WebSphere MQ

Monitoring WebSphere MQ

Version 7.0

SC34-6937-00

WebSphere MQ

Monitoring WebSphere MQ

Version 7.0

SC34-6937-00

First edition (April 2008)

This edition of the book applies to the following products:

- IBM WebSphere MQ, Version 7.0
- IBM WebSphere MQ for z/OS, Version 7.0

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

Contents

Figures

Tables

Chapter 1. Introduction

An introduction to monitoring WebSphere MQ

This section describes:

- "Monitoring WebSphere MQ"
- "Event monitoring" on page 2
- "Message monitoring" on page 2
- "Accounting and statistics messages" on page 3
- "Real-time monitoring" on page 3

Monitoring WebSphere MQ

There are many reasons for monitoring a queue manager network. Depending on the size and complexity of your queue manager network, you can benefit in different ways from the techniques available with WebSphere® MQ for monitoring your queue manager network. By applying monitoring techniques available with WebSphere MQ you can do the following:

- Detect problems in your queue manager network.
- Assist in determining the causes of problems in your queue manager network.
- Improve the efficiency of your queue manager network.
- Familiarize yourself with the running of your queue manager network.
- Confirm that your queue manager network is running correctly.
- Generate messages when certain events occur.
- Record message activity.
- Determine the last known location of a message.
- Check various statistics of a queue manager network in real time.
- Generate an audit trail.
- Account for application resource usage.
- Capacity planning.

Approaches to monitoring WebSphere MQ

There are various approaches to monitoring WebSphere MQ. Each approach is applied and used in a different way, and each approach returns monitoring information in a different form. Depending on how you intend to monitor your WebSphere MQ system you will use one, or a combination of, the following approaches:

- Event monitoring, see "Event monitoring" on page 2.
- Message monitoring, see "Message monitoring" on page 2.
- Accounting and statistics message, see "Accounting and statistics messages" on page 3.
- Real-time monitoring, see "Real-time monitoring" on page 3.

Event monitoring

Event monitoring is the process of detecting occurrences of *instrumentation events* in a queue manager network. An instrumentation event is a logical combination of events 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 the queue managers in your queue manager network, allowing you to do the following:

- Detect problems in your queue manager network.
- Assist in determining the causes of problems in your queue manager network.
- Generate an audit trail.
- React to queue manager state changes

For more information on event monitoring, see "An introduction to instrumentation events" on page 5.

Message monitoring

Message monitoring is the process of identifying the route a message has taken through a queue manager network. As a message passes through a queue manager network, various processes perform activities on behalf of the message. By identifying the types of activities, and the sequence of activities performed on behalf of a message, the message route can be determined.

The following techniques are available for determining a message route:

- The WebSphere MQ display route application (dspmqrte), see "WebSphere MQ display route application" on page 185.
- Trace-route messaging, see "Trace-route messaging" on page 169.
- Activity recording, see "Activity recording" on page 163.

These techniques all generate special messages that contain information about the activities performed on the message as it passed through a queue manager network. The information returned in these special messages can be used to do the following:

- Record message activity.
- Determine the last known location of a message.
- Detect routing problems in your queue manager network.
- Assist in determining the causes of routing problems in your queue manager network.
- Confirm that your queue manager network is running correctly.
- Familiarize yourself with the running of your queue manager network.
- Trace published messages.

Accounting and statistics messages

Accounting and statistics messages are generated intermittently by queue managers to record information about the MQI operations performed by WebSphere MQ applications, or to record information about the activities occurring in a WebSphere MQ system.

Accounting messages
Accounting messages are used to record information about the MQI operations performed by WebSphere MQ applications, see "Accounting messages" on page 245.

Statistics messages
Statistics messages are used to record information about the activities occurring in a WebSphere MQ system, see "Statistics messages" on page 249.

Accounting and statistics messages are delivered to one of two system queues. User applications can retrieve the messages from these system queues and use the recorded information to do the following:

- Account for application resource use.
- Record application activity.
- Capacity planning.
- Detect problems in your queue manager network.
- Assist in determining the causes of problems in your queue manager network.
- Improve the efficiency of your queue manager network.
- Familiarize yourself with the running of your queue manager network.
- Confirm that your queue manager network is running correctly.

For more information on accounting and statistics monitoring, see "Accounting and statistics messages" on page 245.

Real-time monitoring

Real-time monitoring is a technique that allows you to determine the current state of queues and channels within a queue manager. A number of commands are available that when issued return real-time information about queues and channels. Varying amounts of information can be returned for a single queue or channel, or for multiple queues or channels. The information returned is accurate at the moment the command was issued. Real-time monitoring can be used to:

- Help system administrators understand the steady state of their WebSphere MQ system. This helps with problem diagnosis if a problem occurs in the system.
- Determine the condition of your queue manager at any moment, even if no specific event or problem has been detected.
- Assist in determining the cause of a problem in your system.

When using real-time monitoring, information can be returned for either queues or channels. The amount of real-time information returned is controlled by queue manager, queue, and channel attributes. For more information see "An introduction to real-time monitoring" on page 297.

Real-time monitoring for queues and channels is in addition to, and separate from, performance and channel event monitoring.

Chapter 2. Event monitoring

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 6 illustrates the concept of instrumentation events.

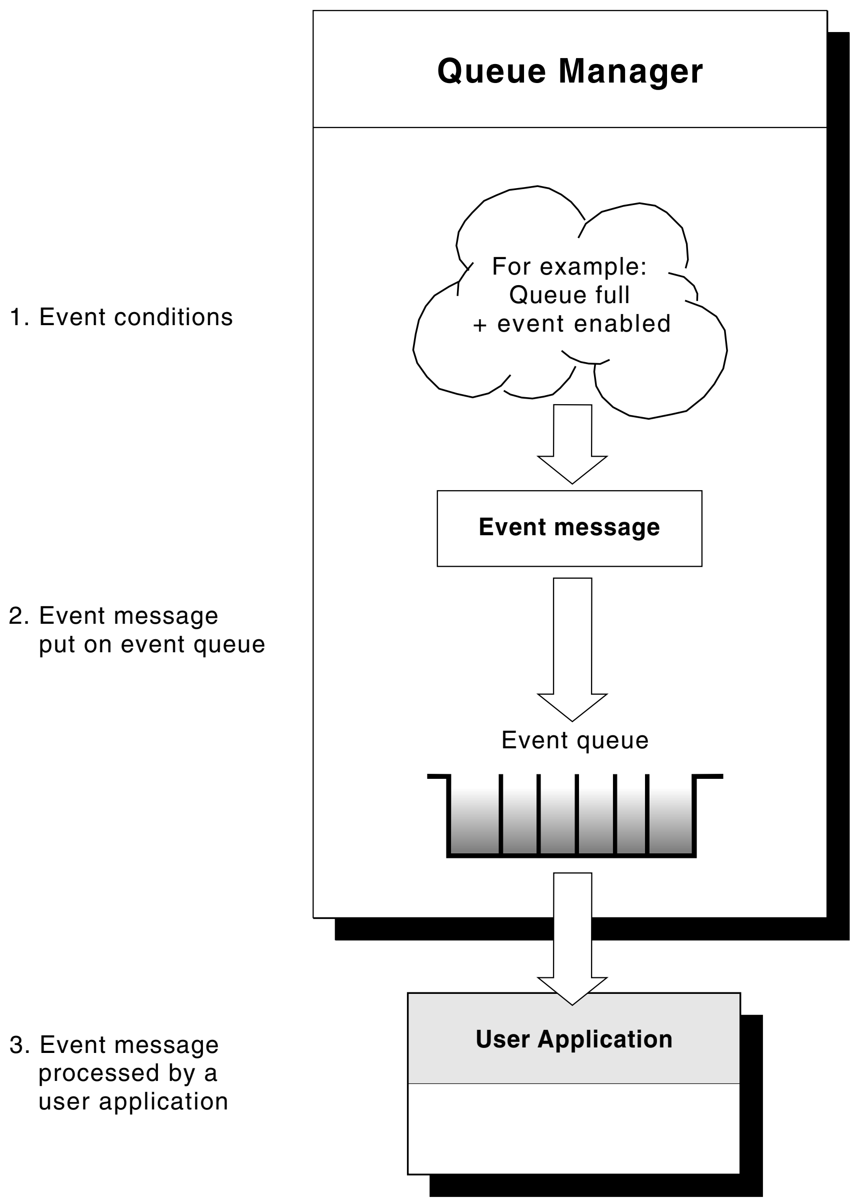

Queue Manager
For example: Queue full + event enabled
Event message
Event queue
User Application
1. Event conditions
2. Event message put on event queue
3. Event message processed by a user application

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 18. For detailed descriptions of the format of each event message, see "Event message format" on page 50.

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 i5/OS®, Linux®, Windows®, and UNIX® systems.
- Objects are created, deleted, changed or refreshed.
- An MQSC, or PCF, command that successfully executes.
- A queue manager starts writing to a new log extent.

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

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 and bridge 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 that some action has been performed on an object. They are generated automatically when an object is created, changed, or deleted, or when they are explicitly requested for. For example, when a namelist is created.

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

Command events
These events are notifications that a WebSphere MQ command has been performed successfully.

Command events are available with WebSphere MQ for z/OS only.

Logger events
These events are notifications that the queue manager has started writing to a new log extent. Logger events are only available on queue managers that use linear logging.

Logger events are not available 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
SYSTEM.ADMIN.COMMAND.EVENT	Command events
SYSTEM.ADMIN.LOGGER.EVENT	Logger 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.

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

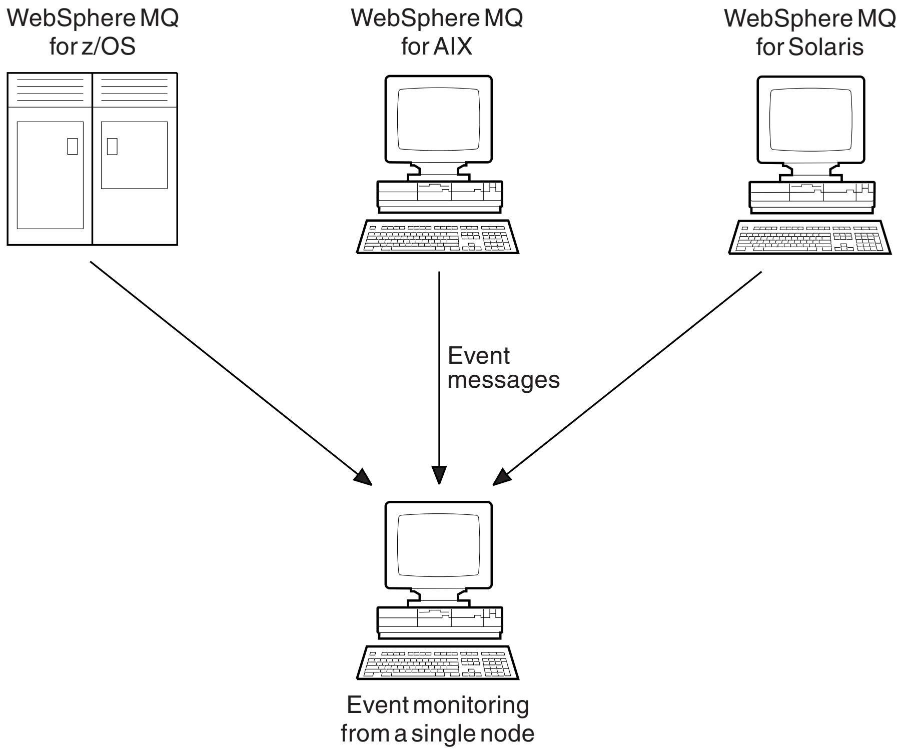


Figure 2. Monitoring queue managers across different platforms, on a single node

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

Logger events are not available on WebSphere MQ for z/OS.

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, HP OpenVMS, 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:

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.

All authority events are valid on HP OpenVMS, Windows, and UNIX systems only.

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

- "Not Authorized (type 1)" on page 106
- "Not Authorized (type 2)" on page 107
- "Not Authorized (type 3)" on page 109
- "Not Authorized (type 4)" on page 111
- "Not Authorized (type 5)" on page 112
- "Not Authorized (type 6)" on page 114

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, or against a topic where the topic is inhibited for publishes.

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

- "Get Inhibited" on page 103
- "Put Inhibited" on page 116

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 59
- "Unknown Alias Base Queue" on page 142
- "Unknown Object Name" on page 146

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 95
- "Default Transmission Queue Usage Error" on page 97
- "Queue Type Error" on page 130
- "Remote Queue Name Error" on page 136
- "Transmission Queue Type Error" on page 138
- "Transmission Queue Usage Error" on page 140
- "Unknown Default Transmission Queue" on page 144
- "Unknown Remote Queue Manager" on page 148
- "Unknown Transmission Queue" on page 150

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:
- "Queue Manager Active" on page 124
- "Queue Manager Not Active" on page 125

Channel and bridge events

Channel events are reported by channels as a result of conditions detected during their operation, such as when a channel instance is stopped. The event messages for channel and bridge events are put on the SYSTEM.ADMIN.CHANNEL.EVENT queue. Channel events are generated:
- By a command to start or stop a channel.
- When a channel instance starts or stops.
- When a channel receives a conversion error warning when getting a message.

- When an attempt is made to create a channel automatically; the event is generated whether the attempt succeeds or fails.

Note: Client connections 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 (providing they support events).

If a channel event is put on to 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 Activated" on page 68
- "Channel Auto-definition Error" on page 69
- "Channel Auto-definition OK" on page 71
- "Channel Conversion Error" on page 72
- "Channel Not Activated" on page 74
- "Channel Started" on page 79
- "Channel Stopped" on page 80
- "Channel Stopped By User" on page 83

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" on page 61
- "Bridge Stopped" on page 62

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 76

Performance events

These events report that a resource has reached a threshold condition. For example, a queue depth limit might have been reached. The event messages for performance events are put on the SYSTEM.ADMIN.PERFM.EVENT queue. For further details on performance events, see "Understanding performance events" on page 19.

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 events are 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

Configuration events are reported when objects are created, or modified. A configuration event message contains information about the attributes of an object. For example, a configuration event message is generated if a namelist object is created, and will contain information about the attributes of the namelist object. The event messages for configuration events are put on the SYSTEM.ADMIN.CONFIG.EVENT queue. For more information see "Understanding configuration events" on page 37.

Configuration events are available on WebSphere MQ for z/OS only.

There are four types of configuration event:

Create object events:

Create object events are generated when an object is created. For more information see "Create object" on page 91.

Change object events:

Change object events are generated when an object is changed. For more information see "Change object" on page 64.

Delete object events:

Delete object events are generated when an object is deleted. For more information see "Delete object" on page 99.

Refresh object events:

Refresh object events are generated by an explicit request to refresh. For more information see "Refresh object" on page 132.

Command events

A command event is reported when an MQSC or PCF command is executed successfully. For example, a command event message is generated if the MQSC command, ALTER QLOCAL, is executed successfully. A command event message contains information about the origin, context, and content of a command. The event messages for command events are put on the SYSTEM.ADMIN.COMMAND.EVENT queue. For more information, see "Understanding command events" on page 40.

Command events are available on WebSphere MQ for z/OS only.

Logger events

A logger event is reported when a queue manager, that employs linear logging, starts writing log records to a new log extent, or on i5/OS, a new journal receiver. A logger event message contains information specifying the log extents required by the queue manager for queue manager restart, or media recovery. The event messages for logger events are put on the SYSTEM.ADMIN.LOGGER.EVENT queue. For more information, see "Understanding logger events" on page 42.

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	Refer to...
Authority events	Not Authorized (type 1)	"Not Authorized (type 1)" on page 106
	Not Authorized (type 2)	"Not Authorized (type 2)" on page 107
	Not Authorized (type 3)	"Not Authorized (type 3)" on page 109
	Not Authorized (type 4)	"Not Authorized (type 4)" on page 111
	Not Authorized (type 5)	"Not Authorized (type 5)" on page 112
	Not Authorized (type 6)	"Not Authorized (type 6)" on page 114
Channel events	Channel Activated	"Channel Activated" on page 68
	Channel Auto-definition Error	"Channel Auto-definition Error" on page 69
	Channel Auto-definition OK	"Channel Auto-definition OK" on page 71
	Channel Conversion Error	"Channel Conversion Error" on page 72
	Channel Not Activated	"Channel Not Activated" on page 74
	Channel Started	"Channel Started" on page 79
	Channel Stopped	"Channel Stopped" on page 80
	Channel Stopped By User	"Channel Stopped By User" on page 83
Command events	Command	"Command" on page 84

Table 1. Event message data summary (continued)

Event type	Event name	Refer to...
Configuration events	Create object	"Create object" on page 91
	Change object	"Change object" on page 64
	Delete object	"Delete object" on page 99
	Refresh object	"Refresh object" on page 132
IMS Bridge events	Bridge started	"Bridge Started" on page 61
	Bridge stopped	"Bridge Stopped" on page 62
Inhibit events	Get inhibited	"Get Inhibited" on page 103
	Put inhibited	"Put Inhibited" on page 116
Local events	Alias base queue type error	"Alias Base Queue Type Error" on page 59
	Unknown alias base queue	"Unknown Alias Base Queue" on page 142
	Unknown object name	"Unknown Object Name" on page 146
Logger events	Logger	"Command" on page 84
Performance events	Queue Depth High	"Queue Depth High" on page 118
	Queue Depth Low	"Queue Depth Low" on page 120
	Queue Full	"Queue Full" on page 122
	Queue Service Interval High	"Queue Service Interval High" on page 126
	Queue Service Interval OK	"Queue Service Interval OK" on page 128
Remote events	Default Transmission Queue Type Error	"Default Transmission Queue Type Error" on page 95
	Default Transmission Queue Usage Error	"Default Transmission Queue Usage Error" on page 97
	Queue Type Error	"Queue Type Error" on page 130
	Remote Queue Name Error	"Remote Queue Name Error" on page 136
	Transmission Queue Type Error	"Transmission Queue Type Error" on page 138
	Transmission Queue Usage Error	"Transmission Queue Usage Error" on page 140
	Unknown Default Transmission Queue	"Unknown Default Transmission Queue" on page 144
	Unknown Remote Queue Manager	"Unknown Remote Queue Manager" on page 148
	Unknown Transmission Queue	"Unknown Transmission Queue" on page 150
SSL events	Channel SSL Error	"Channel SSL Error" on page 76
Start and stop events	Queue Manager Active	"Queue Manager Active" on page 124
	Queue Manager Not Active	"Queue Manager Not Active" on page 125

Controlling 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.
- The WebSphere MQ Explorer. For more information, see the WebSphere MQ System Administration Guide.

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

Controlling queue manager events

Queue manager events are controlled using queue manager attributes. To enable queue manager events, set the appropriate queue manager attribute to ENABLED. To disable queue manager events, set the appropriate queue manager attribute to DISABLED. To enable or disable queue manager events you can use the MQSC command ALTER QMGR specifying the appropriate queue manager attribute, as follows:

Authority events
Set the queue manager attribute AUTHOREV.

Inhibit events
Set the queue manager attribute INHIBITEV.

Local events
Set the queue manager attribute LOCALEV.

Remote events
Set the queue manager attribute REMOTEEV.

Start and stop events
Set the queue manager attribute STRSTPEV.

Enabling queue manager events summary:

Table 2 summarizes how to enable queue manager events:

Table 2. Enabling queue manager events using MQSC commands

Event	*ALTER QMGR parameter*
Authority	AUTHOREV (ENABLED)
Inhibit	INHIBTEV (ENABLED)
Local	LOCALEV (ENABLED)
Remote	REMOTEEV (ENABLED)
Start and Stop	STRSTPEV (ENABLED)

Controlling channel and bridge events

Channel events are controlled using queue manager attributes. To enable channel events, set the appropriate queue manager attribute to ENABLED. To disable channel events, set the appropriate queue manager attribute to DISABLED. To enable or disable channels events you can use the MQSC command ALTER QMGR specifying the appropriate queue manager attribute, as follows:

Channel events
: Set the queue manager attribute CHLEV.

  To specify that only the events related to channel errors are to be recorded, set CHLEV to EXCEPTION.

IMS Bridge events
: Set the queue manager attribute BRIDGEEV.

SSL events
: Set the queue manager attribute SSLEV.

Channel auto-definition events
: Set the queue manager attribute CHADEV.

  Channel auto-definition events are not available on WebSphere MQ for z/OS.

Enabling channel and bridge events summary:

Table 3 summarizes how to enable channel and bridge events:

Table 3. Enabling channel and bridge events using MQSC commands

Event	*ALTER QMGR parameter*
Channel	CHLEV (ENABLED)
Channel errors only	CHLEV (EXCEPTION)
IMS Bridge	BRIDGEEV (ENABLED)
SSL	SSLEV (ENABLED)
Channel auto-definition	CHADEV(ENABLED)

Controlling performance events

Performance events as a whole are controlled using the PERFMEV queue manager attribute. To enable performance events, set PERFMEV to ENABLED. To disable performance events, set the PERFMEV queue manager attribute to DISABLED. For example, to set the PERFMEV queue manager attribute to ENABLED, you can use the following MQSC command:

```
ALTER QMGR PERFMEV (ENABLED)
```

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.

Controlling 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 "Understanding queue depth events" on page 29.

Controlling 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 QSVCINT attribute for the queue to the appropriate length of time.

For more information, see "Enabling queue service interval events" on page 23.

Note: When enabled, a queue service interval event can be generated at any appropriate time, not necessarily waiting until an MQI call for the queue is issued. However, if an MQI call is used on a queue to put or remove a message, any applicable performance event will be generated at that time. The event is **not** generated when the elapsed time becomes equal to the service interval time.

