQCCSI_DEFAULT in the MQCFST structure within the message, as this will prevent conversion of the message.

StringLength

Description: 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.

Datatype: MQLONG.

Initial value: 0.

String

Description: The value of the parameter identified by the *Parameter* field.

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). *StringLength* gives the length of the string actually present in the message.

Datatype: MQCHAR×*StringLength*.

Initial value: In C, the initial value of this field is the null string.

Valid value: 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*.

Language considerations: The way that this field is declared depends on the programming language:

- For the C programming language, the field is declared as an array with one element. Storage for the structure should be allocated dynamically, and pointers used to address the fields within it.
- For the COBOL, PL/I, System/390 assembler, and Visual Basic programming languages, the field is omitted from the structure declaration. When an instance of the structure is declared, the user should include MQCFST in a larger structure, and declare additional fields following MQCFST, to represent the *String* field as required.

Special note: 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 an MQFMT_EVENT message, the receiving application receives all of the data specified by the sending application. The data may, of course, have been converted between character sets (for example, by the receiving application specifying the MQGMO_CONVERT option on the MQGET call).

C language declaration (MQCFST)

```
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;
```

In the C programming language, the macro variable MQCFST_DEFAULT contains the initial values of the MQCFST structure. 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};
```

COBOL language declaration (MQCFST)

```
**   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 (MQCFST)

```
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 */
```

RPG/ILE declaration (MQCFST) (OS/400 only)

```
     D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
     D* MQCFST Structure
     D*
     D* Structure type
     D  STTYP                  1      4I 0
     D* Structure length
     D  STLEN                  5      8I 0
     D* Parameter identifier
     D  STPRM                  9     12I 0
     D* Coded character set identifier
     D  STCSI                 13     16I 0
     D* Length of string
     D  STSTL                 17     20I 0
```

System/390 assembler-language declaration (MQCFST)

```
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 (MQCFST)