Controlling configuration events

Configuration events as a whole are controlled using the CONFIGEV queue manager attribute. To enable configuration events, set CONFIGEV to ENABLED. To disable configuration events, set CONFIGEV to DISABLED. For example, you can enable configuration events by using the following MQSC command:

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

Controlling command events

Command events as a whole are controlled using the CMDEV queue manager attribute. To enable command events, set CMDEV to ENABLED. To enable command events for commands except `DISPLAY` MQSC commands and `Inquire` PCF commands, set the CMDEV to NODISPLAY. To disable command events, set CMDEV to DISABLED. For example, you can enable command events by using the following MQSC command:

```
ALTER QMGR CMDEV (ENABLED)
```

Controlling logger events

Logger events as a whole are controlled using the LOGGEREV queue manager attribute. To enable logger events, set LOGGEREV to ENABLED. To disable logger events, set LOGGEREV to DISABLED. For example, you can enable logger events by using the following MQSC command:

```
ALTER QMGR LOGGEREV(ENABLED)
```

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.

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.

In the case of configuration and command events the same is true again. In the absence of an event queue, an event message is **not** generated, however the command or call is executed regardless.

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 51.

Using event monitoring in a 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.

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:

- "What performance events are"
- "Understanding queue service interval events" on page 20
- "Queue service interval events examples" on page 24
- "Understanding queue depth events" on page 29
- "Queue depth events examples" on page 34

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.

Performance event messages can be generated at any appropriate time, not necessarily waiting until an MQI call for the queue is issued. However, if an MQI call is used on a queue to put or remove a message, any appropriate performance events are generated at that time.

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 118
- "Queue Depth Low" on page 120
- "Queue Full" on page 122
- "Queue Service Interval High" on page 126
- "Queue Service Interval OK" on page 128

Performance event statistics

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

Table 4. Performance event statistics

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

Performance event statistics are reset when any 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 an application 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.

Throughout this section where the term *get operation* is used, it refers to an MQGET call or an activity that removes a messages from a queue, such as using the CLEAR QLOCAL command.

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
   - A get operation that leaves a non-empty queue

   a get operation 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 a get operation.

   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
   - A get operation that leaves a non-empty queue

   a get operation was **not** performed within a user-defined service interval.

   The Queue Service Interval High event message can be caused by a get operation or an 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.

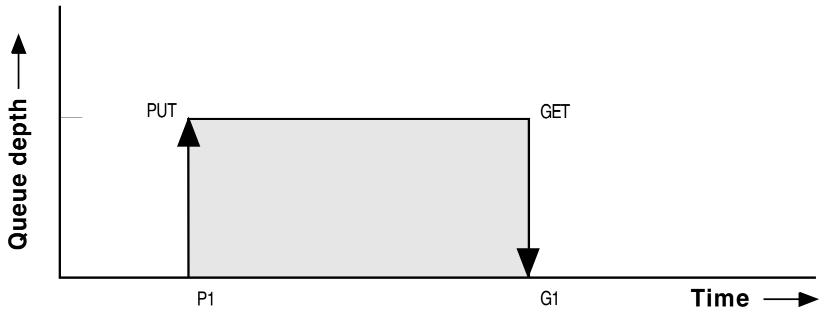


Figure 3. Understanding queue service interval events

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."

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 a get operation, 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.

When is the service timer reset?
The service timer is always reset after a get operation . 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 a get operation 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 there is a get operation 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 a get operation, 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, or before, the next MQPUT or get operation.

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, or before, the next get operation.

Enabling queue service interval events

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

1. Enable performance events on the queue manager, by setting the queue manager attribute PERFMEV to ENABLED.
2. Set the control attribute, QSVCIEV, for a Queue Service Interval High or OK event on the queue, as required.
3. Specify the service interval time by setting the QSVCINT attribute for the queue to the appropriate length of time.

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)
```

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.

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

Table 5. 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.

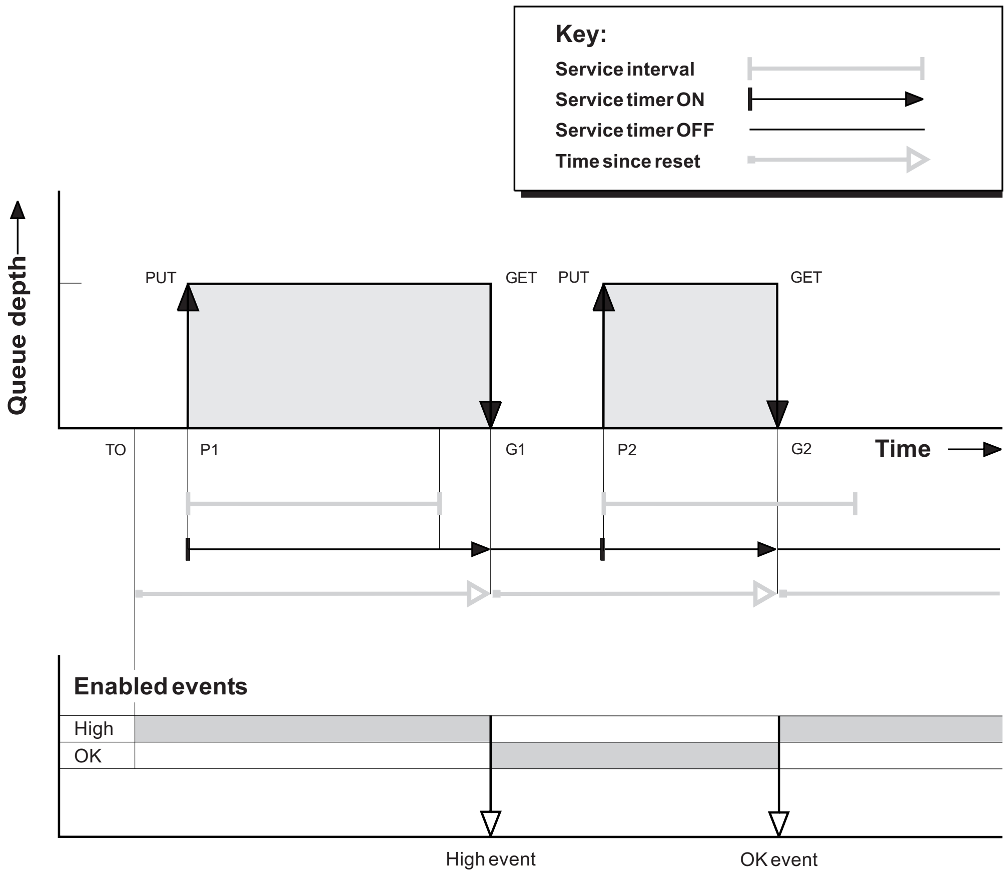


Figure 4. Queue service interval events - example 1

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 6 summarizes the event statistics for this example.

Table 6. 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 25 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 25 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)

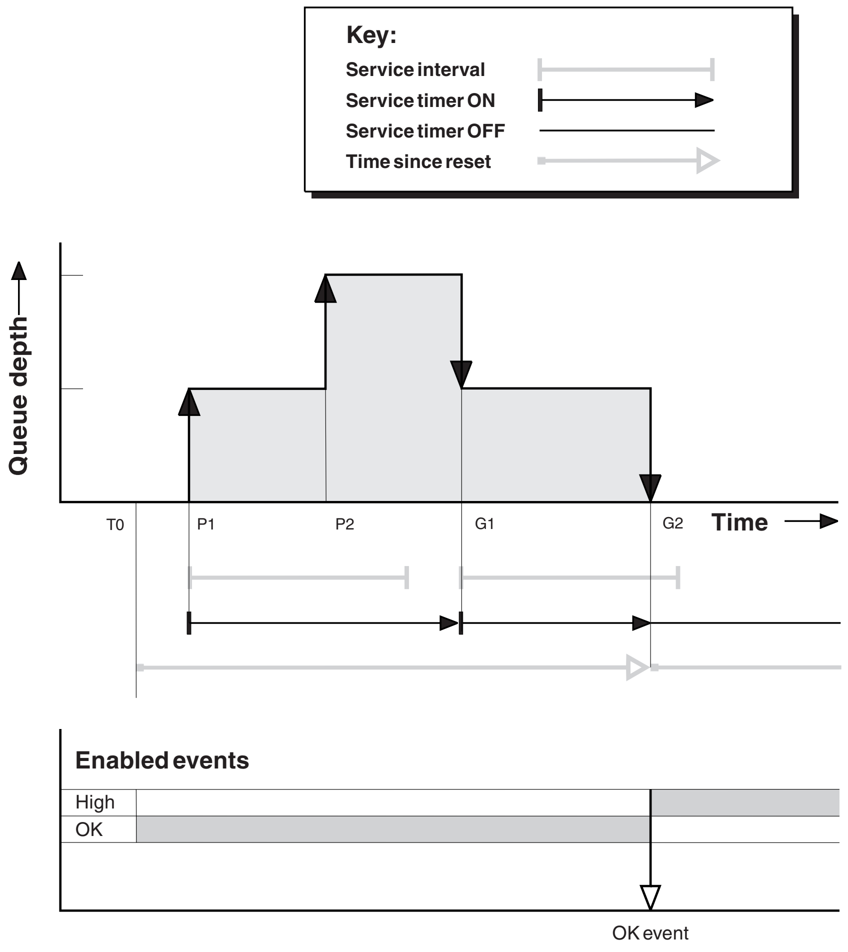


Figure 5. Queue service interval events - example 2

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 7 summarizes the event statistics for this example.

Table 7. Event statistics summary for example 2

Time of event	T(G2)

Table 7. Event statistics summary for example 2 (continued)

Type of event	OK
TimeSinceReset	T(G2) - T(0)
HighQDepth	2
MsgEnqCount	2
MsgDeqCount	2

Example 3 (queue service interval events)

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

Commentary:

1. At time T(0), the queue statistics are reset and Queue Service Interval High events are enabled.
2. At P1, the first put starts the service timer.
3. At P2, the second put increases the queue depth to two. A high event is not generated here because the service interval time has not been exceeded.
4. At P3, the third put causes a high event to be generated. (The timer has exceeded the service interval.) The timer is not reset because the queue depth was not zero before the put. However, OK events are enabled.
5. At G1, the MQGET call does not generate an event because the service interval has been exceeded and OK events are enabled. The MQGET call does, however, reset the service timer.
6. At G2, the MQGET call does not generate an event because the service interval has been exceeded and OK events are enabled. Again, the MQGET call resets the service timer.
7. At G3, the third get empties the queue and the service timer is *equal* to the service interval. Therefore an OK event is generated. The service timer is reset and high events are enabled. The MQGET call empties the queue, and this puts the timer in the OFF state.

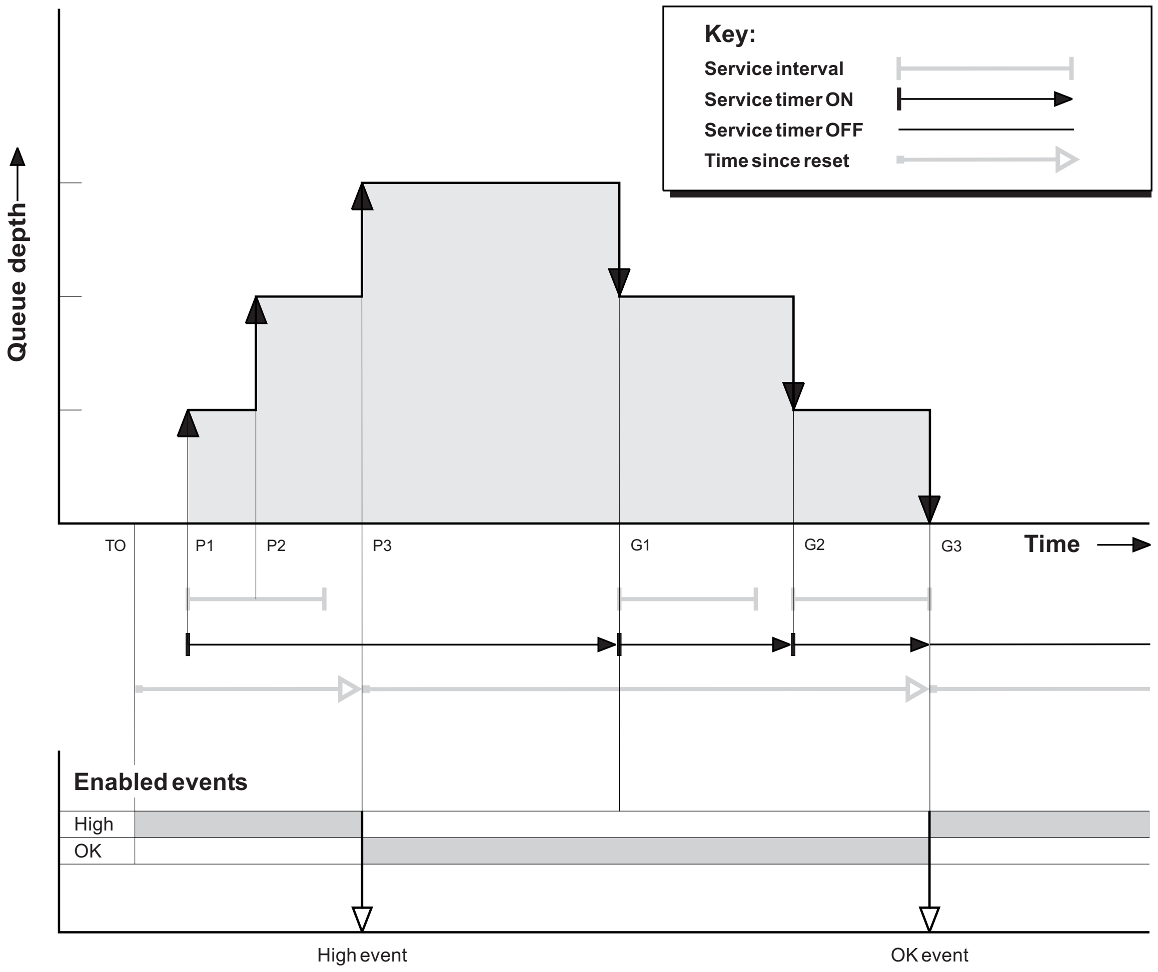


Figure 6. Queue service interval events - example 3

Event statistics summary for example 3:

Table 8 summarizes the event statistics for this example.

Table 8. 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 7 on page 34 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 PERFMEV 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 PERFMEV 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 PERFMEV.
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*.

If enabled for performance events, the **coordinating queue manager** does the following:

1. 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.
2. 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.

Having received a request from the coordinating queue manager, if there is a performance event enabled **active queue manager in the queue-sharing group**, it issues an event message for the shared queue. The issued event message contains information about all the 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 them on nonshared queues. A key difference is that events are switched on 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 a get 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 9. 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)

Note: 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)
```

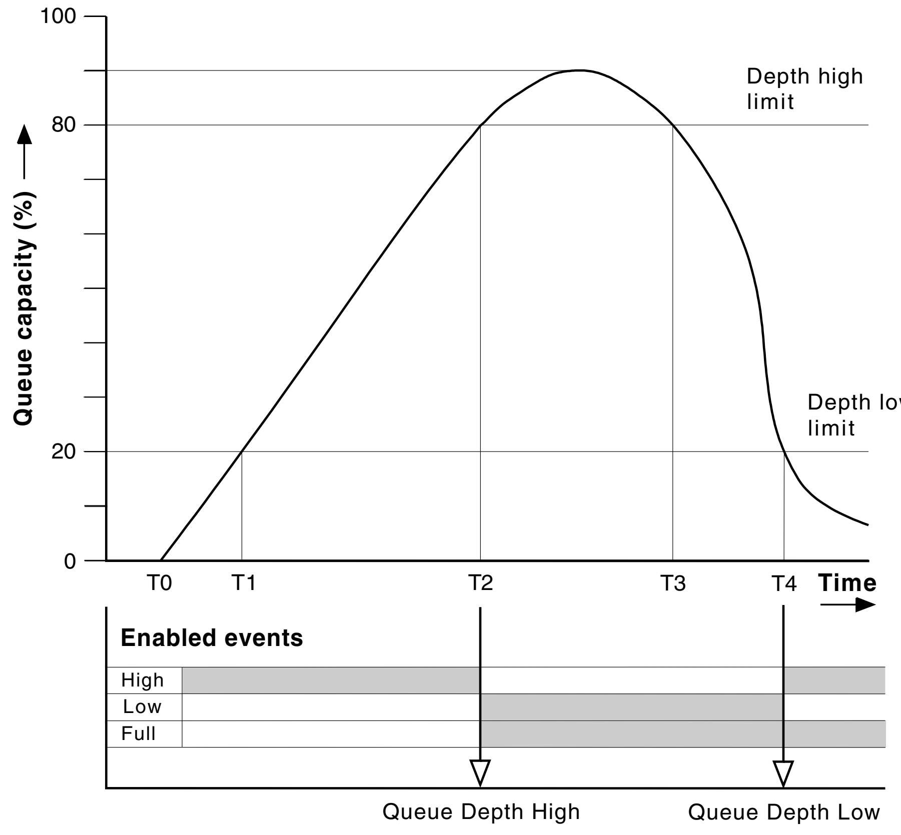


Figure 7. Queue depth events (1)

Commentary:

Figure 7 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 10 summarizes the queue event statistics and Table 11 summarizes which events are

Table 10. 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 11. 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 Definition of MYQUEUE1.

Table 12 on page 36 summarizes the queue event statistics and Table 13 on page 37 summarizes which events are enabled at different times for this example.

Figure 8 on page 36 shows the variation of queue depth over time.

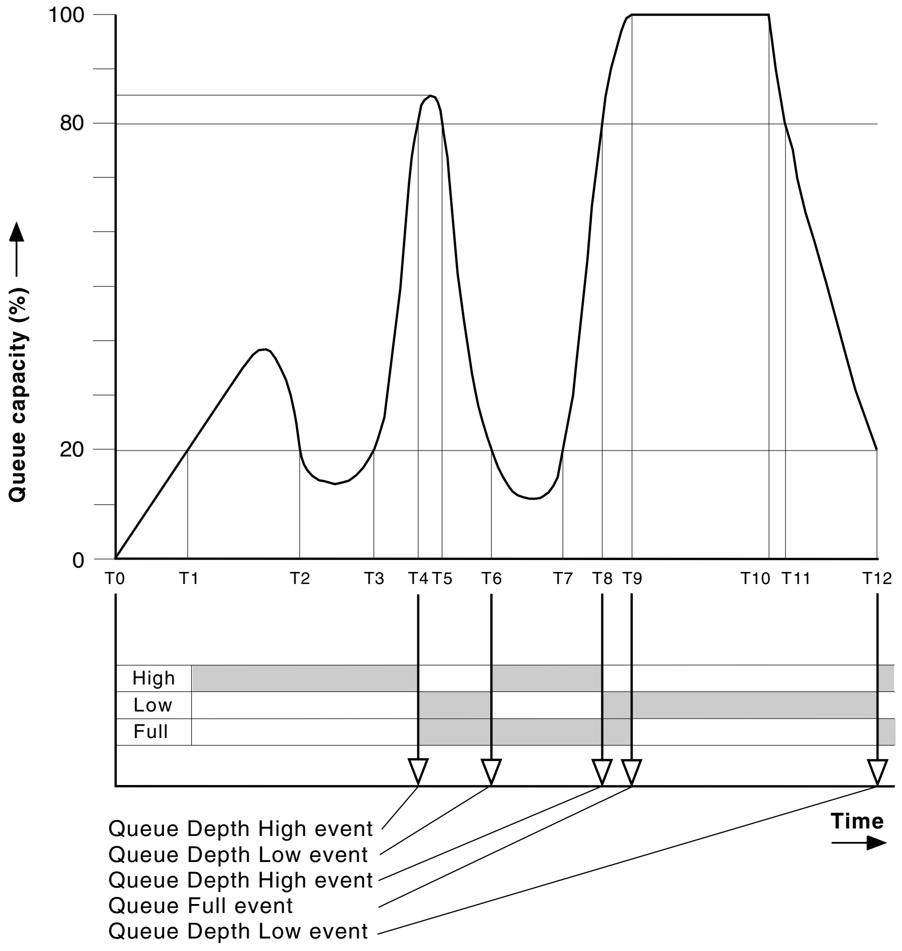


Figure 8. Queue depth events (2)

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 12. 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

Table 12. Event statistics summary for queue depth events (example 2) (continued)

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 13. 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.

Understanding configuration events

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
- How configuration events are used

Configuration event messages are only available with WebSphere MQ for z/OS.

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: "Event message reference" on page 48.

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, or their PCF equivalent, 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, or their PCF equivalent, 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, or their PCF equivalent, 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 call 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:
- Commands 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.
- REFRESH/RESET CLUSTER and RESUME/SUSPEND QMGR commands that cause clustering changes.
- Creating or deleting a queue manager.

How configuration events are used

Configuration events can be used for the following 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.

   This can be particularly useful when command events are also enabled. If an MQSC or PCF command causes a configuration event and a command event to be generated, both event messages will share the same correlation identifier in their message descriptor.

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 is executed regardless.

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 MQSC command, REFRESH QMGR, or its PCF equivalent.

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.

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.

Understanding command events

This collection of topics describes what command events are, when they are generated, when they are not generated, and how they are used.

Command event messages are only available with WebSphere MQ for z/OS.

What command events are

Command events are notifications that an MQSC, or PCF command has been executed successfully.

The event data of a command event message contains the following information:

- **Origin information**, describes the queue manager from where the command was executed, the ID of the user that executed the command, and how the command was executed, for example by a console command.
- **Context information**, is a replica of the context information in the message data from the command message. If a command is not entered using a message, context information is omitted.