```
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
```

Appendix B. Constants

This appendix specifies the values of the named constants that apply to events.

The constants are grouped according to the parameter or field to which they relate. All the names of the constants in a group begin with a common prefix of the form MQ*xxxx*_, where *xxxx* represents a string of 0 through 4 characters that indicates the nature of the values defined in that group. The constants are ordered alphabetically by the prefix.

Notes:

1. For constants with numeric values, the values are shown in both decimal and hexadecimal forms.
2. Hexadecimal values are represented using the notation X'hhhh', where each h denotes a single hexadecimal digit.
3. Character values are shown delimited by single quotation marks; the quotation marks are not part of the value.
4. Blanks in character values are represented by one or more occurrences of the symbol b.

List of constants

The following sections list all the named constants mentioned in this book, with their values.

MQ_* (Lengths of character string and byte fields)

MQ_APPL_NAME_LENGTH	28	X'0000001C'
MQ_AUTH_INFO_NAME_LENGTH	48	X'00000030'
MQ_BRIDGE_NAME_LENGTH	24	X'00000018'
MQ_CHANNEL_NAME_LENGTH	20	X'00000014'
MQ_CONN_NAME_LENGTH	264	X'00000108'
MQ_FORMAT_LENGTH	8	X'00000008'
MQ_PROCESS_NAME_LENGTH	48	X'00000030'
MQ_SSL_HANDSHAKE_STAGE_LENGTH	32	X'00000020'
MQ_STORAGE_CLASS_LENGTH	8	X'00000008'
MQ_STORAGE_CLASS_DESC_LENGTH	64	X'00000040'
MQ_Q_MGR_NAME_LENGTH	48	X'00000030'
MQ_Q_NAME_LENGTH	48	X'00000030'
MQ_USER_ID_LENGTH	12	X'0000000C'
MQ_XCF_GROUP_NAME_LENGTH	8	X'00000008'
MQ_XCF_MEMBER_NAME_LENGTH	16	X'00000010'

MQBACF_* (Byte attribute command format parameter)

MQBACF_EVENT_ACCOUNTING_TOKEN	7001	X'00001B59'
MQBACF_EVENT_SECURITY_ID	7002	X'00001B5A'

MQBT_* (Bridge type)

MQBT_OTMA	1	X'00000001'

MQCA_* (Character attribute selector)

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_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_Q_MGR_IDENTIFIER	2032	X'000007F0'
MQCA_CLUSTER_WORKLOAD_EXIT	2033	X'000007F1'
MQCA_CLUSTER_WORKLOAD_DATA	2034	X'000007F2'
MQCA_CF_STRUC_NAME	2039	X'000007F7'
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_AUTH_INFO_CONN_NAME	2053	X'00000805'
MQCA_LAST_USED	2053	X'00000805'

MQCACF_* (Character attribute command format parameter)

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_EVENT_APPL_IDENTITY	3049	X'00000BE9'
MQCACF_EVENT_APPL_NAME	3050	X'00000BEA'
MQCACF_EVENT_APPL_ORIGIN	3051	X'00000BEB'
MQCACF_LAST_USED	3051	X'00000BEB'

MQCACH_* (Channel character attribute command format parameter)

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_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_LOCAL_ADDRESS	3520	X'00000DC0'
MQCACH_PASSWORD	3518	X'00000DBE'
MQCACH_MCA_USER_ID	3527	X'00000DC7'
MQCACH_FORMAT_NAME	3533	X'00000DCD'
MQCACH_SSL_CIPHER_SPEC	3544	X'00000DD8'
MQCACH_SSL_PEER_NAME	3545	X'00000DD9'
MQCACH_SSL_HANDSHAKE_STAGE	3546	X'00000DDA'

MQCC_* (Completion code)

MQCC_OK	0	X'00000000'
MQCC_WARNING	1	X'00000001'

MQCFC_* (Command format control options)

MQCFC_LAST	1	X'00000001'

MQCFH_* (Command format header structure length)

MQCFH_STRUC_LENGTH	36	X'00000024'

Constants

MQCFH_* (Command format header version)

MQCFH_VERSION_1	1	X'00000001'

MQCFIN_* (Command format integer parameter structure length)

MQCFIN_STRUC_LENGTH	16	X'00000010'

MQCFST_* (Command format string parameter structure length)

MQCFST_STRUC_LENGTH_FIXED	20	X'00000014'

MQCFT_* (Command structure type)

MQCFT_COMMAND	1	X'00000001'
MQCFT_INTEGER	3	X'00000003'
MQCFT_STRING	4	X'00000004'
MQCFT_EVENT	7	X'00000007'

MQCHT_* (Channel type)

MQCHT_SENDER	1	X'00000001'
MQCHT_SERVER	2	X'00000002'
MQCHT_RECEIVER	3	X'00000003'
MQCHT_REQUESTER	4	X'00000004'
MQCHT_CLNTCONN	6	X'00000006'
MQCHT_SVRCONN	7	X'00000007'
MQCHT_CLUSRCVR	8	X'00000008'
MQCHT_CLUSSDR	9	X'00000009'

MQCMD_* (Command identifier)

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'

MQEVO_* (Event origin)

MQEVO_OTHER	0	X'00000000'
MQEVO_CONSOLE	1	X'00000001'
MQEVO_INIT	2	X'00000002'
MQEVO_MSG	3	X'00000003'
MQEVO_MQSET	4	X'00000004'
MQEVO_INTERNAL	5	X'00000005'

MQIA_* (Integer attribute selector)

MQIA_APPL_TYPE	1	X'00000001'
MQIA_CODED_CHAR_SET_ID	2	X'00000002'
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_Q_TYPE	20	X'00000014'