  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.
- **Command information**, describes the type of command that was executed.
- **Command data**, for PCF commands is a replica of the command data, or for MQSC commands is the command text.

Every command event message that is generated is placed on the command event queue, SYSTEM.ADMIN.COMMAND.EVENT.

For more information about the event data returned in command event messages see: "Event message reference" on page 48.

When command events are generated

A command event message is generated for the following:

- When the queue manager attribute, CMDEV, is specified as ENABLED and an MQSC or PCF command is successfully executed.
- When the queue manager attribute, CMDEV, is specified as NODISPLAY and any command is successfully executed, with the exception of DISPLAY commands (MQSC), and Inquire commands (PCF).
- When the MQSC command, ALTER QMGR, or the PCF command, Change Queue Manager, is executed and either:
  - the queue manager attribute, CMDEV, after the change is not specified as DISABLED.
  - the queue manager attribute, CMDEV, before the change was not specified as DISABLED.

If a command is executed against the command event queue, SYSTEM.ADMIN.COMMAND.EVENT, a command event is generated providing the queue still exists and it is not put inhibited.

When command events are not generated

A command event message is not generated for:

- A command that fails.
- A queue manager that encounters an error trying to put a command event on the event queue. In this situation, the command executes regardless, but no event message is generated.
- The MQSC command REFRESH QMGR TYPE (EARLY).
- The MQSC command START QMGR MQSC.
- The MQSC command SUSPEND QMGR, if the parameter LOG is specified.
- The MQSC command RESUME QMGR, if the parameter LOG is specified.

How command events are used

Command event messages can be used 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. This can be particularly useful when configuration events are also enabled. If an MQSC or PCF command causes a command event and a configuration event to be generated, both event messages will share the same correlation identifier in their message descriptor.

If a command event message is generated, but cannot be put on the command event queue, for example the if the command event queue has not been defined, then the command for which the command event was generated is still executed regardless.

Effects of CMDSCOPE

For commands where CMDSCOPE is used, the command 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.

Understanding logger events

This collection of topics describes what logger events are, when they are generated, when they are not generated, and how they are used.

Logger event messages are not available with WebSphere MQ for z/OS.

What logger events are

Logger events are notifications that a queue manager has started writing to a new log extent, or on i5/OS a journal receiver.

The event data of a logger event message contains the following:
- The name of the current log extent.
- The name of the earliest log extent needed for restart recovery.
- The name of the earliest log extent needed for media recovery.
- The directory in which the log extents are located.

For information on restart and media recovery, see the WebSphere MQ System Administration Guide.

Every logger event message that is generated is placed on the logger event queue, SYSTEM.ADMIN.LOGGER.EVENT.

For more information about the event data returned in logger event messages see: “Event message reference” on page 48.

When logger events are generated

A logger event message is generated for the following:

- When the queue manager attribute, LOGGEREV, is specified as ENABLED and the queue manager starts writing to a new log extent, or on i5/OS a journal receiver.
- When the queue manager attribute, LOGGEREV, is specified as ENABLED and the queue manager starts.
- When the queue manager attribute, LOGGEREV, is changed from DISABLED to ENABLED.

Note: The MQSC command, RESET QMGR, can be used to request that a queue manager start writing to a new log extent.

When logger events are not generated

A logger event message is not generated when:

- A queue manager is configured to use circular logging.

  In this case, the queue manager attribute, LOGGEREV, is set as DISABLED and cannot be altered.
- A queue manager that encounters an error trying to put a logger event on the event queue. In this situation, the action that caused the event completes, but no event message is generated.

How logger events are used

Logger event message can be used to determine the log extents that are no longer required for queue manager restart, or media recovery. Superfluous log extents can be archived to a medium such as tape for disaster recovery, then removed from the active log directory. Regular removal of superfluous log extents keeps disk space usage to a minimum.

If the queue manager attribute LOGGEREV is enabled, but a logger event message cannot be put on the logger event queue, for example the event queue has not been defined, the action that caused the event continues regardless.

Monitor the logger event queue (amqslog0.c)

```
/******************************************************************************/
/*                                                                            */
/* Program name: AMQSLOG0.C                                                   */
/*                                                                            */
/* Description:  Sample C program to monitor the logger event queue and output*/
/*               a message to stdout when a logger event occurs               */
/* &lt;N_OCO_COPYRIGHT>                                                         */
/* Licensed Materials - Property of IBM                                       */
/*                                                                            */
/* 63H9336                                                                    */
/* (c) Copyright IBM Corp. 2005 All Rights Reserved.                          */
/*                                                                            */
/* US Government Users Restricted Rights - Use, duplication or                */
/* disclosure restricted by GSA ADP Schedule Contract with                    */
/* IBM Corp.                                                                  */
/* <NOC_COPYRIGHT>                                                            */
/******************************************************************************/
/*                                                                            */
/* Function: AMQSLOG is a sample program which monitors the logger event      */
/* queue for new event messages, reads those messages, and puts the contents  */
/* of the message to stdout.                                                  */
/*                                                                            */
/******************************************************************************/
```

```
/*                                                                          */
/* AMQSLOG has 1 parameter - the queue manager name (optional, if not       */
/* specified then the default queue manager is implied)                     */
/*                                                                          */
/****************************************************************************/

/****************************************************************************/
/* Includes                                                                 */
/****************************************************************************/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include <cmqc.h>        /* MQI constants*/
#include <cmqcfc.h>      /* PCF constants*/

/****************************************************************************/
/* Constants                                                                */
/****************************************************************************/

#define   MAX_MESSAGE_LENGTH   8000

typedef struct _ParmTableEntry
{
  MQLONG  ConstVal;
  PMQCHAR Desc;
} ParmTableEntry;

ParmTableEntry ParmTable[] =
{
  0                              ,"",
  MQCA_Q_MGR_NAME                ,"Queue Manager Name",
  MQCMD_LOGGER_EVENT             ,"Logger Event Command",
  MQRC_LOGGER_STATUS             ,"Logger Status",
  MQCACF_CURRENT_LOG_EXTENT_NAME,"Current Log Extent",
  MQCACF_RESTART_LOG_EXTENT_NAME,"Restart Log Extent",
  MQCACF_MEDIA_LOG_EXTENT_NAME  ,"Media Log Extent",
  MQCACF_LOG_PATH                ,"Log Path"};

/****************************************************************************/
/* Function prototypes                                                      */
/****************************************************************************/

static void ProcessPCF(MQHCONN    hConn,
                       MQHOBJ     hEventQueue,
                       PMQCHAR    pBuffer);

static PMQCHAR ParmToString(MQLONG Parameter);

/********************************************************************/
/* Function: main                                                   */
/********************************************************************/
int main(int argc, char * argv[])
{
  MQLONG    CompCode;
  MQLONG    Reason;
  MQHCONN   hConn = MQHC_UNUSABLE_HCONN;
  MQOD      ObjDesc = { MQOD_DEFAULT };
  MQCHAR    QMName[MQ_Q_MGR_NAME_LENGTH+1]    = "";
  MQCHAR    LogEvQ[MQ_Q_NAME_LENGTH]        = "SYSTEM.ADMIN.LOGGER.EVENT";
  MQHOBJ    hEventQueue;
  PMQCHAR   pBuffer = NULL;

  printf("\n/*************************************/\n");
  printf("/* Sample Logger Event Monitor start */\n");
  printf("/*************************************/\n");
```

```
/********************************************************************/
/* Parse any command line options                                   */
/********************************************************************/

if (argc > 1)
   strncpy(QMName, argv[1], (size_t)MQ_Q_MGR_NAME_LENGTH);

pBuffer = (char *)malloc(MAX_MESSAGE_LENGTH);
if (!pBuffer)
{
  printf("Can't allocate %d bytes\n",MAX_MESSAGE_LENGTH);
  goto MOD_EXIT;
}

/********************************************************************/
/* Connect to the specified (or default) queue manager              */
/********************************************************************/

MQCONN(QMName,
       &hConn;,
       &CompCode;,
       &Reason;);

if (Reason != MQCC_OK)
{
  printf("Error in call to MQCONN, Reason %d, CompCode %d\n", Reason,
  CompCode);
  goto MOD_EXIT;
}

/* Open the logger event queue for input  */

strncpy(ObjDesc.ObjectQMgrName,QMName, MQ_Q_MGR_NAME_LENGTH);
strncpy(ObjDesc.ObjectName    ,LogEvQ, MQ_Q_NAME_LENGTH);

MQOPEN(  hConn,
        &ObjDesc;,
         MQOO_INPUT_EXCLUSIVE,
        &hEventQueue;,
        &CompCode;,
        &Reason; );
if (Reason)
{
  printf("MQOPEN failed for queue manager %.48s Queue %.48s Reason: %d\n",
                                ObjDesc.ObjectQMgrName,
                                    ObjDesc.ObjectName,
                                    Reason);
  goto MOD_EXIT;
}
else
{
  ProcessPCF(hConn, hEventQueue, pBuffer);
}

MOD_EXIT:

if (pBuffer != NULL) {
  free(pBuffer);
}

/********************************************************************/
/* Disconnect                                                       */
/********************************************************************/
if (hConn != MQHC_UNUSABLE_HCONN) {
   MQDISC(&hConn;, &CompCode;, &Reason; );
}
```

```
  return 0;
}

/********************************************************************************/
/* Function: ProcessPCF                                                          */
/********************************************************************************/
/*                                                                              */
/* Input Parameters:  Handle to queue manager connection                        */
/*                    Handle to the opened logger event queue object            */
/*                    Pointer to a memory buffer to store the incoming PCF msg*/
/*                                                                              */
/* Output Parameters: None                                                      */
/*                                                                              */
/* Logic: Wait for messages to appear on the logger event queue and display   */
/* their contents.                                                              */
/*                                                                              */
/********************************************************************************/

static void ProcessPCF(MQHCONN    hConn,
                       MQHOBJ     hEventQueue,
                       PMQCHAR    pBuffer)
{
  MQCFH   * pCfh;
  MQCFST  * pCfst;
  MQGMO     Gmo     = { MQGMO_DEFAULT };
  MQMD      Mqmd    = { MQMD_DEFAULT };
  PMQCHAR   pPCFCmd;
  MQLONG    Reason  = 0;
  MQLONG    CompCode;
  MQLONG    MsgLen;
  PMQCHAR   Parm = NULL;
                                         /* Set timeout value           */
  Gmo.Options     |= MQGMO_WAIT;
  Gmo.Options |= MQGMO_CONVERT;
  Gmo.WaitInterval = MQWI_UNLIMITED;
  /*****************************************************************/
  /* Process response Queue                                          */
  /*****************************************************************/
  while (Reason == MQCC_OK)
  {
    memcpy(&Mqmd.MsgId;   , MQMI_NONE, sizeof(Mqmd.MsgId));
    memset(&Mqmd.CorrelId;, 0, sizeof(Mqmd.CorrelId));

    MQGET( hConn,
           hEventQueue,
          &Mqmd;,
          &Gmo;,
           MAX_MESSAGE_LENGTH,
           pBuffer,
          &MsgLen;,
          &CompCode;,
          &Reason; );
    if (Reason != MQCC_OK)
    {
      switch(Reason)
      {
        case MQRC_NO_MSG_AVAILABLE:
             printf("Timed out");
             break;

        default:
             printf("MQGET failed RC(%d)\n", Reason);
             break;
      }
      goto MOD_EXIT;
    }
```

```
/******************************************************************/
/* Only expect PCF event messages on this queue                  */
/******************************************************************/
if (memcmp(Mqmd.Format, MQFMT_EVENT, sizeof(Mqmd.Format)))
{
 printf("Unexpected message format '%8.8s' received\n",Mqmd.Format);
 continue;
}


/*******************************************************************/
/* Build the output by parsing the received PCF message, first the */
/* header, then each of the parameters                             */
/*******************************************************************/

pCfh = (MQCFH *)pBuffer;

if (pCfh -> Reason)
{
 printf("-------------------------------------------------------------------\n");
 printf("Event Message Received\n");

 Parm = ParmToString(pCfh->Command);
 if (Parm != NULL) {
   printf("Command  :%s \n",Parm);
 }
 else
 {
   printf("Command  :%d \n",pCfh->Command);
 }

 printf("CompCode :%d\n"   ,pCfh->CompCode);

 Parm = ParmToString(pCfh->Reason);
 if (Parm != NULL) {
   printf("Reason   :%s \n",Parm);
 }
 else
 {
   printf("Reason   :%d \n",pCfh->Reason);
 }
}

pPCFCmd  = (char *)  (pCfh+1);
printf("-------------------------------------------------------------------\n");
while(pCfh -> ParameterCount--)
{
  pCfst = (MQCFST *) pPCFCmd;
  switch(pCfst -> Type)
  {
    case MQCFT_STRING:
         Parm = ParmToString(pCfst -> Parameter);
         if (Parm != NULL) {
           printf("%-32s",Parm);
         }
         else
         {
           printf("%-32d",pCfst -> Parameter);
         }

         fwrite( pCfst -> String, pCfst -> StringLength, 1, stdout);
         pPCFCmd += pCfst -> StrucLength;
         break;
    default:
         printf("Unrecoginised datatype %d returned\n",pCfst->Type);
         goto MOD_EXIT;
  }
```

```
      putchar('\n');
    }
    printf("-----------------------------------------------------------------\n");
  }
MOD_EXIT:

 return;
}

/********************************************************************************/
/* Function: ParmToString                                                      */
/********************************************************************************/
/*                                                                              */
/* Input Parameters:  Parameter for which to get string description            */
/*                                                                              */
/* Output Parameters: None                                                      */
/*                                                                              */
/* Logic: Takes a parameter as input and returns a pointer to a string         */
/* description for that parameter, or NULL if the parameter does not  */
/* have an associated string description                                       */
/********************************************************************************/

static PMQCHAR ParmToString(MQLONG Parameter){
  long i;
  for (i=0 ; i< sizeof(ParmTable)/sizeof(ParmTableEntry); i++)
  {
    if (ParmTable[i].ConstVal == Parameter &amp;&amp; ParmTable[i].Desc)
      return ParmTable[i].Desc;
  }
  return NULL;
}
</cmqcfc.h></cmqc.h></string.h></stdio.h></stdlib.h></NOC_COPYRIGHT>
</N_OCO_COPYRIGHT>
```

Sample output

This application produces the following form of output:

```
/**************************************/
/* Sample Logger Event Monitor start */
/**************************************/
-----------------------------------------------------------------
Event Message Received
Command  :Logger Event Command
CompCode :0
Reason   :Logger Status
-----------------------------------------------------------------
Queue Manager Name              CSIM

Current Log Extent              AMQA000001
Restart Log Extent              AMQA000001
Media Log Extent                AMQA000001
Log Path                        QMCSIM
-----------------------------------------------------------------
```

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" on page 50
- "Event message MQMD (message descriptor)" on page 51

- "Event message MQCFH (PCF header)" on page 56
- "Alias Base Queue Type Error" on page 59
- "Bridge Started" on page 61
- "Bridge Stopped" on page 62
- "Change object" on page 64
- "Channel Activated" on page 68
- "Channel Auto-definition Error" on page 69
- "Channel Auto-definition OK" on page 71
- "Channel Conversion Error" on page 72
- "Channel Not Activated" on page 74
- "Channel SSL Error" on page 76
- "Channel Started" on page 79
- "Channel Stopped" on page 80
- "Channel Stopped By User" on page 83
- "Command" on page 84
- "Create object" on page 91
- "Default Transmission Queue Type Error" on page 95
- "Default Transmission Queue Usage Error" on page 97
- "Delete object" on page 99
- "Get Inhibited" on page 103
- "Logger" on page 104
- "Not Authorized (type 1)" on page 106
- "Not Authorized (type 2)" on page 107
- "Not Authorized (type 3)" on page 109
- "Not Authorized (type 4)" on page 111
- "Not Authorized (type 5)" on page 112
- "Not Authorized (type 6)" on page 114
- "Put Inhibited" on page 116
- "Queue Depth High" on page 118
- "Queue Depth Low" on page 120
- "Queue Full" on page 122
- "Queue Manager Active" on page 124
- "Queue Manager Not Active" on page 125
- "Queue Service Interval High" on page 126
- "Queue Service Interval OK" on page 128
- "Queue Type Error" on page 130
- "Refresh object" on page 132
- "Remote Queue Name Error" on page 136
- "Transmission Queue Type Error" on page 138
- "Transmission Queue Usage Error" on page 140
- "Unknown Alias Base Queue" on page 142
- "Unknown Default Transmission Queue" on page 144
- "Unknown Object Name" on page 146
- "Unknown Remote Queue Manager" on page 148
- "Unknown Transmission Queue" on page 150

Event message format

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

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

Table 14. Event message structure for queue service interval events

Message descriptor	Message data	
MQMD structure	PCF header MQCFH structure	Event data [1]
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 Command identifier Message sequence number Control options Completion code Reason code Parameter count	Queue manager name Queue name Time since last reset Maximum number of messages on queue Number of messages put to queue Number of messages retrieved from queue

Note:

1. 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 message descriptor for an event message contains information that can be used by a 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 "Event message 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 in programmable command format (PCF), as 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:

- The category of event. Whether the event is a queue manager, performance, channel, configuration, command, or logger 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 "Event message MQCFH (PCF header)" on page 56.

Event data:

See "Event message descriptions" on page 57.

Event message 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®, HP-UX, z/OS, HP OpenVMS, i5/OS, Solaris, Linux, Windows, and all WebSphere MQ clients connected to these systems.

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

Format

Description:	Format name of message data.
Datatype:	MQCHAR8.
Value:	**MQFMT_EVENT** Event message.

Priority

Description:	Message priority.
Datatype:	MQLONG.
Value:	**MQPRI_PRIORITY_AS_Q_DEF** The priority is that of the event queue.

Persistence

Description:	Message persistence.
Datatype:	MQLONG.
Value:	**MQPER_PERSISTENCE_AS_Q_DEF** The priority is that of the event queue.

MsgId

Description:	Message identifier.
Datatype:	MQBYTE24.
Value:	A unique value generated by the queue manager.

CorrelId

Description:	Correlation identifier.
Datatype:	MQBYTE24.
Value:	

For performance, queue manager, logger, channel, bridge, and SSL events:

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

For such events on a shared queue, a nonzero correlation identifier is set. This parameter is set so that 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

For configuration and command events:

A unique nonzero correlation identifier
All messages relating to the same event have the same CorrelId.

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.

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.

Event message MQCFH (PCF header)

For an event, the MQCFH structure contains the following values:

Type

Description:	Structure type that identifies the content of the message.
Datatype:	MQLONG.
Value:	**MQCFH_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 and command events.
	MQCFH_VERSION_2 Version-2 for configuration events.
	MQCFH_VERSION_3 Version-3 for command 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.
	MQCMD_COMMAND_EVENT Command event.
	MQCMD_LOGGER_EVENT Logger 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 that follow the MQCFH structure. A group structure (MQCFGR), and its included parameter structures, are counted as one structure only.
Datatype:	MQLONG.
Values:	0 or greater.

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 parameters of the structures are optional, and are returned only if they contain information that is relevant to the circumstances giving rise to the event. The values in the data structures depend on the circumstances that caused the event to be generated.

Note:

1. The PCF structures in the message 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 Chapter 6, "Structure datatypes," on page 311.

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 *BaseObjectName* 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.

BaseObjectName

Description:	Object name to which the alias resolves.
Identifier:	MQCA_BASE_OBJECT_NAME. For compatibility with existing applications you can still use 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

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

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.

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

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 only.
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 command. **MQEVO_INIT** Initialization input data set command. **MQEVO_INTERNAL** Directly by queue manager. **MQEVO_MQSET** MQSET call. **MQEVO_MSG** Command message on SYSTEM.COMMAND.INPUT. **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.
Values:	**MQOT_CHANNEL** Channel. **MQOT_NAMELIST** Namelist. **MQOT_PROCESS** Process. **MQOT_Q** Queue. **MQOT_Q_MGR** Queue manager. **MQOT_STORAGE_CLASS** Storage class. **MQOT_AUTH_INFO** Authentication information. **MQOT_TOPIC** Topic.
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_Q_MGR_NAME • MQCA_STORAGE_CLASS • MQCA_AUTH_INFO_NAME • MQCA_CF_STRUC_NAME • MQCA_TOPIC_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 Chapter 7, “Event data for object attributes,” on page 335

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.
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:	All, except 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.

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:	All, except 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.

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.
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.

ConversionReasonCode

Description: Identifier of the cause of the conversion error.
Identifier: MQIACF_CONV_REASON_CODE.
Datatype: MQCFIN.
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 following: • MaxActiveChannels parameter in the qm.ini file for AIX, HP-UX, and Solaris • MaxActiveChannels parameter in the Registry for Windows. • ACTCHL parameter on the ALTER QMGR command 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.
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 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:	All.
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. 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.
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.

SSLReturnCode

Description:	A numeric return code from a failing SSL call. 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.
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. Client connections do not produce this event.
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 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. Client connections do not produce this event.
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_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.
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.

Command

Event name:	Command.
Reason code in MQCFH:	MQRC_COMMAND_MQSC (2412, X'96C'). MQSC command successfully issued, or, MQRC_COMMAND_PCF (2413, X'96D'). PCF command successfully issued.
Event description:	Command successfully issued.
Event type:	Command.
Platforms:	WebSphere MQ for z/OS only.
Event queue:	SYSTEM.ADMIN.COMMAND.EVENT.

Event data

The event data consists of two groups, *CommandContext* and *CommandData*.

CommandContext

Description:	PCF group containing the elements related to the context of the issued command.
Identifier:	MQGACF_COMMAND_CONTEXT.
Datatype:	MQCFGR.
PCF elements in group:	• *EventUserId* • *EventOrigin* • *EventQMgr* • *EventAccountingToken* • *EventIdentityData* • *EventApplType* • *EventApplName* • *EventApplOrigin* • *Command*
Returned:	Always.

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; 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 command. **MQEVO_INIT** Initialization input data set command. **MQEVO_MSG** Command message on SYSTEM.COMMAND.INPUT. **MQEVO_INTERNAL** Directly by queue manager. **MQEVO_OTHER** None of the above.
Returned:	Always.

EventQMgr

Description:	The queue manager where the command 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.

EventIdentityData

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.

Command

Description:	The command code.
Identifier:	MQIACF_COMMAND.
Datatype:	MQCFIN.

Values:

- If the event relates to a PCF command, then the value is that of the Command parameter in the MQCFH structure in the command message.
- If the event relates to an MQSC command, then the value is as follows:

MQCMD_ARCHIVE_LOG
ARCHIVE LOG

MQCMD_BACKUP_CF_STRUC
BACKUP CFSTRUCT

MQCMD_CHANGE_AUTH_INFO
ALTER AUTHINFO

MQCMD_CHANGE_BUFFER_POOL
ALTER BUFFPOOL

MQCMD_CHANGE_CF_STRUC
ALTER CFSTRUCT

MQCMD_CHANGE_CHANNEL
ALTER CHANNEL

MQCMD_CHANGE_NAMELIST
ALTER NAMELIST

MQCMD_CHANGE_PAGE_SET
ALTER PSID

MQCMD_CHANGE_PROCESS
ALTER PROCESS

MQCMD_CHANGE_Q
ALTER QLOCAL/QREMOTE/QALIAS/QMODEL

MQCMD_CHANGE_Q_MGR
ALTER QMGR, DEFINE MAXSMSGS

MQCMD_CHANGE_SECURITY
ALTER SECURITY

MQCMD_CHANGE_STG_CLASS
ALTER STGCLASS

MQCMD_CHANGE_TRACE
ALTER TRACE

MQCMD_CLEAR_Q
: CLEAR QLOCAL

MQCMD_CREATE_AUTH_INFO
: DEFINE AUTHINFO

MQCMD_CREATE_BUFFER_POOL
: DEFINE BUFFPOOL

MQCMD_CREATE_CF_STRUC
: DEFINE CFSTRUCT

MQCMD_CREATE_CHANNEL
: DEFINE CHANNEL

MQCMD_CREATE_LOG
: DEFINE LOG

MQCMD_CREATE_NAMELIST
: DEFINE NAMELIST

MQCMD_CREATE_PAGE_SET
: DEFINE PSID

MQCMD_CREATE_PROCESS
: DEFINE PROCESS

MQCMD_CREATE_Q
: DEFINE QLOCAL/QREMOTE/QALIAS/QMODEL

MQCMD_CREATE_STG_CLASS
: DEFINE STGCLASS

MQCMD_DELETE_AUTH_INFO
: DELETE AUTHINFO

MQCMD_DELETE_BUFFER_POOL
: DELETE BUFFPOOL

MQCMD_DELETE_CF_STRUC
: DELETE CFSTRUCT

MQCMD_DELETE_CHANNEL
: DELETE CHANNEL

MQCMD_DELETE_NAMELIST
: DELETE NAMELIST

MQCMD_DELETE_PAGE_SET
: DELETE PSID

MQCMD_DELETE_PROCESS
: DELETE PROCESS

MQCMD_DELETE_Q
: DELETE QLOCAL/QREMOTE/QALIAS/QMODEL

MQCMD_DELETE_STG_CLASS
: DELETE STGCLASS

MQCMD_INQUIRE_ARCHIVE
: DISPLAY ARCHIVE

MQCMD_INQUIRE_AUTH_INFO
: DISPLAY AUTHINFO

MQCMD_INQUIRE_CF_STRUC
: DISPLAY CFSTRUCT

MQCMD_INQUIRE_CF_STRUC_STATUS
DISPLAY CFSTATUS

MQCMD_INQUIRE_CHANNEL
DISPLAY CHANNEL

MQCMD_INQUIRE_CHANNEL_INIT
DISPLAY CHINIT

MQCMD_INQUIRE_CHANNEL_STATUS
DISPLAY CHSTATUS

MQCMD_INQUIRE_CLUSTER_Q_MGR
DISPLAY CLUSQMGR

MQCMD_INQUIRE_CMD_SERVER
DISPLAY CMDSERV

MQCMD_INQUIRE_CONNECTION
DISPLAY CONN

MQCMD_INQUIRE_LOG
DISPLAY LOG

MQCMD_INQUIRE_NAMELIST
DISPLAY NAMELIST

MQCMD_INQUIRE_PROCESS
DISPLAY PROCESS

MQCMD_INQUIRE_Q
DISPLAY QUEUE

MQCMD_INQUIRE_Q_MGR
DISPLAY QMGR, DISPLAY MAXSMSGS

MQCMD_INQUIRE_QSG
DISPLAY GROUP

MQCMD_INQUIRE_Q_STATUS
DISPLAY QSTATUS

MQCMD_INQUIRE_SECURITY
DISPLAY SECURITY

MQCMD_INQUIRE_STG_CLASS
DISPLAY STGCLASS

MQCMD_INQUIRE_SYSTEM
DISPLAY SYSTEM

MQCMD_INQUIRE_THREAD
DISPLAY THREAD

MQCMD_INQUIRE_TRACE
DISPLAY TRACE

MQCMD_INQUIRE_USAGE
DISPLAY USAGE

MQCMD_MOVE_Q
MOVE QLOCAL

MQCMD_PING_CHANNEL
PING CHANNEL

MQCMD_RECOVER_BSDS
RECOVER BSDS

MQCMD_RECOVER_CF_STRUC
RECOVER CFSTRUCT

MQCMD_REFRESH_CLUSTER
REFRESH CLUSTER

MQCMD_REFRESH_Q_MGR
REFRESH QMGR

MQCMD_REFRESH_SECURITY
REFRESH SECURITY

MQCMD_RESET_CHANNEL
RESET CHANNEL

MQCMD_RESET_CLUSTER
RESET CLUSTER

MQCMD_RESET_Q_STATS
RESET QSTATS

MQCMD_RESET_TPIPE
RESET TPIPE

MQCMD_RESOLVE_CHANNEL
RESOLVE CHANNEL

MQCMD_RESOLVE_INDOUBT
RESOLVE INDOUBT

MQCMD_RESUME_Q_MGR
RESUME QMGR other than CLUSTER/CLUSNL

MQCMD_RESUME_Q_MGR_CLUSTER
RESUME QMGR CLUSTER/CLUSNL

MQCMD_REVERIFY_SECURITY
REVERIFY SECURITY

MQCMD_SET_ARCHIVE
SET ARCHIVE

MQCMD_SET_LOG
SET LOG

MQCMD_SET_SYSTEM
SET SYSTEM

MQCMD_START_CHANNEL
START CHANNEL

MQCMD_START_CHANNEL_INIT
START CHINIT

MQCMD_START_CHANNEL_LISTENER
START LISTENER

MQCMD_START_CMD_SERVER
START CMDSERV

MQCMD_START_TRACE
START TRACE

MQCMD_STOP_CHANNEL
STOP CHANNEL

MQCMD_STOP_CHANNEL_INIT
STOP CHINIT

MQCMD_STOP_CHANNEL_LISTENER
STOP LISTENER

MQCMD_STOP_CMD_SERVER
STOP CMDSERV

MQCMD_STOP_Q_MGR
STOP QMGR

MQCMD_STOP_TRACE
STOP TRACE

MQCMD_SUSPEND_Q_MGR
SUSPEND QMGR other than CLUSTER/CLUSNL

MQCMD_SUSPEND_Q_MGR_CLUSTER
SUSPEND QMGR CLUSTER/CLUSNL

Returned:	Always.

CommandData

Description:	PCF group containing the elements related to the command data.
Identifier:	MQGACF_COMMAND_DATA.
Datatype:	MQCFGR.
PCF elements in group:	• If generated for an MQSC command, this group only contains the PCF element *CommandMQSC*. • If generated for a PCF command, this group contains the PCF elements that comprised the PCF command, exactly as in the command message.
Returned:	Always.

CommandMQSC

Description:	The text of the MQSC command.
Identifier:	MQCACF_COMMAND_MQSC.
Datatype:	MQCFST.
Maximum length:	MQ_COMMAND_MQSC_LENGTH.
Returned:	Only if Reason in the message descriptor is MQRC_COMMAND_MQSC.

Create object

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 only.
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 command.

MQEVO_INIT
Initialization input data set command.

MQEVO_INTERNAL
Directly by queue manager.

MQEVO_MQSET
MQSET call.

MQEVO_MSG
Command message on SYSTEM.COMMAND.INPUT.

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.

Values:	**MQOT_CHANNEL** Channel. **MQOT_NAMELIST** Namelist. **MQOT_PROCESS** Process. **MQOT_Q** Queue. **MQOT_STORAGE_CLASS** Storage class. **MQOT_AUTH_INFO** Authentication information. **MQOT_CF_STRUC** CF structure. **MQOT_TOPIC** Topic.
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_AUTH_INFO_NAME • MQCA_CF_STRUC_NAME • MQCA_TOPIC_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 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 Chapter 7, “Event data for object attributes,” on page 335

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

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 only.
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 command.

MQEVO_INIT
Initialization input data set command.

MQEVO_INTERNAL
Directly by queue manager.

MQEVO_MSG
Command message on SYSTEM.COMMAND.INPUT.

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.

Values:

MQOT_CHANNEL
Channel.

MQOT_NAMELIST
Namelist.

MQOT_PROCESS
Process.

MQOT_Q
Queue.

MQOT_STORAGE_CLASS
Storage class.

MQOT_AUTH_INFO
Authentication information.

MQOT_CF_STRUC
CF structure.

MQOT_TOPIC
Topic.

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_AUTH_INFO_NAME
- MQCA_CF_STRUC_NAME
- MQCA_TOPIC_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 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 Chapter 7, "Event data for object attributes," on page 335

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.

Logger

Event name:	Logger.
Reason code in MQCFH:	MQRC_LOGGER_STATUS (2411, X'96B') New log extent started.
Event description:	Issued when a queue manager starts writing to a new log extent or on i5/OS a new journal receiver.
Event type:	Logger.
Platforms:	All, except WebSphere MQ for z/OS.
Event queue:	SYSTEM.ADMIN.LOGGER.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.

CurrentLogExtent

Description:	Name of the log extent, or on i5/OS the journal receiver being written, when the event message was generated.
Identifier:	MQCACF_CURRENT_LOG_EXTENT_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_LOG_EXTENT_NAME_LENGTH.
Returned:	Always.

RestartRecoveryLogExtent

Description:	Name of the oldest log extent, or on i5/OS the oldest journal receiver, required by the queue manager to perform restart recovery.
Identifier:	MQCACF_RESTART_LOG_EXTENT_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_LOG_EXTENT_NAME_LENGTH.
Returned:	Always.

MediaRecoveryLogExtent

Description:	Name of the oldest log extent, or on i5/OS the oldest journal receiver, required by the queue manager to perform media recovery.
Identifier:	MQCACF_MEDIA_LOG_EXTENT_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_LOG_EXTENT_NAME_LENGTH.
Returned:	Always.

LogPath

Description:	The directory where log files are created by the queue manager.
Identifier:	MQCACF_LOG_PATH.
Datatype:	MQCFST.

Maximum length:	MQ_LOG_PATH_LENGTH.
Returned:	Always.

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.
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.
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 being opened is of type QUEUE.

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.

TopicString

Description:	Topic string being subscribed to, or opened.
Identifier:	MQCA_TOPIC_STRING.
Datatype:	MQCFST.
Maximum length:	MQ_TOPIC_STR_LENGTH.
Returned:	If the object opened is a topic object.

AdminTopicNames

Description:	List of topic admin objects against which authority is checked.
Identifier:	MQCA_ADMIN_TOPIC_NAMES
Datatype:	MQCFSL.
Maximum length:	MQ_TOPIC_NAME_LENGTH.
Returned:	If the object opened is a topic 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:	When closing a queue using the 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. When closing a subscription using an MQCLOSE call, the user has requested that the subscription be removed using the MQCO_REMOVE_SUB option, but the user is not the creator of the subscription or does not have *sub* authority on the TOPIC associated with the subscription.
Event type:	Authority.
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 for type 3 authority events.
Identifier:	MQIACF_REASON_QUALIFIER.
Datatype:	MQCFIN.
Values:	**MQRQ_SUB_NOT_AUTHORIZED** Subscribe not authorized.
Returned:	Always.

Options

Description:	Options specified on the MQSUB call
Identifier:	MQIACF_SUB_OPTIONS
Datatype:	MQCFIN.
Returned:	Always.

UserIdentifier

Description:	User identifier that caused the authorization check
Identifier:	MQCACF_APPL_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_APPL_NAME_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.

QName

Description:	Object name from object descriptor (MQOD).
Identifier:	MQCA_Q_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	If the handle being closed is to a queue

SubName

Description:	Name of subscription being removed.
Identifier:	MQCACF_SUB_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_SUB_NAME_LENGTH.
Returned:	If the handle being closed is to a subscription.

TopicString

Description:	Topic string being opened.
Identifier:	MQCA_TOPIC_STRING
Datatype:	MQCFST.
Maximum length:	MQ_TOPIC_STR_LENGTH.
Returned:	If the handle being closed is to a subscription.

AdminTopicNames

Description:	List of topic administration objects against which authority was checked.
Identifier:	MQCA_ADMIN_TOPIC_NAMES
Datatype:	MQCFSL.
Maximum length:	MQ_TOPIC_NAME_LENGTH.
Returned:	If the handle being closed is to a subscription.

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.
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 "Event message MQCFH (PCF header)" on page 56.
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.

Not Authorized (type 5)

Event name:	Not Authorized (type 5).
Reason code in MQCFH:	MQRC_NOT_AUTHORIZED (2035, X'7F3'). Not authorized for access.
Event description:	On an MQSUB call, the user is not authorized to subscribe to the specified topic.
Event type:	Authority.
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 for type 5 authority events.
Identifier:	MQIACF_REASON_QUALIFIER.
Datatype:	MQCFIN.
Values:	**MQRQ_SUB_NOT_AUTHORIZED** Subscribe not authorized.
Returned:	Always.

Options

Description:	Options specified on the MQSUB call.
Identifier:	MQIACF_SUB_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.

TopicString

Description:	Topic string being opened or subscribed to.
Identifier:	MQCA_TOPIC_STRING.
Datatype:	MQCFST.
Maximum length:	MQ_TOPIC_STR_LENGTH.
Returned:	Always.

AdminTopicNames

Description:	List of topic administration objects against which authority is checked.
Identifier:	MQCA_ADMIN_TOPIC_NAMES.
Datatype:	MQCFSL.
Maximum length:	MQ_TOPIC_NAME_LENGTH.
Returned:	Always.

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

Not Authorized (type 6)

Event name:	Not Authorized (type 6).
Reason code in MQCFH:	MQRC_NOT_AUTHORIZED (2035, X'7F3'). Not authorized for access.
Event description:	On an MQSUB call, the user is not authorized to use the destination queue with the required level of access. This event is only returned for subscriptions using non-managed destination queues. When creating, altering, or resuming a subscription, and a handle to the destination queue is supplied on the request, the user does not have PUT authority on the destination queue provided. When resuming or alerting a subscription and the handle to the destination queue is to be returned on the MQSUB call, and the user does not have PUT, GET and BROWSE authority on the destination queue.
Event type:	Authority.
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 for type 6 authority events.
Identifier:	MQIACF_REASON_QUALIFIER.
Datatype:	MQCFIN.
Values:	**MQRO_SUB_DEST_NOT_AUTHORIZED** Subscription destination queue usage not authorized.
Returned:	Always.

Options

Description:	Options specified on the MQSUB call.
Identifier:	MQIACF_SUB_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.

TopicString

Description:	Topic string being subscribed to.
Identifier:	MQCA_TOPIC_STRING.
Datatype:	MQCFST.
Maximum length:	MQ_TOPIC_STR_LENGTH.
Returned:	Always.

DestQMgrName

Description:	Hosting queue manager name of the subscription's destination queue.
Identifier:	MQCACF_OBJECT_Q_MGR_NAME
Datatype:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	If the queue manager hosting the destination queue is not the queue manager to which the application is currently connected.

DestQName

Description:	The subscription's destination queue.
Identifier:	MQCA_Q_NAME
Datatype:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

DestOpenOptions

Description:	The open options requested for the destination queue.
Identifier:	MQIACF_OPEN_OPTIONS
Datatype:	MQCFIN.
Returned:	Always.

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

Put Inhibited

Event name:	Put Inhibited.
Reason code in MQCFH:	MQRC_PUT_INHIBITED (2051, X'803'). Put calls inhibited for the queue or topic.
Event description:	MQPUT and MQPUT1 calls are currently inhibited for the queue or topic (see the *InhibitPut* queue attribute or the *InhibitPublications* topic 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:	If the object opened is a queue object

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.

TopicString

Description:	Topic String being opened
Identifier:	MQCA_TOPIC_STRING
Datatype:	MQCFST.
Maximum length:	MQ_TOPIC_STR_LENGTH.
Returned:	If the object opened is a topic.

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.
Event type:	Performance.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.PERFM.EVENT.

Note:

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 "Event message MQMD (message descriptor)" on page 51 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:	A get operation has caused the queue depth to be decremented to or below the limit specified in the *QDepthLowLimit* attribute.
Event type:	Performance.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.PERFM.EVENT.

Note:

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 "Event message MQMD (message descriptor)" on page 51 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
Event type:	Performance.
Platforms:	All.
Event queue:	SYSTEM.ADMIN.PERFM.EVENT.

Note:

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 "Event message MQMD (message descriptor)" on page 51 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.

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 get operations or MQPUT calls have been detected within an interval greater than the limit specified in the *QServiceInterval* attribute.
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.

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 operation has been detected within an interval less than or equal to the limit specified in the *QServiceInterval* attribute.
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.
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.
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

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 only.
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 command. **MQEVO_INIT** Initialization input data set command. **MQEVO_INTERNAL** Directly by queue manager. **MQEVO_MSG** Command message on SYSTEM.COMMAND.INPUT. **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.

Values:	**MQOT_CHANNEL** Channel. **MQOT_NAMELIST** Namelist. **MQOT_PROCESS** Process. **MQOT_Q** Queue. **MQOT_Q_MGR** Queue manager. **MQOT_STORAGE_CLASS** Storage class. **MQOT_AUTH_INFO** Authentication information. **MQOT_CF_STRUC** CF structure. **MQOT_TOPIC** Topic.
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_Q_MGR_NAME • MQCA_STORAGE_CLASS • MQCA_AUTH_INFO_NAME • MQCA_CF_STRUC_NAME • MQCA_TOPIC_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 Chapter 7, "Event data for object attributes," on page 335

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 one of the following occurs: • 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. • 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.
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.

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.
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_UNKNOWN_ALIAS_BASE_Q (2082, X'822'). Unknown alias base queue or topic.
Event description:	An MQOPEN or MQPUT1 call was issued specifying an alias queue as the destination, but the *BaseObjectName* in the alias queue attributes is not recognized as a queue or topic 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.

BaseObjectName

Description:	Object name to which the alias resolves.
Identifier:	MQCA_BASE_OBJECT_NAME. For compatibility with existing applications you can still use 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.

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.

BaseType

Description:	Type of object to which the alias resolves.
Identifier:	MQIA_BASE_TYPE.
Datatype:	MQCFIN.
Values:	**MQOT_Q** Base object type is a queue **MQOT_TOPIC** Base object type is a topic
Returned:	Always.

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.
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.
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_Q_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	If the object opened is a queue object. Either *QName* or *TopicName* is returned.

ProcessName

Description:	Process object name 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. One of *ProcessName*, *QName*, or *TopicName* 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.

TopicName

Description:	Topic object name from object descriptor (MQOD).
Identifier:	MQCA_TOPIC_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_TOPIC_NAME_LENGTH.
Returned:	If the object opened is a topic object. One of *ProcessName*, *QName*, or *TopicName* is returned.

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.

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.
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.

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

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

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



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

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

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

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



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

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

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

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

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

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



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

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

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


  /**************************************************************/
  /* . . . process each message received                        */
  /**************************************************************/

  if (CompCode != MQCC_FAILED)
  {
    /************************************************************/
    /* announce a message                                       */
    /************************************************************/
    printf("\a\a\a\a\a\a\a");
    time(&ltime);
    printf(ctime(&ltime));

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

⋮

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

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

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

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

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

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

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


paras = (MQCFST *)(evtmsg + 1);
counter = 1;
while (counter <= evtmsg->ParameterCount)
{
  switch (paras->Type)
  {
    case MQCFT_STRING:
      printfmqcfst(paras);
      paras = (MQCFST *)((char *)paras
                         + paras->StrucLength);
      break;
    case MQCFT_INTEGER:
      printfmqcfin((MQCFIN*)paras);
      paras = (MQCFST *)((char *)paras
                         + paras->StrucLength);
```

```
                break;
              default:
                printf("unknown parameter type, %ld\n",
                       paras->Type);
                counter = evtmsg->ParameterCount;
                break;
            }
            counter++;
          }
        }
      }   /* end evtmsg action             */
      free(evtmsg);
      evtmsg = NULL;
    }     /* end process for successful GET */
  }       /* end message processing loop    */

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


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

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

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


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

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

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



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

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


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

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

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

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

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

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

Chapter 3. Message monitoring

An introduction to message monitoring

Message monitoring is used to identify the route a message has taken through a queue manager network. As a message is routed through a queue manager network, various applications perform *activities* on behalf of the message. These activities are used to determine the message route.