MQIA_RETENTION_INTERVAL	21	X'00000015'
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_UNCOMMITED_MSGS	33	X'00000021'
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_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_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_INDEX_TYPE	57	X'00000039'
MQIA_CLUSTER_WORKLOAD_LENGTH	58	X'0000003A'
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	66	X'00000042'
MQIA_SSL_TASKS	69	X'00000045'

MQIACF_* (Integer attribute command format parameter)

MQIACF_EVENT_APPL_TYPE	1010	X'000003F2'
MQIACF_EVENT_ORIGIN	1011	X'000003F3'
MQIACF_ERROR_IDENTIFIER	1013	X'000003F5'
MQIACF_OBJECT_TYPE	1016	X'000003F8'
MQIACF_REASON_QUALIFIER	1020	X'000003FC'
MQIACF_COMMAND	1021	X'000003FD'
MQIACF_OPEN_OPTIONS	1022	X'000003FE'
MQIACF_AUX_ERROR_DATA_INT_1	1070	X'0000042E'
MQIACF_AUX_ERROR_DATA_INT_2	1071	X'0000042F'
MQIACF_CONV_REASON_CODE	1072	X'00000430'
MQIACF_BRIDGE_TYPE	1073	X'00000431'
MQIACF_REFRESH_TYPE	1078	X'00000436'

MQIACH_* (Channel Integer attribute command format parameter)

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_CONVERSION	1515	X'000005EB'
MQIACH_MCA_TYPE	1517	X'000005ED'
MQIACH_SSL_RETURN_CODE	1533	X'000005FD'
MQIACH_NPM_SPEED	1562	X'0000061A'
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'

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'

MQQSGD_* (Queue Sharing Group Disposition)

MQQSGD_Q_MGR	0	X'00000000'

MQQSGD_COPY	1	X'00000001'
MQQSGD_SHARED	2	X'00000002'
MQQSGD_GROUP	3	X'00000003'

MQQT_* (Queue type)

MQQT_LOCAL	1	X'00000001'
MQQT_MODEL	2	X'00000002'
MQQT_ALIAS	3	X'00000003'
MQQT_REMOTE	6	X'00000006'

MQRC_* (Reason code in MQCFH)

MQRC_ALIAS_BASE_Q_TYPE_ERROR	2001	X'000007D1'
MQRC_BRIDGE_STARTED	2125	X'0000084D'
MQRC_BRIDGE_STOPPED	2126	X'0000084E'
MQRC_CHANNEL_ACTIVATED	2295	X'000008F7'
MQRC_CHANNEL_AUTO_DEF_ERROR	2234	X'000008BA'
MQRC_CHANNEL_AUTO_DEF_OK	2233	X'000008B9'
MQRC_CHANNEL_CONV_ERROR	2284	X'000008EC'
MQRC_CHANNEL_NOT_ACTIVATED	2296	X'000008F8'
MQRC_CHANNEL_SSL_ERROR	2371	X'00000943'
MQRC_CHANNEL_STARTED	2282	X'000008EA'
MQRC_CHANNEL_STOPPED	2283	X'000008EB'
MQRC_CHANNEL_STOPPED_BY_USER	2279	X'000008E7'
MQRC_CONFIG_CHANGE_OBJECT	2368	X'00000940'
MQRC_CONFIG_CREATE_OBJECT	2367	X'0000093F'
MQRC_CONFIG_DELETE_OBJECT	2369	X'00000941'
MQRC_CONFIG_REFRESH_OBJECT	2370	X'00000942'
MQRC_DEF_XMIT_Q_TYPE_ERROR	2198	X'00000896'
MQRC_DEF_XMIT_Q_USAGE_ERROR	2199	X'00000897'
MQRC_GET_INHIBITED	2016	X'000007E0'
MQRC_NOT_AUTHORIZED	2035	X'000007F3'
MQRC_PUT_INHIBITED	2051	X'00000803'
MQRC_Q_DEPTH_HIGH	2224	X'000008B0'
MQRC_Q_DEPTH_LOW	2225	X'000008B1'
MQRC_Q_FULL	2053	X'00000805'
MQRC_Q_MGR_ACTIVE	2222	X'000008AE'
MQRC_Q_MGR_NOT_ACTIVE	2223	X'000008AF'
MQRC_Q_SERVICE_INTERVAL_HIGH	2226	X'000008B2'
MQRC_Q_SERVICE_INTERVAL_OK	2227	X'000008B3'
MQRC_Q_TYPE_ERROR	2057	X'00000809'
MQRC_REMOTE_Q_NAME_ERROR	2184	X'00000888'
MQRC_UNKOWN_ALIAS_BASE_Q	2082	X'00000822'
MQRC_UNKNOWN_DEF_XMIT_Q	2197	X'00000895'
MQRC_UNKNOWN_OBJECT_NAME	2085	X'00000825'
MQRC_UNKNOWN_REMOTE_Q_MGR	2087	X'00000827'
MQRC_UNKNOWN_XMIT_Q	2196	X'00000894'
MQRC_XMIT_Q_TYPE_ERROR	2091	X'0000082B'
MQRC_XMIT_Q_USAGE_ERROR	2092	X'0000082C'

MQRCCF_* (Reason code for command format)

MQRCCF_SUPPRESSED_BY_EXIT	4085	X'00000FF5'

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'

Appendix C. Header, COPY, and INCLUDE files

We provide various header, COPY, and INCLUDE files to help applications process event messages. These are described below for each supported programming language. Not all the files are available in all environments.

See:

C header files

The following header files are provided for the C programming language.

Table 14. C header files

File	Contents relating to this book
CMQC	Elementary data types, some named constants.
CMQCFC	PCF structures and additional named constants.
CMQXC	Named constants 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. COBOL is not supported by Windows clients.

Table 15. COBOL COPY files

File (with initial values)	File name (without initial values)	Contents relating to this book
CMQV	–	Some named constants
CMQCFV	–	Additional named constants
CMQXV	–	Named constants relating to channels
CMQCFHV	CMQCFHL	Header structure
CMQCFINV	CMQCFINL	Single-integer parameter structure
CMQCFSTV	CMQCFSTL	Single-string parameter structure
CMQCFBSV	CMQCFBSL	Byte string parameters structure