This section describes:
- "Activities and operations"
- "How activities are used"
- "How to determine a message route" on page 160
- "Message route completeness" on page 162
- "How activity information is stored" on page 163

Activities and operations

Activities are discrete actions performed on behalf of a message by an application, and consist of operations, which are single pieces of work that are performed by an application.

The following are examples of activities:
- A message channel agent (MCA) sending a message from a transmission queue down a channel.
- An MCA receiving a message from a channel and putting it on its target queue.
- An application getting a message from a queue, and putting a reply message in response.
- The WebSphere MQ publish/subscribe engine processing a message.

Activities consist of one or more *operations*. Operations are single pieces of work that are performed by an application. For example, the activity of an MCA sending a message from a transmission queue down a channel, consists of the following operations:
1. Getting a message from a transmission queue (a *Get* operation).
2. Sending the message down a channel (a *Send* operation).

In a publish/subscribe network, the activity of a message being processed by the WebSphere MQ publish/subscribe engine can consist of the following multiple operations:
1. Putting a message to a topic string (a *Put* operation).
2. Zero or more operations for each of the subscribers that are considered for receipt of the message (a *Publish* operation, a *Discarded Publish* operation or an *Excluded Publish* operation).

How activities are used

By recording information related to the activities performed on behalf of a message as it is routed through a queue manager network, you can identify the sequence of activities performed on a message. From the sequence of activities performed on a message you can determine the route that it took through the queue manager network.

By determining a message route it is possible to determine the following information:

The last known location of a message
If a message does not reach its intended destination, having a complete, or partial, message route allows the last known location of the message to be determined.

Configuration issues with a queue manager network
By studying the route a message takes through a queue manager network, it can become apparent that it has not gone where expected. There are many reasons why this can occur, for example, a channel could be inactive forcing the message to take an alternative route.

In the case of a publish/subscribe application, it is also possible to determine the route of a message being published to a topic and any messages that flow in a queue manager network as a result of being published to subscribers.In these situations, a system administrator can determine whether there are any problems in the queue manager network, and if there are, correct them.

How to determine a message route

WebSphere MQ provides techniques that allow activity information to be recorded for a message as it is routed through a queue manager network. The techniques available follow.

Activity recording

If a message has the appropriate report option specified, it requests that applications generate *activity reports* as it is routed through a queue manager network. When an application performs an activity on behalf of a message, an activity report can be generated, and delivered to an appropriate location. An activity report contains information about the activity that was performed on the message. For more information, see "Activity recording" on page 163.

The activity information collected using activity reports must be arranged in order before a message route can be determined.

Trace-route messaging
Trace-route messaging is a technique that involves sending a *trace-route message* into a queue manager network. When an application performs an activity on behalf of the trace-route message, activity information can be accumulated in the message data of the trace-route message, or activity reports can be generated. If activity information is accumulated in the message data of the trace-route message, when it reaches its target queue a trace-route reply message containing all the information from the trace-route message can be generated and delivered to an appropriate location.

Because a trace-route message is dedicated to recording the sequence of activities performed on its behalf, there are more processing options available compared with normal messages that request activity reports. For comparisons between these techniques, see Table 15 on page 161.

Trace-route messaging is described in "Trace-route messaging" on page 169.

Only certain applications record activity information, see "Message route completeness" on page 162.

Comparing activity recording and trace-route messaging

Both activity recording and trace-route messaging can provide activity information to determine the route a message has taken through a queue manager network. Both methods have their own advantages. Table 15 shows the benefits of each method:

Table 15. Comparing activity recording and trace-route messaging

Benefit	**Activity recording**	**Trace-route messaging**
Can determine the last known location of a message	Yes	Yes
Can determine configuration issues with a queue manager network	Yes	Yes
Can be requested by any message (is not restricted to use with trace-route messages)	Yes	No
Message data is left unmodified	Yes	No
Message processed normally	Yes	No
Activity information can be accumulated in the message data	No	Yes
Optional message delivery to target queue	No	Yes
If a message is caught in an infinite loop, it can be detected and dealt with	No	Yes
Activity information can be put in order reliably	No	Yes
Application provided to display the activity information	No	Yes

The WebSphere MQ display route application

All WebSphere MQ platforms, with the exception of WebSphere MQ for z/OS, provide an application called the WebSphere MQ display route application, which is available by way of the command **dspmqrte**. The WebSphere MQ display route application provides a command line interface that allows you to do the following:

Configure, generate, and put trace-route messages into a queue manager network.
Characteristics of the trace-route messages can be specified such as:
- The destination of the trace-route message.
- How the trace-route message mimics another message.
- How the trace-route message should be handled as it is routed through a queue manager network.
- Whether activity recording or trace-route messaging are used to record activity information.

Order and display activity information related to a trace-route message.
The WebSphere MQ display route application can arrange in order and display any activity information returned to it.

For more information on the WebSphere MQ display route application, see "WebSphere MQ display route application" on page 185.

Approaches to message monitoring

There are various reasons for determining a message route as described in "How activities are used" on page 159. Depending on the reason for determining a message route, there are two general approaches that you can use, as follows:

Using activity information recorded for a trace-route message
Trace-route messages are used to record activity information for a specific purpose. They can be used to determine configuration issues with a queue manager network, or used to determine the last known location of a message. If a trace-route message is generated to determine the last known location of a message that did not reach its intended destination, it can mimic the original message. This gives the trace-route message the greatest chance of following the route the original message took. For information on how to mimic a message, see "Mimicking a message" on page 175.

The WebSphere MQ display route application can generate trace-route messages.

Using activity information recorded for the original message
Any message can be enabled for activity recording and have activity information recorded on its behalf. If a message doesn't reach its intended destination then the recorded activity information can be used to determine the last known location of the message. By using activity information from the original message, the most accurate possible message route can be determined, leading to the last known location. To use this approach, the original message must be enabled for activity recording.

Warning: It is recommended that you do not enable all messages in a queue manager network for activity recording. Messages enabled for activity recording can have many activity reports generated on their behalf. If every message in a queue manager network is enabled for activity recording, the queue manager network traffic can increase many times over.

Message route completeness

In some cases it is not possible to identify the full sequence of activities performed on behalf of a message. If this is the case then only a partial message route can be determined. The completeness of a message route is directly influenced by the queue manager network that the messages are routed through. The way in which the completeness of a message route is influenced depends on the level of the queue managers in the queue manager network, as follows:

Queue managers at WebSphere MQ Version 6.0 and subsequent releases
MCAs and user-written applications connected to queue managers at WebSphere MQ Version 6.0 or subsequent releases, can record information related to the activities performed on behalf of a message. The recording of activity information is controlled by the queue manager attributes ACTIVREC and ROUTEREC. If a queue manager network consists of queue managers at WebSphere MQ Version 6.0 or subsequent releases only, complete message routes can be determined.

WebSphere MQ queue managers before Version 6.0
Applications connected to WebSphere MQ queue managers before Version

6.0 **do not** record the activities that they have performed on behalf of a message. If a queue manager network contains any WebSphere MQ queue manager prior to Version 6.0, then only a partial message route can be determined.

How activity information is stored

Activity information can be stored in activity reports, trace-route messages, or trace-route reply messages. In all cases the information is stored in exactly the same format; in a structure called the *Activity* PCF group. A trace-route message or trace-route reply message can contain many Activity PCF groups depending on the number of activities performed on the message. Activity reports contain one Activity PCF group because a separate activity report is generated for every recorded activity. For details of the fields in the Activity PCF group, see “Activity report reference” on page 206.

When using trace-route messaging, additional information can be recorded. This additional information is stored in a structure called the *TraceRoute* PCF group. The TraceRoute PCF group contains a number of PCF structures that are used to store additional activity information, and to specify options that determine how the trace-route message is handled as it is routed through a queue manager network. For more information on the TraceRoute PCF group, see “The TraceRoute PCF group” on page 176.

Activity recording

This chapter contains the following:
- “An introduction to activity recording”
- “Controlling activity recording” on page 164
- “Using a common queue for activity reports” on page 166
- “Using activity reports” on page 166

An introduction to activity recording

Activity recording is a technique that is used to determine the routes messages take through a queue manager network. To determine the route a message has taken, the activities performed on behalf of the message are recorded. When using activity recording, each activity performed on behalf of a message can be recorded in an activity report. An activity report is a type of report message. Each activity report contains information about the application that performed the activity on behalf of the message, when the activity took place, and information about the operations that were performed as part of the activity. For information on activities, and operations, see “Activities and operations” on page 159. Activity reports are typically delivered to a reply-to queue where they are collected together. By studying the activity reports related to a message, the route that the message took through the queue manager network can be determined. For more information on activity information recorded with activity reports, see “Activity report reference” on page 206.

What activity reports are used for

When messages are routed through a queue manager network, activity reports can be generated. The information returned in activity reports can be used in the following ways:

Determine the last known location of a message
: If a message that is enabled for activity recording does not reach its intended destination, activity reports generated for the message as it was routed through a queue manager network can be studied to determine the last known location of the message.

Determine configuration issues with a queue manager network
: A number of messages enabled for activity recording can be sent into a queue manager network. By studying the activity reports related to each message it can become apparent that they have not taken the expected route. There are many reasons why this can occur, for example, a channel could have stopped, forcing the message to take an alternative route. In these situations, a system administrator can determine whether there are any problems in the queue manager network, and if there are, correct them.

Note: You can use activity recording in conjunction with trace-route messages by using the WebSphere MQ display route application. For more information, see "WebSphere MQ display route application" on page 185.

Activity report format

Activity reports are PCF messages generated by applications that have performed an activity on behalf of a message. Activity reports are standard WebSphere MQ report messages containing a message descriptor and message data, as follows:

The message descriptor
: An MQMD structure

Message data
: Consists of the following:
  - An embedded PCF header (MQEPH).
  - Activity report message data.

Activity report message data consists of the *Activity* PCF group, and if generated for a trace-route message, the *TraceRoute* PCF group.

For more information on the format of activity reports, see "Activity report reference" on page 206.

Controlling activity recording

Activity recording is controlled at the queue manager level. Therefore, to enable an entire queue manager network, every queue manager in this network must be enabled for activity recording individually. The more enabled queue managers, the more activity reports that can be generated.

For activity reports to be generated for a message as it is routed through a queue manager, the following are required:
- The message must be defined to request activity reports, see "Requesting activity reports for a message" on page 165.
- The queue manager must be enabled for activity recording, see "Controlling queue managers for activity recording" on page 165.
- Applications performing activities on the message must be capable of generating activity reports, see"Enabling applications for activity recording" on page 165.

Alternatively, to specify that activity reports are not to be generated for a message as it is routed through a queue manager, the queue manager can be disabled for activity recording, see "Controlling queue managers for activity recording."

Requesting activity reports for a message

To request that activity reports be generated for a message, do the following:

1. In the message descriptor of the message, specify MQRO_ACTIVITY in the *Report* field.
2. In the message descriptor of the message, specify the name of a reply-to queue in the *ReplyToQ* field.

Warning: It is recommended that you do not enable all messages in a queue manager network for activity recording. Messages enabled for activity recording can have many activity reports generated on their behalf. If every message in a queue manager network is enabled for activity recording, the queue manager network traffic can increase many times over.

Controlling queue managers for activity recording

To control whether queue managers are enabled or disabled for activity recording, use the queue manager attribute *ACTIVREC*. You can use the MQSC command `ALTER QMGR` specifying the parameter `ACTIVREC` to change the value of the queue manager attribute. The value can be:

MSG The queue manager is enabled for activity recording. Any activity reports generated are delivered to the reply-to queue specified in the message descriptor of the message. This is the default value.

QUEUE
The queue manager is enabled for activity recording. Any activity reports generated are delivered to the local system queue SYSTEM.ADMIN.ACTIVITY.QUEUE. The system queue can also be used to forward activity reports to a common queue, see "Using a common queue for activity reports" on page 166.

DISABLED
The queue manager is disabled for activity recording. No activity reports are generated while in the scope of this queue manager.

For example, to enable a queue manager for activity recording and specify that any activity reports generated are delivered to the local system queue SYSTEM.ADMIN.ACTIVITY.QUEUE, use the following MQSC command:

```
ALTER QMGR ACTIVREC(QUEUE)
```

Note: If the queue manager attribute *ACTIVREC* is modified, a running MCA will not pick up the updates until the channel is restarted.

Enabling applications for activity recording

Message channel agents (MCAs) are enabled for activity recording. MCAs use the following algorithm to determine whether to generate an activity report for a message:

1. Verify that the message has requested activity reports to be generated, see "Requesting activity reports for a message."

2. Verify that the queue manager where the message currently resides is enabled for activity recording, see “Controlling queue managers for activity recording” on page 165.
3. Generate an activity report. For the format and content of activity reports, see “Activity report reference” on page 206.
4. Put the activity report on the queue determined by the queue manager attribute *ACTIVREC*.

Other user applications should follow this algorithm to be enabled for activity recording.

Using a common queue for activity reports

Depending on the queue manager attribute, ACTIVREC, activity reports are delivered to either:

- The reply-to queue specified in the message.
- The local system queue SYSTEM.ADMIN.ACTIVITY.QUEUE.

In some cases a system administrator might prefer for activity information not to be returned the reply-to queue specified in the message. If this is the case then activity reports can be delivered to the local system queue. If a number of queue managers in a queue manager network are set this way, then it can be time consuming determining the locations of the activity reports related to a specific message. An alternative is to use a common queue on a single node.

A single node is a queue manager that hosts a common queue. All the queue managers in a queue manager network can deliver activity reports to this common queue. For an example of this configuration, see Figure 2 on page 8. The benefit of using a common queue is that queue managers do not have to deliver activity reports to the reply-to queue specified in a message, and when determining the locations of the activity reports related to a message only one queue need be queried.

To set up a common queue, do the following:

1. Select, or define, a queue manager as the single node.
2. On the single node select, or define, a queue for use as the common queue.
3. On all queue managers where activity reports are to be delivered to the common queue, redefine the local system queue SYSTEM.ADMIN.ACTIVITY.QUEUE as a remote queue definition specifying the following:
   - The name of the single node as the remote queue manager name.
   - The name of the common queue as the remote queue name.

Using activity reports

Activity reports are used to determine route information about a message. For common uses see “What activity reports are used for” on page 163. The activity information collected using activity reports must be arranged in order before a message route can be determined. Initially you must determine whether there are enough activity reports on the reply-to queue to enable you to determine the required information. The order that activity reports are put on the reply-to queue does not necessarily correlate to the order in which the activities were performed.

Activity reports must be ordered manually, unless they are generated for a trace-route message, in which case the WebSphere MQ display route application can be used to order the activity reports. For more information on the WebSphere MQ display route application, see "WebSphere MQ display route application" on page 185.

To determine whether there are enough activity reports on the reply-to queue to enable you to determine the necessary information, do the following:

1. Identify all related activity reports on the reply-to queue, by comparing identifiers of the activity reports and the original message. You should set the report option of the original message such that the activity reports can be correlated with the original message.
2. Order the identified activity reports from the reply-to queue so that the route, or partial route, that the message took through the queue manager network can be determined. The following parameters from the activity report can be used when ordering them:

   OperationType
   : The operations detailed in an activity report indicate the operations that were performed as part of the activity. By considering the types of operations that were performed, it can be possible to determine the activity report that was generated directly before, or after, the current activity report.

     For example, an activity report details that an MCA sent a message from a transmission queue down a channel. The last operation detailed in the activity report has an *OperationType* of **send** and details that the message was sent using the channel, CH1, to the destination queue manager, QM1. This means that the next activity performed on the message will have occurred on queue manager, QM1, and that it will have begun with a **receive** operation from channel, CH1. By using this information you can identify the next activity report, providing it exists and has been acquired.

   OperationDate **and** *OperationTime*
   : By looking at the dates and times of the operations in each activity report, the general order of the activities can be determined.

     Warning: Unless every queue manager in the queue manager network has their system clocks synchronized, ordering this way does not guarantee the activity reports are ordered correctly. The order must be established manually.

   Once ordered, the route, or partial route, that the message took is represented by the order of the activity reports.
3. Study the activity information from the ordered activity reports, and try to determine the information you need.

If the necessary information about the message has not been determined, it might be possible to acquire further activity reports.

Retrieving further activity reports

If the activity reports related to a message have been retrieved from the reply-to queue that the message specified, but the necessary information has not been determined, then further activity reports might be available. To determine the locations of any further activity reports, do the following:

1. If there are any queue managers in the queue manager network that deliver activity reports to a common queue, do the following. For information on using a common queue, see "Using a common queue for activity reports" on page 166.
   a. Retrieve any activity reports from the common queue that have a *CorrelId* that matches the *MsgId* of the original message.
2. If there are any queue managers in the queue manager network that do not deliver activity reports to a common queue, do the following:
   a. By studying the existing activity reports, identify queue managers that the message was routed through.
   b. From these queue managers, identify the queue managers that are enabled for activity recording.
   c. From these queue managers, identify any that did not return activity reports to the specified reply-to queue.
   d. From each of the identified queue managers, check the system queue SYSTEM.ADMIN.ACTIVITY.QUEUE and retrieve any activity reports that have a *CorrelId* that matches the *MsgId* of the original message.
   e. If no activity reports are found on the system queue, check the queue manager dead letter queue if one exists.

      Note: An activity report can only be delivered to a dead letter queue if the report option, MQRO_DEAD_LETTER_Q, is set.
3. Arrange all the acquired activity reports in order.

   Once ordered the route, or partial route, that the message took is represented by the order of the activity reports.
4. Study the activity information from the ordered activity reports, and try to determine the information you need.

There are certain circumstances where recorded activity information cannot reach the specified reply-to queue, a common queue, or a system queue. For more information, see "Circumstances where activity information is not acquired."

Circumstances where activity information is not acquired

To determine the complete sequence of activities performed on behalf of a message, information related to every activity must be acquired. If the information relating to any activity has not been recorded, or has not been acquired, then only a partial sequence of activities can be determined.

The following are circumstances where activity information is not recorded:
- The message is processed by a WebSphere MQ queue manager prior to Version 6.0.
- The message is processed by a queue manager that is not enabled for activity recording.
- The application expected to process the message is not running.

The following are circumstances where recorded activity information is unable to reach the specified reply-to queue:
- There is no channel defined to route activity reports to the reply-to queue.
- The channel to route activity reports to the reply-to queue is not running.
- The remote queue definition to route activity reports back to the queue manager where the reply-to queue resides (the queue manager alias), is not defined.

- The user that generated the original message does not have open, or put, authority to the queue manager alias.
- The user that generated the original message does not have open, or put, authority to the reply-to queue.
- The reply-to queue is put inhibited.

The following are circumstances where recorded activity information is unable to reach the system queue, or a common queue:
- If a common queue is to be used and there is no channel defined to route activity reports to the common queue.
- If a common queue is to be used and the channel to route activity reports to the common queue is not running.
- If a common queue is to be used and the system queue is incorrectly defined.
- The user that generated the original message does not have open, or put, authority to the system queue.
- The system queue is put inhibited.
- If a common queue is to be used and the user that generated the original message does not have open, or put, authority to the common queue.
- If a common queue is to be used and the common queue is put inhibited.

In these circumstances, providing the activity report does not have the report option MQRO_DISCARD_MSG specified, the activity report can be retrieved from a dead letter queue if one was defined on the queue manager where the activity report was rejected. An activity report will only have this report option specified if the original message, from which the activity report was generated, had both MQRO_PASS_DISCARD_AND_EXPIRY and MQRO_DISCARD_MSG specified in the Report field of the message descriptor.

Trace-route messaging

This section contains information about trace-route messaging and how to use it for collecting activity information.

An introduction to trace-route messaging

Trace-route messaging is a technique that uses *trace-route messages* to record activity information for a message. Trace-route messaging involves sending a trace-route message into a queue manager network. As the trace-route message is routed through the queue manager network, activity information is recorded. This activity information includes information about the applications that performed the activities, when they were performed, and the operations that were performed as part of the activities. For more information on activity information recorded for trace-route messages, see “Trace-route message reference” on page 230.

What trace-route messaging is used for

The information recorded using trace-route messaging can be used in the following ways:

To determine the last known location of a message
If a message does not reach its intended destination, trace-route messaging can be used to help determine the last known location of the message. A trace-route message is sent into a queue manager network with the same target destination as the original message, in the hope that it will follow the same route. Activity information can be accumulated in the message

data of the trace-route message, or recorded using activity reports. To increase the chance that the trace-route message will follow the same route as the original message, the trace-route message can be modified to mimic the original message, see "Mimicking a message" on page 175. The activity information recorded for a trace-route message can be used to determine the last known location of the original message.

To determine configuration issues with a queue manager network
Trace-route messages are sent into a queue manager network and activity information is recorded. By studying the activity information recorded for a trace-route message, it can become apparent that the trace-route message did not follow the expected route. There are many reasons why this can occur, for example, a channel could be inactive forcing the message to take an alternative route. In these situations, a system administrator can determine whether there are any problems in the queue manager network, and if there are, correct them.

Note:

1. Trace-route messages can be configured, generated, and put in to a queue manager network using the WebSphere MQ display route application, see "WebSphere MQ display route application" on page 185.
2. If you put a trace-route message to a distribution list, the results are undefined.

How activity information is recorded

When using trace-route messaging, activity information can be recorded using either, or both, of the following techniques:

Accumulating activity information in the message data of the trace-route message
As a trace-route message is routed through a queue manager network, information about the activities performed on behalf of the trace-route message can be accumulated in the message data of the trace-route message. The activity information is stored in *Activity* PCF groups, see "How activity information is stored" on page 163. For every activity performed on behalf of the trace-route message, an *Activity* PCF group is written to the end of the PCF block in the message data of the trace-route message.

Additional activity information is recorded in trace-route messaging, in a PCF group called the *TraceRoute* PCF group. The additional activity information is stored in this PCF group, and can be used to help determine the sequence of recorded activities. For information on the parameters contained in the *TraceRoute* PCF group, see "The TraceRoute PCF group" on page 176.

This technique is controlled by the *Accumulate* parameter in the *TraceRoute* PCF group, see Accumulate.

Recording activity information using activity reports
As a trace-route message is routed through a queue manager network, an activity report can be generated for every activity that was performed on behalf of the trace-route message. The activity information is stored in the *Activity* PCF group, see "How activity information is stored" on page 163. For every activity performed on behalf of a trace-route message, an activity report is generated containing an *Activity* PCF group. Activity recording for

trace-route messages works in the same way as for any other message. For more information on activity recording, see “Activity recording” on page 163.

Activity reports generated for trace-routes messages contain additional activity information compared to the those generated for any other message. The additional information is returned in a *TraceRoute* PCF group. The information contained in the *TraceRoute* PCF group is only accurate from the time the activity report was generated. The additional information can be used to help determine the sequence of activities performed on behalf of the trace-route message. For information on the parameters contained in the *TraceRoute* PCF group, see “The TraceRoute PCF group” on page 176.

How recorded activity information is acquired

Activity information can be recorded in two ways, see “How activity information is recorded” on page 170. Depending on how activity information has been recorded, the method of acquiring the activity information once the trace-route message has reached its intended destination, or has been discarded, varies as follows:

Accumulating activity information

Once the trace-route message has reached its intended destination, or is discarded, the activity information needs to be acquired. This can be done using the following methods:

Retrieving the trace-route message

The *Deliver* parameter, in the *TraceRoute* PCF group, controls whether a trace-route message is placed on the target queue on arrival, or whether it is discarded, see Deliver. If the trace-route message is delivered to the target queue, then the trace-route message can be retrieved from this queue. Once retrieved, the WebSphere MQ display route application can be used to display the activity information, see “WebSphere MQ display route application” on page 185.

To request activity information be accumulated in the message data of a trace-route message, set the *Accumulate* parameter in the *TraceRoute* PCF group to `MQROUTE_ACCUMULATE_IN_MSG`.

Using a trace-route reply message

Once a trace-route message reaches its intended destination, or the trace-route message cannot be routed any further in a queue manager network, a trace-route reply message can be generated. A trace-route reply message contains a duplicate of all the activity information from the trace-route message, and is either delivered to a specified reply-to queue, or the system queue SYSTEM.ADMIN.TRACE.ROUTE.QUEUE. The WebSphere MQ display route application can be used to display the activity information, see “WebSphere MQ display route application” on page 185.

To request a trace-route reply message, set the *Accumulate* parameter in the *TraceRoute* PCF group to `MQROUTE_ACCUMULATE_AND_REPLY`.

Using activity reports

If activity reports are generated for a trace-route message, the activity reports must be initially located so that the activity information can be acquired. Once located, the activity reports must be ordered so that the sequence of activities can be determined. For information on how to locate and order activity reports, see "Acquiring and using recorded information" on page 181.

Once located, an alternative to manually ordering activity reports generated for a trace-route message, is to use the WebSphere MQ display route application. Given the location of the activity reports, the WebSphere MQ display route application can automatically order and display the activity information, see "WebSphere MQ display route application" on page 185.

For more information on how to acquire and use activity information for a trace-route message, see "Acquiring and using recorded information" on page 181.

Controlling trace-route messaging

Enabling a queue manager for trace-route messaging means that applications in the scope of that queue manager can write activity information to a trace-route message. To enable an entire queue manager network for trace-route messaging, every queue manager in the network must be enabled for trace-route messaging individually. The more enabled queue managers, the more recorded activity information.

When using activity reports to record activity information for a trace-route message there are further considerations, see "Controlling activity recording" on page 164.

For activity information to be recorded for a trace-route message as it is routed through a queue manager, the following are required:

- The trace-route message must define how activity information is to be recorded, see "Configuring and generating a trace-route message" on page 175.
- If accumulating activity information in the trace-route message, then the queue manager must be enabled for trace-route messaging, see "Controlling queue managers for trace-route messaging."
- If accumulating activity information in the trace-route message, then applications performing activities on the trace-route message must be capable of writing activity information to the message data of the trace-route message, see "Enabling applications for trace-route messaging" on page 173.

Controlling queue managers for trace-route messaging

To control whether queue managers are enabled or disabled for trace-route messaging use the queue manager attribute ROUTEREC. You can use the MQSC command `ALTER QMGR` specifying the parameter `ROUTEREC` to change the value of the queue manager attribute. The value can be:

MSG The queue manager is enabled for trace-route messaging. Applications within the scope of the queue manager can write activity information to the trace-route message.

If the *Accumulate* parameter in the *TraceRoute* PCF group is set as `MQROUTE_ACCUMULATE_AND_REPLY`, and the next activity to be performed on the trace-route message:

- is a discard
- is a put to a local queue (target queue or dead-letter queue)

- will cause the total number of activities performed on the trace-route message to exceed the value of parameter the *MaxActivities*, in the *TraceRoute* PCF group .

Then a trace-route reply message is generated, and delivered to the reply-to queue specified in the message descriptor of the trace-route message.

For information on the *TraceRoute* PCF group, see "The TraceRoute PCF group" on page 176.

QUEUE
: The queue manager is enabled for trace-route messaging. Applications within the scope of the queue manager can write activity information to the trace-route message.

If the *Accumulate* parameter in the *TraceRoute* PCF group is set as `MQROUTE_ACCUMULATE_AND_REPLY`, and the next activity to be performed on the trace-route message:

- is a discard
- is a put to a local queue (target queue or dead-letter queue)
- will cause the total number of activities performed on the trace-route message to exceed the value of parameter the *MaxActivities*, in the *TraceRoute* PCF group .

then a trace-route reply message is generated, and delivered to the local system queue SYSTEM.ADMIN.TRACE.ROUTE.QUEUE.

For information on the *TraceRoute* PCF group, see "The TraceRoute PCF group" on page 176.

DISABLED
: The queue manager is disabled for trace-route messaging. Activity information is not accumulated in the the trace-route message, however the *TraceRoute* PCF group can be updated while in the scope of this queue manager.

For example, to disable a queue manager for trace-route messaging, use the following MQSC command:

```
ALTER QMGR ROUTEREC(DISABLED)
```

Note: If the queue manager attribute ROUTEREC is modified a running MCA will not pick up the updates until the channel is restarted.

Enabling applications for trace-route messaging

Message channel agents (MCAs) are enabled for trace-route messaging. The algorithm that MCAs use is detailed below. To enable a user application for trace-route messaging, use this algorithm with the exception of steps 2 on page 174 and 6 on page 174.

Note: Enabling a user application for trace-route messaging can be complicated, only enable user applications where necessary.

1. Determine whether the message being processed is a trace-route message.

   Compare the format of the message with the format of a trace-route message as detailed in "Trace-route message reference" on page 230.

   - If the message conforms to the format of a trace-route message, then the algorithm continues to step 2 on page 174.

- If the message does not conform to the format of a trace-route message, then the message is not processed as a trace-route message.

2. If the trace-route message is received from a queue manager prior to WebSphere MQ Version 6.0, increment the parameter, *DiscontinuityCount*, in the trace-route message data.

   Note: User applications do not perform this step.
3. Determine whether activity information is to be recorded.

   Providing the detail level of the performed activity is not less than the level of detail specified by the parameter *Detail*, activity information will be recorded if the trace-route message requests accumulation and the queue manager is enabled for trace-route messaging, or if the trace-route message requests an activity report and the queue manager is enabled for activity recording.
   - If activity information is to be recorded, increment the parameter *RecordedActivities*. For information, see RecordedActivities.
   - If activity information is not to be recorded, increment the parameter *UnrecordedActivities*. For information, see UnrecordedActivities.
4. Determine whether the total number of activities performed on the trace-route message exceeds the value of the parameter *MaxActivities*.

   The total number of activities is the sum of *RecordedActivities*, *UnrecordedActivities*, and *DiscontinuityCount*.
   - If the total number of activities exceeds *MaxActivities*, then reject the message with feedback MQFB_MAX_ACTIVITIES.
   - If the total number of activities does not exceed *MaxActivities*, then the algorithm continues to step 5.
5. If both of the following conditions are true:
   - The value of *Accumulate* is set as MQROUTE_ACCUMULATE_IN_MSG or MQROUTE_ACCUMULATE_AND_REPLY, and
   - The queue manager is enabled for trace-route messaging

   then write an Activity PCF group to the end of the PCF block in the message data of a trace-route message.

   The format of the Activity PCF group is detailed in "Activity report message data" on page 214.
6. If delivering the trace-route message to a transmission queue, then follow the algorithm specified in Forwarding.

   Note: User applications do not perform this step.
7. If delivering the trace-route message to a local queue, then do one of the following:
   - If the parameter, *Deliver*, is specified as MQROUTE_DELIVER_NO, then reject the trace-route message with feedback MQFB_NOT_DELIVERED.
   - If the parameter, *Deliver*, is specified as MQROUTE_DELIVER_YES, then deliver the trace-route message to the local queue.
8. If all the following conditions are true:
   - The trace-route message was delivered to a local queue or rejected, and
   - The value of the parameter, *Accumulate*, is MQROUTE_ACCUMULATE_AND_REPLY, and
   - The queue manager is enabled for trace-route messaging

   then generate a trace-route reply message.

The format of the trace-route reply message is detailed in “Trace-route reply message reference” on page 241. The trace-route reply message is put on the queue determined by the queue manager attribute, ROUTEREC. For information on this queue manager attribute, see “Controlling queue managers for trace-route messaging” on page 172.

9. If the trace-route message requested an activity report and the queue manager is enabled for activity recording, then generate an activity report.

   The format of the activity report is detailed in “Activity report reference” on page 206. The activity report is put on the queue determined by the queue manager attribute, ACTIVREC. For information on this queue manager attribute, see “Controlling queue managers for activity recording” on page 165.

Configuring and generating a trace-route message

To generate a trace-route message, use one of the following methods:

- Use the WebSphere MQ display route application.
- Manually configure and generate a trace-route message.

Mimicking a message

When using a trace-route message to determine the route another message has taken through a queue manager network, it is important to mimic the original message. The more closely a trace-route message mimics the original message, the greater the chance that the trace-route message will follow the same route as the original message. Many characteristics can effect where a message is forwarded to within a queue manager network. The following are influential message characteristics:

Priority
The priority can be specified in the message descriptor of the message.

Persistence
The persistence can be specified in the message descriptor of the message.

Expiration
The expiration can be specified in the message descriptor of the message.

Report options
Report options can be specified in the message descriptor of the message.

Message size
To mimic the size of a message, additional data can be written to the message data of the message. For this purpose, additional message data can be meaningless. The WebSphere MQ display route application cannot specify message size.

Message data
Some queue manager networks use content based routing to determine where messages are forwarded. In these cases the message data of the trace-route message needs to be written to mimic the message data of the original message. The WebSphere MQ display route application cannot specify message data.

For details of all the fields available in the message descriptor of a trace-route message, see “Trace-route message reference” on page 230.

Using the WebSphere MQ display route application

The WebSphere MQ display route application, **dspmqrte**, can be used to configure, generate and put a trace-route message into a queue manager network. **dspmqrte** cannot be issued on queue managers before WebSphere MQ Version 6.0 or on WebSphere MQ for z/OS queue managers. If you want the first queue manager the trace-route message is routed through to be a queue manager of this type, connect to the queue manager as a WebSphere MQ Version 6.0 or later client using the optional parameter `-c`.

Note: When using the WebSphere MQ display route application to generate a trace-route message, the *Format* parameter in the message descriptor is set to MQFMT_ADMIN. The implication of this is that you cannot add user data to the trace-route message generated by the WebSphere MQ display route application.

For more information on how to use the WebSphere MQ display route application, see "WebSphere MQ display route application" on page 185.

Manual generation

A trace-route message consists of the following:

The message descriptor
An MQMD structure, with the *Format* field set to MQFMT_ADMIN or MQFMT_EMBEDDED_PCF.

Message data
Consists of either:

- A PCF header (MQCFH) and trace-route message data, if *Format* is set to MQFMT_ADMIN, or
- An embedded PCF header (MQEPH), trace-route message data, and additional user-specified message data, if *Format* is set to MQFMT_EMBEDDED_PCF.

The trace-route message data consists of the *TraceRoute* PCF group and one or more *Activity* PCF groups. When generating a trace-route message manually, an *Activity* PCF group is not required. *Activity* PCF groups are written to the message data of the trace-route message when an MCA or user-written application performs an activity on its behalf. The *TraceRoute* group is detailed in "The TraceRoute PCF group."

For information on the parameters contained in the message descriptor and message data of a trace-route message, see "Trace-route message reference" on page 230.

The TraceRoute PCF group:

The behavior of a trace-route message is controlled by attributes in the *TraceRoute* PCF group. The *TraceRoute* group is a PCF group that resides in the message data of every trace-route message. The following table details the parameters in the *TraceRoute* group, that an MCA will recognize. Further parameters can be added if user-written applications are written to recognize them, see "Recording additional information" on page 182.

Table 16. TraceRoute PCF group

parameter	*Type*
TraceRoute	MQCFGR
Detail	MQCFIN
RecordedActivities	MQCFIN
UnrecordedActivities	MQCFIN
DiscontinuityCount	MQCFIN
MaxActivities	MQCFIN
Accumulate	MQCFIN
Forward	MQCFIN
Deliver	MQCFIN

Descriptions of each parameter in the *TraceRoute* PCF group follows:

Detail Specifies the detail level of activity information that is to be recorded. The value can be:

MQROUTE_DETAIL_LOW
Only activities performed by user application are recorded.

MQROUTE_DETAIL_MEDIUM
Activities specified in MQROUTE_DETAIL_LOW should be recorded. Additionally, activities performed by MCAs are recorded.

MQROUTE_DETAIL_HIGH
Activities specified in MQROUTE_DETAIL_LOW, and MQROUTE_DETAIL_MEDIUM should be recorded. MCAs do not record any further activity information at this level of detail. This option is only available to user applications that are to record further activity information. For example, if a user application determines the route a message takes by considering certain message characteristics, the information about the routing logic could be included with this level of detail.

RecordedActivities
Specifies the number of recorded activities performed on behalf of the trace-route message. An activity is considered to be recorded if information about it has been written to the trace-route message, or if an activity report has been generated. For every recorded activity, *RecordedActivities* increments by one.

UnrecordedActivities
Specifies the number of unrecorded activities performed on behalf of the trace-route message. An activity is considered to be unrecorded if an application that is enabled for trace-route messaging neither accumulates, nor writes the related activity information to an activity report.

An activity performed on behalf of a trace-route message is unrecorded in the following circumstances:

- The detail level of the performed activity is less than the level of detail specified by the parameter *Detail*. For more information, see Detail.
- The trace-route message requests an activity report but not accumulation, and the queue manager is not enabled for activity recording.
- The trace-route message requests accumulation but not an activity report, and the queue manager is not enabled for trace-route messaging.

- The trace-route message requests both accumulation and an activity report, and the queue manager is not enabled for activity recording and trace route messaging.
- The trace-route message requests neither accumulation nor an activity report.

For every unrecorded activity the parameter, *UnrecordedActivities*, increments by one.

DiscontinuityCount

Specifies the number of times the trace-route message has been routed through a queue manager whose applications were not enabled for trace-route messaging. This value is incremented by the queue manager. If this value is greater than 0, then only a partial message route can be determined.

MaxActivities

Specifies the maximum number of activities that can be performed on behalf of the trace-route message.

The total number of activities is the sum of *RecordedActivities*, *UnrecordedActivities*, and *DiscontinuityCount*. The total number of activities must not exceed the value of *MaxActivities*.

The value of *MaxActivities* can be:

A positive integer

The maximum number of activities.

If the maximum number of activities is exceeded, then the trace-route message is rejected with feedback MQFB_MAX_ACTIVITIES. This can prevent the trace-route message from being forwarded indefinitely if caught in an infinite loop.

MQROUTE_UNLIMITED_ACTIVITIES

An unlimited number of activities can be performed on behalf of the trace-route message.

Accumulate

Specifies the method used to accumulate activity information. The value can be:

MQROUTE_ACCUMULATE_IN_MSG

If the queue manager is enabled for trace-route messaging, activity information is accumulated in the message data of the trace-route message.

If this value is specified, the trace-route message data consists of the following:

- The *TraceRoute* PCF group.
- Zero or more *Activity* PCF groups.

MQROUTE_ACCUMULATE_AND_REPLY

If the queue manager is enabled for trace-route messaging, activity information is accumulated in the message data of the trace-route message, and a trace-route reply message is generated if any of the following occur:

- The trace-route message is discarded by a WebSphere MQ Version 6 (or later) queue manager.

- The trace-route message is put to a local queue (target queue or dead-letter queue) by a WebSphere MQ Version 6 (or later) queue manager.
- The number of activities performed on the trace-route message exceeds the value of *MaxActivities*.

If this value is specified, the trace-route message data consists of the following:

- The *TraceRoute* PCF group.
- Zero or more *Activity* PCF groups.

MQROUTE_ACCUMULATE_NONE
Activity information is not accumulated in the message data of the trace-route message.

If this value is specified, the trace-route message data consists of the following:

- The *TraceRoute* PCF group.

Forward
Specifies where a trace-route message can be forwarded to. The value can be:

MQROUTE_FORWARD_IF_SUPPORTED
The trace-route message is only forwarded to queue managers that will honor the value of the *Deliver* parameter from the *TraceRoute* group.

MQROUTE_FORWARD_ALL
The trace-route message is forwarded to any queue manager, regardless of whether the value of the *Deliver* parameter will be honored.

Queue managers use the following algorithm when determining whether to forward a trace-route message to a remote queue manager:

1. Determine whether the remote queue manager is capable of supporting trace-route messaging.
   - If the remote queue manager is capable of supporting trace-route messaging, then the algorithm continues to step 4 on page 180.
   - If the remote queue manager is not capable of supporting trace-route messaging, then the algorithm continues to step 2
2. Determine whether the *Deliver* parameter from the *TraceRoute* group contains any unrecognized delivery options in the MQROUTE_DELIVER_REJ_UNSUP_MASK bit mask.
   - If any unrecognized delivery options are found, then the trace-route message is rejected with feedback MQFB_UNSUPPORTED_DELIVERY.
   - If no unrecognized delivery options are found, then the algorithm continues to step 3.
3. Determine the value of the parameter *Deliver* from the *TraceRoute* PCF group in the trace-route message.
   - If *Deliver* is specified as MQROUTE_DELIVER_YES, then the trace-route message is forwarded to the remote queue manager.
   - If *Deliver* is specified as MQROUTE_DELIVER_NO, then the algorithm continues to step 4 on page 180.

4. Determine whether the *Forward* parameter from the *TraceRoute* group contains any unrecognized forwarding options in the MQROUTE_FORWARDING_REJ_UNSUP_MASK bit mask.
   - If any unrecognized forwarding options are found, then the trace-route message is rejected with feedback MQFB_UNSUPPORTED_FORWARDING.
   - If no unrecognized forwarding options are found, then the algorithm continues to step 5.
5. Determine the value of the parameter *Forward* from the *TraceRoute* PCF group in the trace-route message.
   - If *Forward* is specified as MQROUTE_FORWARD_IF_SUPPORTED, then the trace-route message is rejected with feedback MQFB_NOT_FORWARDED.
   - If *Forward* is specified as MQROUTE_FORWARD_ALL, then trace-route message can be forwarded to the remote queue manager.

Deliver Specifies the action to be taken if the trace-route message reaches its intended destination. User-written applications must check this attribute before placing a trace-route message on its target queue. The value can be:

MQROUTE_DELIVER_YES
On arrival, the trace-route message is put on the target queue. Any application performing a get operation on the target queue can retrieve the trace-route message.

MQROUTE_DELIVER_NO
On arrival, the trace-route message is not delivered to the target queue. The message is processed according to its report options.

Using a common queue for trace-route reply messages

Depending on the queue manager attribute, ROUTEREC, trace-route reply messages are delivered to either:
- The reply-to queue specified in the message.
- The local system queue SYSTEM.ADMIN.TRACE.ROUTE.QUEUE.

In some cases a system administrator might prefer trace-route reply messages not to be returned to the reply-to queues specified in the message descriptor of the trace-route messages. If this is the case then queue managers can be set to deliver trace-route reply messages to local system queues. If a number of queue managers in a queue manager network are set this way, then it can be time consuming determining the locations of the trace-route reply messages. An alternative is to use a common queue on a single node.

A single node is a queue manager that hosts a common queue. All the queue managers in a queue manager network can deliver trace-route reply messages to this common queue. For an example of this configuration, see Figure 2 on page 8. The benefit of using a common queue is that queue managers do not have to deliver trace-route reply messages to the reply-to queue specified in a message, and when determining the locations of trace-route reply messages, only one queue need be queried.

To set up a common queue, do the following:
1. Select, or define, a queue manager as the single node.
2. On the single node select, or define, a queue for use as the common queue.

3. On all queue managers that will forward trace-route reply messages are to the common queue, redefine the local system queue SYSTEM.ADMIN.TRACE.ROUTE.QUEUE as a remote queue definition specifying the following:
   - The name of the single node as the remote queue manager name.
   - The name of the common queue as the remote queue name.

Acquiring and using recorded information

For a trace-route message, recorded activity information can be acquired using the following:

- A trace-route reply message.
- The trace-route message itself (having been put on a local queue).
- Activity reports.

Activity information is used to determine route information about the trace-route message, for common uses see "What trace-route messaging is used for" on page 169.