CMQCFSLV	CMQCFSLL	String list parameters structure

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

File	Contents relating to this book
CMQP	Some named constants
CMQCFP	PCF structures and additional named constants
CMQXP	Named constants relating to channels

RPG (ILE) COPY files

The following COPY files are provided for the RPG (ILE) programming language. Two COPY files are provided for each structure; one COPY file has initial values, the other does not. These files are available only on OS/400.

Table 17. RPG (ILE) COPY files

File (with initial values)	File name (without initial values)	Contents relating to this book
CMQG	–	Some named constants (not available on DOS clients and Windows clients)
CMQCFG	–	Additional named constants
CMQXG	–	Named constants relating to channels
CMQCFHG	CMQCFHH	Header structure
CMQCFING	CMQCFINH	Single-integer parameter structure
CMQCFSTG	CMQCFSTH	Single-string parameter structure
CMQCFBSG	CMQCFBSH	Byte string parameters structure
CMQCFSLG	CMQCFSLH	String list parameters structure

System/390 Assembler macros

The following macros are provided for the System/390 Assembler programming language. These files are available only on z/OS.

Table 18. System/390 Assembler macros

File	Contents relating to this book
CMQA	Some named constants
CMQCFA	Additional named constants
CMQXA	Named constants relating to channels
CMQCFHA	Header structure
CMQCFINA	Single-integer parameter structure
CMQCFSTA	Single-string parameter structure
CMQCFBSA	Byte string parameters structure
CMQCFSLA	String list parameter structure

Visual Basic header files

The following .BAS files are provided for the Visual Basic programming language. These files are available only on Windows platforms.

Table 19. Visual Basic header files

File	Contents relating to this book
CMQB	Some named constants
CMQCFB	PCF structures and additional named constants
CMQXB	Named constants relating to channels

S/390 Assembler COPY files

Appendix D. Event data for object attributes

This appendix specifies the object attributes data that can be included in the event data of configuration events.

Every object has a different amount of event data, which depends on the type of object to which the configuration event relates.

Authentication information attributes

AuthInfoType (MQCFIN)
Authentication information type (parameter identifier: MQIA_AUTH_INFO_TYPE).

The value is MQAIT_CRL_LDAP.

AuthInfoDesc (MQCFST)
Authentication information description (parameter identifier: MQIA_AUTH_INFO_DESC).

The maximum length of the string is MQ_AUTH_INFO_DESC_LENGTH.

AuthInfoConnName (MQCFST)
Authentication information connection name (parameter identifier: MQ_AUTH_INFO_CONN_NAME).

The maximum length of the string is 48.

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

The maximum length of the string is 256.

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

The maximum length of the string is MQ_LDAP_PASSWORD_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.

CF structure attributes

CFStrucDesc (MQCFST)
CF Structure description (parameter identifier: MQCA_CF_STRUC_DESC).

The maximum length of the string is MQCA_CF_STRUC_DESC_LENGTH.

CFLevel (MQCFIN)
CF level (parameter identifier: MQCA_CF_LEVEL).

Recovery (MQCFIN)
Recovery (parameter identifier: MQIA_CF_RECOVER).

Event data for object attributes

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.

Channel attributes

Only those attributes that apply to the type of channel in question are included in the event 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.

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 is MQ_EXIT_NAME_LENGTH.

Event data for object attributes

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

The number of names in the list is given by the *Count* field in the MQCFSL structure. It will be the same as the *Count* for *MsgUserData*. It may exceed the number of exit names specified for the channel, in which case the excess names are blank; the minimum is 1. The length of each name is given by the *StringLength* field in that structure.

The maximum length of the exit name is MQ_EXIT_NAME_LENGTH.

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