Trace-route reply messages

Activity information can only be acquired from a trace-route reply message if the location of the trace-route reply message is known.

If the trace-route reply message is not on the reply-to queue that was specified in the message descriptor of the trace-route message, check the following locations:

- The local system queue, SYSTEM.ADMIN.TRACE.ROUTE.QUEUE, on the target queue manager of the trace-route message.
- A common queue, see "Using a common queue for trace-route reply messages" on page 180.
- The local system queue, SYSTEM.ADMIN.TRACE.ROUTE.QUEUE, on any queue manager in the queue manager network.

  This can occur if the trace-route message has been put to a dead-letter queue, or the maximum number of activities was exceeded.

To use the recorded activity information in a trace-route reply message, do the following:

1. Retrieve the trace-route reply message from its known location.
2. Use the WebSphere MQ display route application to display the recorded activity information.
3. Study the activity information and determine the information you need.

Trace-route messages

Activity information can only be acquired from a trace-route message if both of the following are true:

- The trace-route message has the parameter *Accumulate* in the *TraceRoute* PCF group specified as either `MQROUTE_ACCUMULATE_IN_MSG` or `MQROUTE_ACCUMULATE_AND_REPLY`.
- The location of the trace-route message is known.

  For the trace-route message to be delivered to the target queue the *Deliver* parameter in the *TraceRoute* PCF group must be specified as `MQROUTE_DELIVER_YES`.

If the trace-route message is not on the target queue, you can try to locate the trace-route message using a trace-route message enabled for activity recording. With the generated activity reports try to determine the last known location of the trace-route message.

To use the recorded activity information in a trace-route message, do the following:

1. Retrieve the trace-route message from its known location.
2. Use the WebSphere MQ display route application to display the recorded activity information.
3. Study the activity information and determine the information you need.

Activity reports

Activity information can only be acquired from activity reports if both of the following are true:

- The report option MQRO_ACTIVITY was specified in the message descriptor of the trace-route message.
- The location of the activity reports is known.

To use the recorded activity information from activity reports, do the following:

1. Locate and order the activity reports, see "Using activity reports" on page 166.

   Once located, an alternative to manually ordering activity reports generated for a trace-route message, is to use the WebSphere MQ display route application. Given the location of the activity reports, the WebSphere MQ display route application can automatically order and display the activity information. For more information, see "WebSphere MQ display route application" on page 185.
2. Study the activity information and determine the information you need.

Circumstances where activity information is not acquired

For information on circumstances when activity information is not acquired for a trace-route message see, "Circumstances where activity information is not acquired" on page 168. When reading this section in relation to trace-route messaging, note the following exceptions:

Applicable to the entire section

- The message being processed is always a trace-route message.
- Situations that affect activity reports are applicable to trace-route reply messages also.

Applicable to circumstances where activity information is not recorded only

- Activity information is not recorded when a trace-route message is processed by a queue manager that is disabled for both activity recording and trace-route messaging.

Recording additional information

As a trace-route message is routed through a queue manager network, user applications can record additional information by including one or more additional PCF parameters when writing the *Activity* group to the message data of the trace-route message or activity report. This additional information can help system administrators to identify the route a trace-route message took, or why it was taken.

If the WebSphere MQ display route application is used to display the recorded information for a trace-route message, any additional PCF parameters can only be displayed with a numeric identifier, unless the parameter identifier of each parameter is recognized by the WebSphere MQ display route application. To recognize a parameter identifier, additional information must be recorded using the following PCF parameters. Include these PCF parameters in an appropriate place in the *Activity* PCF group.

GroupName

Description:	Grouped parameters specifying the additional information.
Identifier:	MQGACF_VALUE_NAMING.
Datatype:	MQCFGR.
Parameters in group:	*ParameterName* *ParameterValue*

ParameterName

Description:	Contains the **name** to be displayed by the WebSphere MQ display route application, which puts the value of *ParameterValue* into context.
Identifier:	MQCA_VALUE_NAME.
Datatype:	MQCFST.
Included in PCF group:	*GroupName*.
Value:	The name to be displayed.

ParameterValue

Description:	Contains the **value** to be displayed by the WebSphere MQ display route application.
Identifier:	The PCF structure identifier for the additional information.
Datatype:	The PCF structure datatype for the additional information.
Included in PCF group:	*GroupName*.
Value:	The value to be displayed.

Examples of recording additional information

This section contains two examples of how a user application can record additional information when performing an activity on behalf of a trace-route message. In both examples, the WebSphere MQ display route application is used to generate a trace-route message, and display the activity information returned to it.

1. In "Example 1," additional activity information is recorded by a user application in a format where the parameter identifier **is not** recognized by the WebSphere MQ display route application.
2. In "Example 2" on page 184, additional activity information is recorded by a user application in a format where the parameter identifier **is** recognized by the WebSphere MQ display route application.

Example 1:

1. The WebSphere MQ display route application is used to generate and put a trace-route message into a queue manager network. The necessary options are set to request the following:
   - Activity information is accumulated in the message data of the trace-route message.

- On arrival at the target queue the trace-route message is discarded, and a trace-route reply message is generated and delivered to a specified reply-to queue.
- On receipt of the trace-route reply message, the WebSphere MQ display route application displays the accumulated activity information.

The trace-route message is put into the queue manager network.

2. As the trace-route message is routed through the queue manager network a user application, that is enabled for trace-route messaging, performs a low detail activity on behalf of the message. In addition to writing the standard activity information to the trace-route message, the user application writes the following PCF parameter to the end of the Activity group:

   ColorValue

Identifier:	65536.
Datatype:	MQCFST.
Value:	'Red'

   This additional PCF parameter gives further information about the activity that was performed, however it is written in a format where the parameter identifier **will not** be recognized by the WebSphere MQ display route application.
3. The trace-route messages reaches the target queue and a trace-route reply message is returned to the WebSphere MQ display route application. The additional activity information is displayed as follows:

   ```
   65536: 'Red'
   ```

   The WebSphere MQ display route application does not recognize the parameter identifier of the PCF parameter and displays it as a numerical value. The context of the additional information is not clear.

   For an example of when the WebSphere MQ display route application does recognize the parameter identifier of the PCF parameter, see "Example 2."

Example 2:

1. The WebSphere MQ display route application is used to generate and put a trace-route message into a queue manager network in the same fashion as in "Example 1" on page 183.
2. As the trace-route message is routed through the queue manager network a user application, that is enabled for trace-route messaging, performs a low detail activity on behalf of the message. In addition to writing the standard activity information to the trace-route message, the user application writes the following PCF parameters to the end of the Activity group:

   ColorInfo

Description:	Grouped parameters specifying information about a color.
Identifier:	MQGACF_VALUE_NAMING.
Datatype:	MQCFGR.
Parameters in group:	*ColorName* *ColorValue*

ColorName

Description:	Contains the **name** to be displayed by the WebSphere MQ display route application which puts the value of *ColorValue* into context.
Identifier:	MQCA_VALUE_NAME.
Datatype:	MQCFST.
Included in PCF group:	*ColorInfo*.
Value:	'Color'

ColorValue

Description:	Contains the **value** to be displayed by the WebSphere MQ display route application.
Identifier:	65536.
Datatype:	MQCFST.
Included in PCF group:	*ColorInfo*.
Value:	'Red'

These additional PCF parameters gives further information about the activity that was performed. These PCF parameters are written in a format where the parameter identifier **will** be recognized by the WebSphere MQ display route application.

3. The trace-route messages reaches the target queue and a trace-route reply message is returned to the WebSphere MQ display route application. The additional activity information is displayed as follows:

```
Color: 'Red'
```

The WebSphere MQ display route application recognizes that the parameter identifier of the PCF structure containing the value of the additional activity information has a corresponding name. The corresponding name is displayed instead of the numerical value.

WebSphere MQ display route application

This section contains information about the display route application (dspmqrte) its purpose and how to use it.

An introduction to the WebSphere MQ display route application

The WebSphere MQ display route application (dspmqrte) can be executed on all WebSphere MQ Version 7.0 queue managers, with the exception of WebSphere MQ for z/OS queue managers. You can execute the WebSphere MQ display route application as a client to a WebSphere MQ for z/OS Version 7.0 queue manager by specifying the `-c` parameter when issuing the `dspmqrte` command.

Note: To run a Client Application against a WebSphere MQ for z/OS queue manager, the client attachment feature must be installed.

The WebSphere MQ display route application is started by the command **dspmqrte**, and is used for the following:

- To configure, generate, and put a trace-route message into a queue manager network.

  By putting a trace-route message into a queue manager network, activity information can be collected and used to determine the route that the trace-route message took.
- To order and display activity information related to a trace-route message.

  If the WebSphere MQ display route application has put a trace-route message into a queue manager network, after the related activity information has been returned, the information can be ordered and displayed immediately. Alternatively, the WebSphere MQ display route application can be used to order, and display, activity information related to a trace-route message that was previously generated.

For more information about trace-route messages, see "Trace-route messaging" on page 169.

Using the WebSphere MQ display route application

The WebSphere MQ display route application can be used to generate a trace-route message, and to display activity information recorded for a trace-route message.

This section details the parameters that are available with the WebSphere MQ display route application when generating trace-route messages, or displaying activity information. For information on how to form a syntactically valid command, see the **dspmqrte** command in the WebSphere MQ System Administration Guide.

Generating trace-route messages

The WebSphere MQ display route application, **dspmqrte**, provides many parameters that determine characteristics of the trace-route message itself, and how the trace-route message should be treated as it is routed through a queue manager network. This section gives an overview of the parameters available with the WebSphere MQ display route application that are related to the generation, configuration, and use of trace-route messages.

Connecting to a queue manager:

The queue manager that the WebSphere MQ display route application connects to is specified using the following parameters:

-c Specifies that the WebSphere MQ display route application connects as a client application. For more information on how to set up client machines, see the WebSphere MQ Clients book.

If you do not specify this parameter, the WebSphere MQ display route application does not connect as a client application.

-m *QMgrName*
The name of the queue manager to which the WebSphere MQ display route application connects. The name can contain up to 48 characters.

If you do not specify this parameter, the default queue manager is used.

The target destination:

The target destination of a trace-route message is specified using the following parameters:

-q *TargetQName*
: If the WebSphere MQ display route application is being used to send a trace-route message into a queue manager network, *TargetQName* specifies the name of the target queue.

-ts *TargetTopicString*
: Specifies the topic string.

-qm *TargetQMgr*
: Qualifies the target destination; normal queue manager name resolution will then apply. The target destination is specified with *-q TargetQName* or *-ts TargetTopicString*.

  If you do not specify this parameter, the queue manager to which the WebSphere MQ display route application is connected is used as the target queue manager.

-o Specifies that the target destination is not bound to a specific destination. Typically this parameter is used when the trace-route message is to be put across a cluster. The target destination is opened with option MQOO_BIND_NOT_FIXED.

  If you do not specify this parameter, the target destination is bound to a specific destination.

The publication topic:

For publish/subscribe applications, you can use the WebSphere MQ display route application to publish a trace-route message (that is, put it to a topic) and follow the results of the put through the network.

The topic string of a trace-route message is specified using the following parameter:

-ts *TopicName*
: Specifies a topic string to which the WebSphere MQ display route application is to publish a trace-route message, and puts this application into topic mode. In this mode, the application traces all of the messages that result from the publish request.

You can also use the WebSphere MQ display route application to display the results from an activity report that was generated for publish messages.

Mimicking a message:

One use of trace-route messaging is to help determine the last known location of a message that did not reach its intended destination. The WebSphere MQ display route application provides parameters that can help configure a trace-route message to mimic the original message. For information on the importance of mimicking the original message, see "Mimicking a message" on page 175. When mimicking a message, you can use the following parameters:

-l *Persistence*
: Specifies the persistence of the generated trace-route message. Possible values for *Persistence* are:

yes	The generated trace-route message is persistent. (MQPER_PERSISTENT).
no	The generated trace-route message is **not** persistent. (MQPER_NOT_PERSISTENT).
q	The generated trace-route message inherits its persistence value from the destination specified by *-q TargetQName* or *-ts TargetTopicString*. (MQPER_PERSISTENCE_AS_Q_DEF).

A trace-route reply message, or any report messages, returned will share the same persistence value as the original trace-route message.

If *Persistence* is specified as **yes**, you must specify the parameter *-rq ReplyToQ*. The reply-to queue must not resolve to a temporary dynamic queue.

If you do not specify this parameter, the generated trace-route message is **not** persistent.

-p *Priority*

Specifies the priority of the trace-route message. The value of *Priority* is either greater than or equal to 0, or MQPRI_PRIORITY_AS_Q_DEF. MQPRI_PRIORITY_AS_Q_DEF specifies that the priority value is taken from the destination specified by *-q TargetQName* or *-ts TargetTopicString*.

If you do not specify this parameter, the priority value is taken from the destination specified by *-q TargetQName* or *-ts TargetTopicString*.

-xs *Expiry*

Specifies the expiry time for the trace-route message, in seconds.

If you do not specify this parameter, the expiry time is specified as 60 seconds.

-ro none | *ReportOption*

none	Specifies no report options are set.
ReportOption	Specifies report options for the trace-route message. Multiple report options can be specified using a comma as a separator. Possible values for *ReportOption* are:

activity	The report option MQRO_ACTIVITY is set.
coa	The report option MQRO_COA_WITH_FULL_DATA is set.
cod	The report option MQRO_COD_WITH_FULL_DATA is set.
exception	The report option MQRO_EXCEPTION_WITH_FULL_DATA is set.
expiration	The report option MQRO_EXPIRATION_WITH_FULL_DATA is set.
discard	The report option MQRO_DISCARD_MSG is set.

If neither `-ro ReportOption` nor `-ro none` are specified, then the MQRO_ACTIVITY and MQRO_DISCARD_MSG report options are specified.

The WebSphere MQ display route application does not allow you to add user data to the trace-route message. If you require user data to be added to the trace-route message then manually generate a trace-route message, see "Configuring and generating a trace-route message" on page 175.

Accumulating activity information:

The route a trace-route message has taken is determined using recorded activity information. Recorded activity information can be returned using the following:

- Activity reports.
- A trace-route reply message.
- The trace-route message itself (having been put on the target queue).

For more information, see "How recorded activity information is acquired" on page 171.

When using **dspmqrte**, the method used to return recorded activity information is determined using the following parameters:

The `activity` report option, specified using `-ro`
Specifies that activity information is returned using activity reports. By default activity recording is enabled.

-ac -ar
Specifies that activity information is accumulated in the trace-route message, and that a trace-route reply message is to be generated.

-ac
Specifies that activity information is to be accumulated within the trace-route message.

If you do not specify this parameter, activity information is **not** accumulated within the trace-route message.

-ar Requests that a trace-route reply message containing all accumulated activity information is generated in the following circumstances:

- The trace-route message is discarded by a WebSphere MQ Version 6.0 or later queue manager.
- The trace-route message is put to a local queue (target queue or dead-letter queue) by a WebSphere MQ Version 6.0 or later queue manager.
- The number of activities performed on the trace-route message exceeds the value of specified in *-s Activities*.

-ac -d yes
Specifies that activity information is accumulated in the trace-route message, and that on arrival, the trace-route message will be put on the target queue.

-ac
Specifies that activity information is to be accumulated within the trace-route message.

If you do not specify this parameter, activity information is **not** accumulated within the trace-route message.

-d yes
On arrival, the trace-route message is put to the target queue, even if the queue manager does not support trace-route messaging.

If you do not specify this parameter, the trace-route message is **not** put to the target queue.

The trace-route message can then be retrieved from the target queue, and the recorded activity information acquired.

You can combine these methods as required.

Additionally, the detail level of the recorded activity information can be specified using the following parameter:

-t `Detail`
: Specifies the activities that are recorded. The possible values for *Detail* are:

low	Activities performed by user-defined application are recorded only.
medium	Activities specified in **low** are recorded. Additionally, publish activities and activities performed by MCAs are recorded.
high	Activities specified in **low**, and **medium** are recorded. MCAs do not expose any further activity information at this level of detail. This option is available to user-defined applications that are to expose further activity information only. For example, if a user-defined application determines the route a message takes by considering certain message characteristics, the routing logic could be included with this level of detail.

If you do not specify this parameter, medium level activities are recorded.

By default the WebSphere MQ display route application uses a temporary dynamic queue to store the returned messages. When the WebSphere MQ display route application ends, the temporary dynamic queue is closed, and any messages are purged. If the returned messages are required beyond the current execution of the WebSphere MQ display route application ends, then a permanent queue must be specified using the following parameters:

-rq `ReplyToQ`
: Specifies the name of the reply-to queue that all responses to the trace-route message are sent to. If the trace-route message is persistent, or if the *-n* parameter is specified, a reply-to queue must be specified that is **not** a temporary dynamic queue.

If you do not specify this parameter then a dynamic reply-to queue is created using the system default model queue, SYSTEM.DEFAULT.MODEL.QUEUE.

-rqm `ReplyToQMgr`
: Specifies the name of the queue manager where the reply-to queue resides. The name can contain up to 48 characters.

If you do not specify this parameter, the queue manager to which the WebSphere MQ display route application is connected is used as the reply-to queue manager.

How the trace-route message is handled:

There are various parameters that are used to control how a trace-route message is handled as it is routed through a queue manager network. The following parameters can restrict where the trace-route message can be routed in the queue manager network:

-d `Deliver`
: Specifies whether the trace-route message is to be delivered to the target queue on arrival. Possible values for *Deliver* are:

yes	On arrival, the trace-route message is put to the target queue, even if the queue manager does not support trace-route messaging.
no	On arrival, the trace-route message is **not** put to the target queue.

If you do not specify this parameter, the trace-route message is **not** put to the target queue.

-f `Forward`

Specifies the type of queue manager that the trace-route message can be forwarded to. Queue managers use an algorithm when determining whether to forward a message to a remote queue manager. For details of this algorithm, see "The TraceRoute PCF group" on page 176. The possible values for *Forward* are:

all	The trace-route message is forwarded to any queue manager. **Warning:** If forwarded to a WebSphere MQ queue manager prior to Version 6.0, the trace-route message will not be recognized and can be delivered to a local queue despite the value of the *-d Deliver* parameter.
supported	The trace-route message is only forwarded to a queue manager that will honor the *Deliver* parameter from the *TraceRoute* PCF group.

If you do not specify this parameter, the trace-route message will only be forwarded to a queue manager that will honor the *Deliver* parameter.

The following parameters can prevent a trace-route message from remaining in a queue manager network indefinitely:

-s `Activities`

Specifies the maximum number of recorded activities that can be performed on behalf of the trace-route message before it is discarded. This prevents the trace-route message from being forwarded indefinitely if caught in an infinite loop. The value of *Activities* is either greater than or equal to 1, or MQROUTE_UNLIMITED_ACTIVITIES. MQROUTE_UNLIMITED_ACTIVITIES specifies that an unlimited number of activities can be performed on behalf of the trace-route message.

If you do not specify this parameter, an unlimited number of activities can be performed on behalf of the trace-route message.

-xs `Expiry`

Specifies the expiry time for the trace-route message, in seconds.

If you do not specify this parameter, the expiry time is specified as 60 seconds.

-xp `PassExpiry`

Specifies whether the expiry time from the trace-route message is passed on to a trace-route reply message. Possible values for *PassExpiry* are:

yes	The report option MQRO_PASS_DISCARD_AND_EXPIRY is specified in the message descriptor of the trace-route message. If a trace-route reply message, or activity reports, are generated for the trace-route message, the MQRO_DISCARD report option (if specified), and the remaining expiry time are passed on. This is the default value.
no	The report option MQRO_PASS_DISCARD_AND_EXPIRY is not specified. If a trace-route reply message is generated for the trace-route message, the discard option and expiry time from the trace-route message are **not** passed on.

If you do not specify this parameter, MQRO_PASS_DISCARD_AND_EXPIRY is not specified.

The `discard` report option, specified using `-ro`
Specifies the MQRO_DISCARD_MSG report option. This can prevent the trace-route message remaining in the queue manager network indefinitely.

Displaying activity information

The WebSphere MQ display route application can display activity information for a trace-route message that it has just put into a queue manager network, or it can display activity information for a previously generated trace-route message.

To specify whether activity information returned for a trace-route message is displayed, specify the following parameter:

-n Specifies that activity information returned for the trace-route message is not to be displayed.

If this parameter is accompanied by a request for a trace-route reply message, (*-ar*), or any of the report generating options from (*-ro ReportOption*), then a specific (non-model) reply-to queue must be specified using *-rq ReplyToQ*. By default, only activity report messages are requested.

After the trace-route message is put to the specified target queue, a 48 character hexadecimal string is displayed containing the message identifier of the trace-route message. The message identifier can be used by the WebSphere MQ display route application to display the activity information for the trace-route message at a later time, using the *-i CorrelId* parameter.

If you do not specify this parameter, activity information returned for the trace-route message is displayed in the form specified by the *-v* parameter.

When displaying activity information for a trace-route message that has just been put into a queue manager network, the following parameter can be specified:

-w *WaitTime*
Specifies the time, in seconds, that the WebSphere MQ display route application will wait for activity reports, or a trace-route reply message, to return to the specified reply-to queue.

If you do not specify this parameter, the wait time is specified as the expiry time of the trace-route message, plus 60 seconds.

When displaying previously accumulated activity information the following parameters must be set:

-q *TargetQName*
: If the WebSphere MQ display route application is being used to view previously gathered activity information, *TargetQName* specifies the name of the queue where the activity information is stored.

-i *CorrelId*
: This parameter is used when the WebSphere MQ display route application is used to display previously accumulated activity information only. There can be many activity reports and trace-route reply messages on the queue specified by *-q TargetQName*. *CorrelId* is used to identify the activity reports, or a trace-route reply message, related to a trace-route message. Specify the message identifier of the original trace-route message in *CorrelId*.