The number of names in the list is given by the *Count* field in the MQCFSL structure. It will be the same as the *Count* for *SendUserData*. It may exceed the number of exit names specified for the channel, in which case the excess names are blank; the minimum is 1. The length of each name is given by the *StringLength* field in that structure.

The maximum length of the exit name is MQ_EXIT_NAME_LENGTH.

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

The number of names in the list is given by the *Count* field in the MQCFSL structure. It will be the same as the *Count* for *ReceiveUserData*. It may exceed the number of exit names specified for the channel, in which case the excess names are blank; the minimum is 1. The length of each name is given by the *StringLength* field in that structure.

The maximum length of the exit name is MQ_EXIT_NAME_LENGTH.

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.

MQPA_ALTERNATE_OR_MCA
Alternate or MCA user identifier is used.

MQPA_ONLY_MCA
Only MCA 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 number of names in the list is given by the *Count* field in the MQCFSL structure. It will be the same as the count for *MsgExit*. The length of each name is given by the *StringLength* field in that structure.

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

SendUserData (MQCFSL)
Send exit user data (parameter identifier: MQCACH_SEND_EXIT_USER_DATA).

The number of names in the list is given by the *Count* field in the MQCFSL structure. It will be the same as the count for *SendExit*. The length of each name is given by the *StringLength* field in that structure.

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

ReceiveUserData (MQCFSL)
Receive exit user data (parameter identifier: MQCACH_RCV_EXIT_USER_DATA).

The number of names in the list is given by the *Count* field in the MQCFSL structure. It will be the same as the count for *ReceiveExit*. The length of each name is given by the *StringLength* field in that structure.

The maximum length of the string is MQ_EXIT_DATA_LENGTH.

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

The value can be:

MQMCAT_PROCESS
Process

MQMCAT_THREAD
Thread

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

The maximum length of the MCA user identifier is MQ_MCA_USER_ID_LENGTH.

UserIdentifier (MQCFST)
Task user identifier (parameter identifier: MQCACH_USER_ID).

The maximum length of the string is MQ_USER_ID_LENGTH.

Password (MQCFST)
Password (parameter identifier: MQCACH_PASSWORD).

The maximum length of the string is MQ_PASSWORD_LENGTH.

BatchInterval (MQCFIN)
Batch interval (parameter identifier: MQIACH_BATCH_INTERVAL).

HeartbeatInterval (MQCFIN)
Heartbeat interval (parameter identifier: MQIACH_HB_INTERVAL).

Event data for object attributes

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 999999. A value of 0 indicates heartbeating is not in use.

KeepAliveInterval (MQCFIN)
Keep alive interval (parameter identifier: MQIACH_KEEP_ALIVE_INTERVAL).

CipherSpec (MQCFST)
SSL cipher specification (parameter identifier: MQCACH_SSL_CIPHER_SPEC).

The maximum length of the string is MQ_SSL_CIPHER_SPEC_LENGTH.

PeerName (MQCFST)
SSL peer name (parameter identifier: MQCACH_SSL_PEER_NAME).

The maximum length of the string is 256.

SSLClientAuthentication (MQCFIN)
SSL client authentication (parameter identifier: MQCACH_SSL_CLIENT_AUTH).

The value can be:

MQSCA_REQUIRED
Certificate required.

MQSCA_OPTIONAL
Certificate optional.

Namelist attributes

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.

NamelistType (MQCFIN)
Namelist type (parameter identifier: MQCA_NAMELIST_TYPE).

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.

Process attributes

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).

ApplId (MQCFST)
Application identifier (parameter identifier: MQCA_APPL_ID).

The maximum length of the string is MQ_PROCESS_APPL_ID_LENGTH.

Event data for object attributes

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.

Queue attributes

Only those attributes that apply to the type of queue in question are included in the event 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_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.

Event data for object attributes

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_SHARED_DYNAMIC
Dynamically defined permanent queue that is shared.

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

The value can be:

MQUS_NORMAL
Normal usage.

MQUS_TRANSMISSION
Transmission queue.

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.

Event data for object attributes

QDepthHighLimit (MQCFIN) (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.

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.

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

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.

IndexType (MQCFIN)
Index type (parameter identifier: MQIA_INDEX_TYPE).

StorageClass (MQCFST)
Storage class name (parameter identifier: MQCA_STORAGE_CLASS).

The maximum length of the string is MQ_STORAGE_CLASS_LENGTH.

CFstructure (MQCFST)
CF structure name (parameter identifier: MQCA_CF_STRUC_NAME).

The maximum length of the string is MQ_CF_STRUC_NAME_LENGTH.

Queue manager attributes

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).

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

CPILevel (MQCFIN)
CPI level (parameter identifier: MQIA_CPI_LEVEL).

IGQPutAuthority (MQCFIN)
IGQ put authority (parameter identifier: MQIA_IGQ_PUT_AUTHORITY).

IGQUserId (MQCFST)
IGQ user identifier (parameter identifier: MQIA_IGQ_USER_ID).

The maximum length of the **string** is MQ_USER_ID_LENGTH.

IntraGroupQueueing (MQCFIN)
Intra group queueing (parameter identifier: MQIA_INTRA_GROUP_QUEUING).

ExpiryInterval (MQCFIN)
Expiry interval (parameter identifier: MQIA_EXPIRY_INTERVAL).

SSLTasks (MQCFIN)
SSL tasks (parameter identifier: MQIA_SSL_TASKS).

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

The maximum length of the string is MQ_SSL_KEY_REPOSITORY_LENGTH.

SSLCRLNameList (MQCFST)
SSL CRL name list (parameter identifier: MQCA_SSL_CRL_NAMELIST).

The maximum length of the string is MQ_NAMELIST_NAME_LENGTH.

ConfigurationEvent (MQCFIN)
Controls whether configuration events are generated (parameter identifier: MQIA_CONFIGURATION_EVENT).

QSGName (MQCFST)
Queue sharing group name (parameter identifier: MQCA_QSG_NAME).

The maximum length of the string is MQ_QSG_NAME_LENGTH.

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.

Event data for object attributes

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).

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.

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.

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

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

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.

ChannelAutoDefExit (MQCFST)
Channel auto-definition exit name (parameter identifier: MQCA_CHANNEL_AUTO_DEF_EXIT).

The maximum length of the exit name is MQ_EXIT_NAME_LENGTH.

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.

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

The time when the information was last altered.

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

The maximum length of the exit name is MQ_EXIT_NAME_LENGTH.

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

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

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

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.

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.

Storage class attributes

PageSetId (MQCFIN)
Page set identifier (parameter identifier: MQIA_PAGE_SET_ID).

StgClassDesc (MQCFST)
Storage class description (parameter identifier: MQCA_STORAGE_CLASS_DESC).

The maximum length of the string is MQ_STORAGE_CLASS_DESC_LENGTH.

XCFGroupName (MQCFST)
XCF group name (parameter identifier: MQCA_XCF_GROUP_NAME).

The maximum length of the string is MQ_XCF_GROUP_NAME_LENGTH.

XCFMemberName (MQCFST)
XCF member name (parameter identifier: MQCA_XCF_MEMBER_NAME).

The maximum length of the string is MQ_XCF_MEMBER_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.

Event data for object attributes

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.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,
Hampshire,
England
SO21 2JN.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

AIX	AS/400	CICS
IBM	IBMLink	IMS
iSeries	MQSeries	NetView
OS/2	OS/390	OS/400
S/390	System/390	WebSphere
z/OS		

ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Other company, product, and service names may be trademarks or service marks of others.

Index

M

N

O

P

Q

R

S

T

U

V

W

Z

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.

SC34-6069-02