  The format of *CorrelId* is a 48 character hexadecimal string.

The following parameters can be used when displaying previously accumulated activity information, or when displaying current activity information for a trace-route message:

-b Specifies that the WebSphere MQ display route application will only browse activity reports or a trace-route reply message related to a message. This allows activity information to be displayed again at a later time.

  If you do not specify this parameter, the WebSphere MQ display route application will destructively get activity reports or a trace-route reply message related to a message.

-v summary | all | none | outline *DisplayOption*

summary	The queues that the trace-route message was routed through are displayed.
all	All available information is displayed.
none	No information is displayed.

`outline DisplayOption` Specifies display options for the trace-route message. Multiple display options can be specified using a comma as a separator.

If no values are supplied the following is displayed:

- The application name
- The type of each operation
- Any operation specific parameters

Possible values for *DisplayOption* are:

activity All non-PCF group parameters in *Activity* PCF groups are displayed.

identifiers
Values with parameter identifiers MQBACF_MSG_ID or MQBACF_CORREL_ID are displayed. This overrides *msgdelta*.

message
All non-PCF group parameters in *Message* PCF groups are displayed. When this value is specified, you cannot specify *msgdelta*.

msgdelta
All non-PCF group parameters in *Message* PCF groups, that have changed since the last operation, are displayed. When this value is specified, you cannot specify *message*.

operation
All non-PCF group parameters in *Operation* PCF groups are displayed.

traceroute
All non-PCF group parameters in *TraceRoute* PCF groups are displayed.

If you do not specify this parameter, a summary of the message route is displayed.

Displaying additional information

As a trace-route message is routed through a queue manager network, user-written applications can record additional information by writing one or more additional PCF parameters to the message data of the trace-route message or to the message data of an activity report. For the WebSphere MQ display route application to display additional information in a readable form it must be recorded in a specific format detailed in "Recording additional information" on page 182.

Examples

The following examples show how you can use the WebSphere MQ display route application. In each example, there are two queue managers (QM1 and QM2) that are inter-connected by two channels (QM2.TO.QM1 and QM1.TO.QM2).

Example 1 - Requesting activity reports

In this example the WebSphere MQ display route application connects to queue manager, QM1, and is used to generate and deliver a trace-route message to the target queue, TARGET.Q, on remote queue manager, QM2. The necessary report option is specified so that activity reports are requested as the trace-route reply message is routed. On arrival at the target queue the trace-route message is discarded. Activity information returned to the WebSphere MQ display route application using activity reports is put in order and displayed.

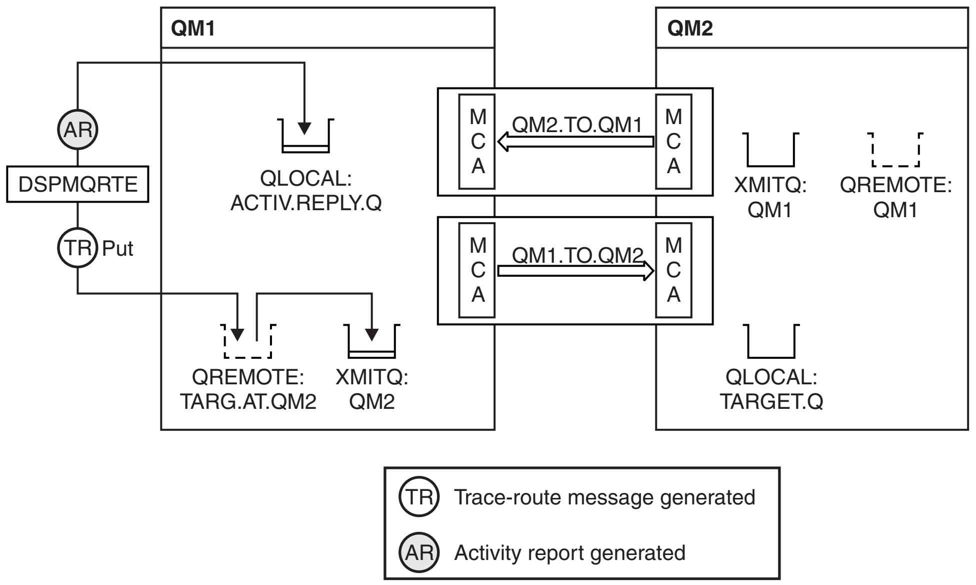


Figure 9. Requesting activity reports, Diagram 1

- The ACTIVREC attribute of each queue manager (QM1 and QM2) is set to MSG.
- The following command is issued:

  ```
  dspmqrte -m QM1 -q TARG.AT.QM2 -rq ACTIV.REPLY.Q
  ```

  QM1 is the name of the queue manager to which the WebSphere MQ display route application connects, TARG.AT.QM2 is the name of the target queue, and ACTIV.REPLY.Q is the name of the queue to which it is requested that all responses to the trace-route message are sent.

  Default values are assumed for all options that are not specified, but note in particular the -f option (the trace-route message is forwarded only to a queue manager that honors the Deliver parameter of the TraceRoute PCF group), the -d option (on arrival, the trace-route message is not put on the target queue), the -ro option (MQRO_ACTIVITY and MQRO_DISCARD_MSG report options are specified), and the -t option (medium detail level activity is recorded).
- DSPMQRTE generates the trace-route message and puts it on the remote queue TARG.AT.QM2.
- DSPMQRTE then looks at the value of the ACTIVREC attribute of queue manager QM1. The value is MSG, therefore DSPMQRTE generates an activity report and puts it on the reply queue ACTIV.REPLY.Q.

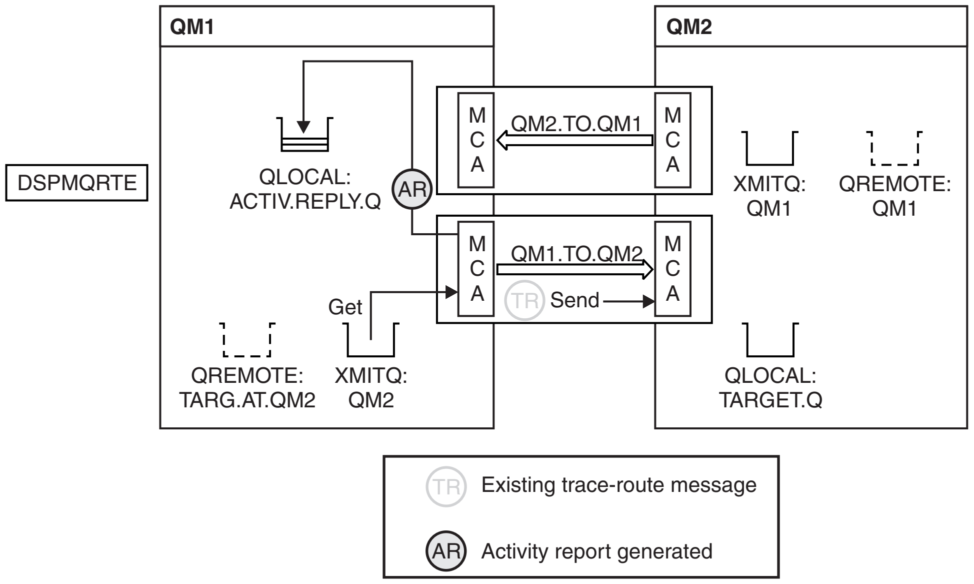


Figure 10. Requesting activity reports, Diagram 2

- The sending message channel agent (MCA) gets the trace-route message from the transmission queue. The message is a trace-route message, therefore the MCA begins to record the activity information.
- The ACTIVREC attribute of the queue manager (QM1) is MSG, and the MQRO_ACTIVITY option is specified in the Report field of the message descriptor, therefore the MCA will later generate an activity report. The RecordedActivities parameter value in the TraceRoute PCF group is incremented by 1.
- The MCA checks that the MaxActivities value in the TraceRoute PCF group has not been exceeded.
- Before the message is forwarded to QM2 the MCA follows the algorithm that is described in Forwarding (points 1 on page 179, 4 on page 180 and 5 on page 180) and the MCA chooses to send the message.
- The MCA then generates an activity report and puts it on the reply queue (ACTIV.REPLY.Q).

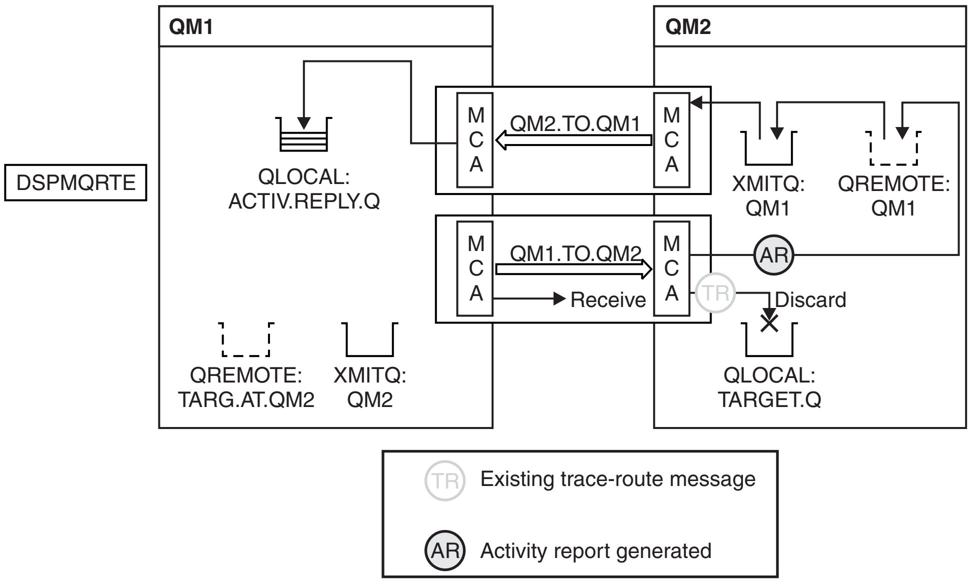


Figure 11. Requesting activity reports, Diagram 3

- The receiving MCA receives the trace-route message from the channel. The message is a trace-route message, therefore the MCA begins to record the information about the activity.
- If the queue manager that the trace-route message has come from is Version 5.3.1 or earlier, the MCA increments the DiscontinuityCount parameter of the TraceRoute PCF by 1. This is not the case here.
- The ACTIVREC attribute of the queue manager (QM2) is MSG, and the MQRO_ACTIVITY option is specified, therefore the MCA will generate an activity report. The RecordedActivities parameter value is incremented by 1.
- The target queue is a local queue, therefore the message is discarded with feedback MQFB_NOT_DELIVERED, in accordance with the Deliver parameter value in the TraceRoute PCF group.
- The MCA then generates the final activity report and puts it on the reply queue. This resolves to the transmission queue that is associated with queue manager QM1 and the activity report is returned to queue manager QM1 (ACTIV.REPLY.Q).

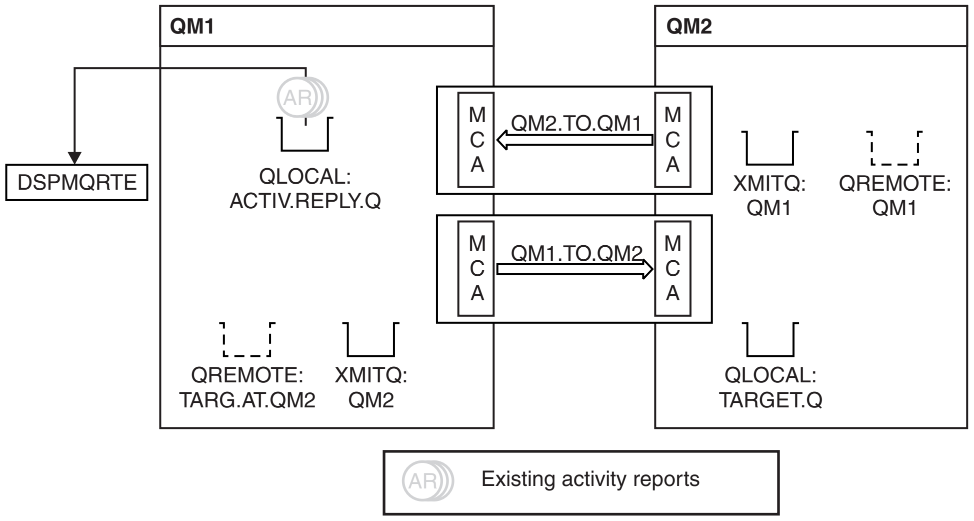


Figure 12. Requesting activity reports, Diagram 4

- Meanwhile, DSPMQRTE has been continually performing MQGETs on the reply queue (ACTIV.REPLY.Q), waiting for activity reports. It will wait for up to 120 seconds (60 seconds longer than the expiry time of the trace-route message) since -w was not specified when DSPMQRTE was started.
- DSPMQRTE gets the 3 activity reports off the reply queue.
- The activity reports are ordered using the RecordedActivities, UnrecordedActivities, and DiscontinuityCount parameters in the TraceRoute PCF group for each of the activities. The only value that is non-zero in this example is RecordedActivities, therefore this is the only parameter that is actually used.
- The program ends as soon as the discard operation is displayed. Even though the final operation was a discard, it is treated as though a put took place because the feedback is MQFB_NOT_DELIVERED.

  The output that is displayed follows:

```
AMQ8653: DSPMQRTE command started with options '-m QM1 -q TARG.AT.QM2
 -rq ACTIV.REPLY.Q'.
AMQ8659: DSPMQRTE command successfully put a message on queue 'QM2',
 queue manager 'QM1'.
AMQ8674: DSPMQRTE command is now waiting for information to display.
AMQ8666: Queue 'QM2' on queue manager 'QM1'.
AMQ8666: Queue 'TARGET.Q' on queue manager 'QM2'.
AMQ8652: DSPMQRTE command has finished.
```

Example 2 - Requesting a trace-route reply message

In this example the WebSphere MQ display route application connects to queue manager, QM1, and is used to generate and deliver a trace-route message to the target queue, TARGET.Q, on remote queue manager, QM2. The necessary option is specified so that activity information is accumulated in the trace-route message. On arrival at the target queue a trace-route reply message is requested, and the trace-route message is discarded.

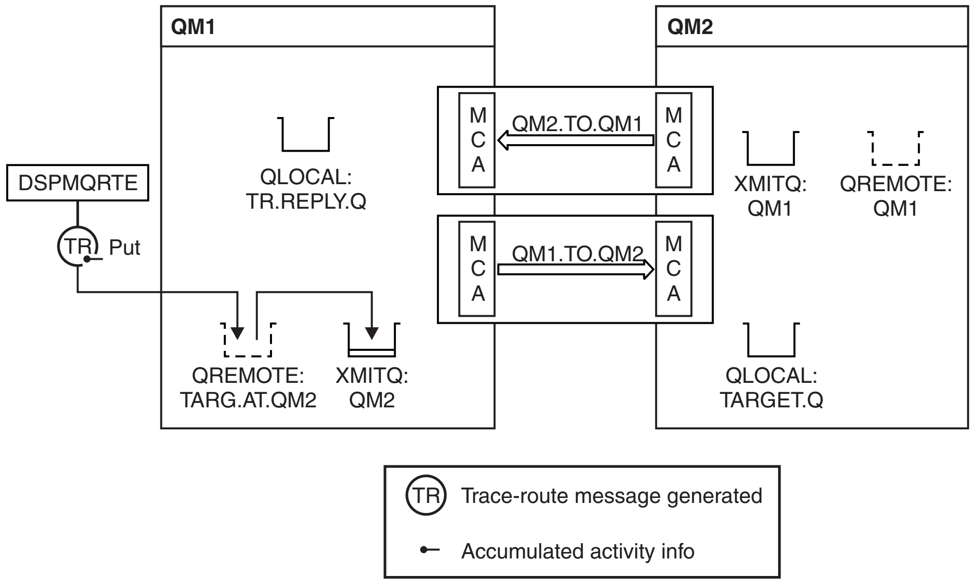


Figure 13. Requesting a trace-route reply message, Diagram 1

- The ROUTEREC attribute of each queue manager (QM1 and QM2) is set to MSG.
- The following command is issued:

  ```
  dspmqrte -m QM1 -q TARG.AT.QM2 -rq TR.REPLY.Q -ac -ar -ro discard
  ```

  QM1 is the name of the queue manager to which the WebSphere MQ display route application connects, TARG.AT.QM2 is the name of the target queue, and ACTIV.REPLY.Q is the name of the queue to which it is requested that all responses to the trace-route message are sent. The -ac option specifies that activity information is accumulated in the trace-route message, the -ar option specifies that all accumulated activity is sent to the reply-to queue that is specified by the -rq option (that is, TR.REPLY.Q). The -ro option specifies that report option MQRO_DISCARD_MSG is set which means that activity reports are not generated in this example.
- DSPMQRTE accumulates activity information in the trace-route message before the message is put on the target route. The queue manager attribute ROUTEREC must not be DISABLED for this to happen.

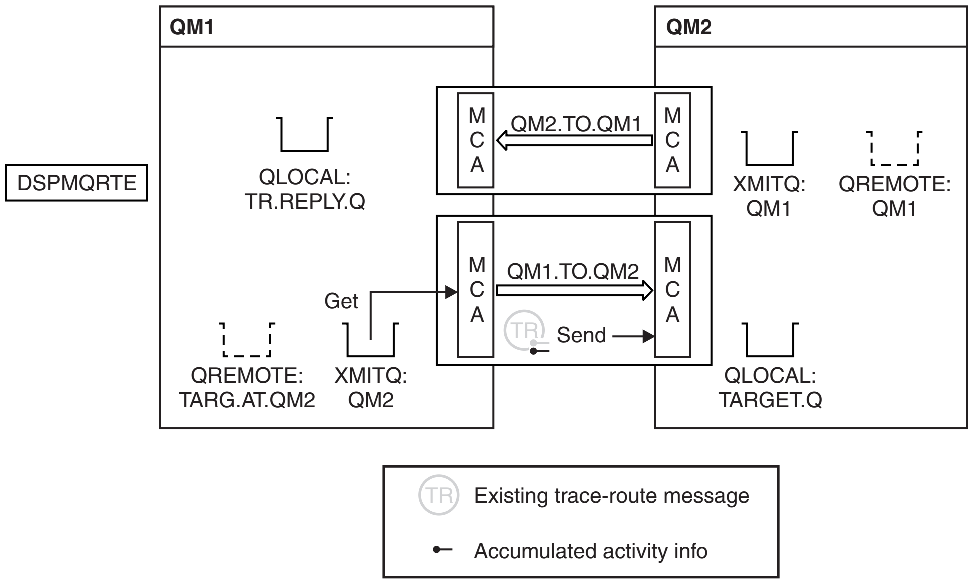


Figure 14. Requesting a trace-route reply message, Diagram 2

- The message is a trace-route message, therefore the sending MCA begins to record information about the activity.
- The queue manager attribute ROUTEREC on QM1 is not DISABLED, therefore the MCA accumulates the activity information within the message, before the message is forwarded to queue manager QM2.

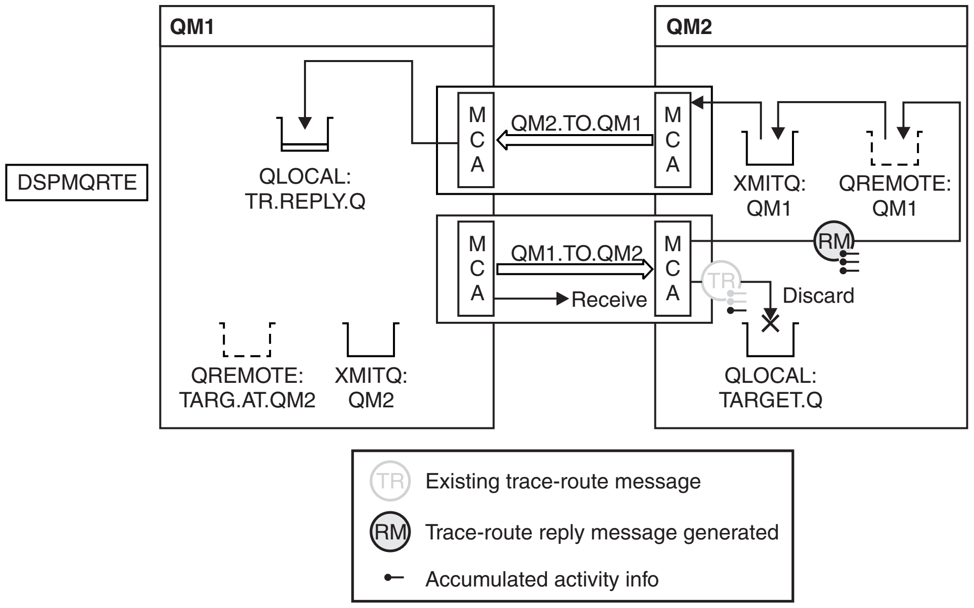


Figure 15. Requesting a trace-route reply message, Diagram 3

- The message is a trace-route message, therefore the receiving MCA begins to record information about the activity.

- The queue manager attribute ROUTEREC on QM2 is not DISABLED, therefore the MCA accumulates the information within the message.
- The target queue is a local queue, therefore the message is discarded with feedback MQFB_NOT_DELIVERED, in accordance with the Deliver parameter value in the TraceRoute PCF group.
- This is the last activity that will take place on the message, and because the queue manager attribute ROUTEREC on QM1 is not DISABLED, the MCA generates a trace-route reply message in accordance with the Accumulate value. The value of ROUTEREC is MSG, therefore the reply message is put on the reply queue. The reply message contains all the accumulated activity information from the trace-route message.

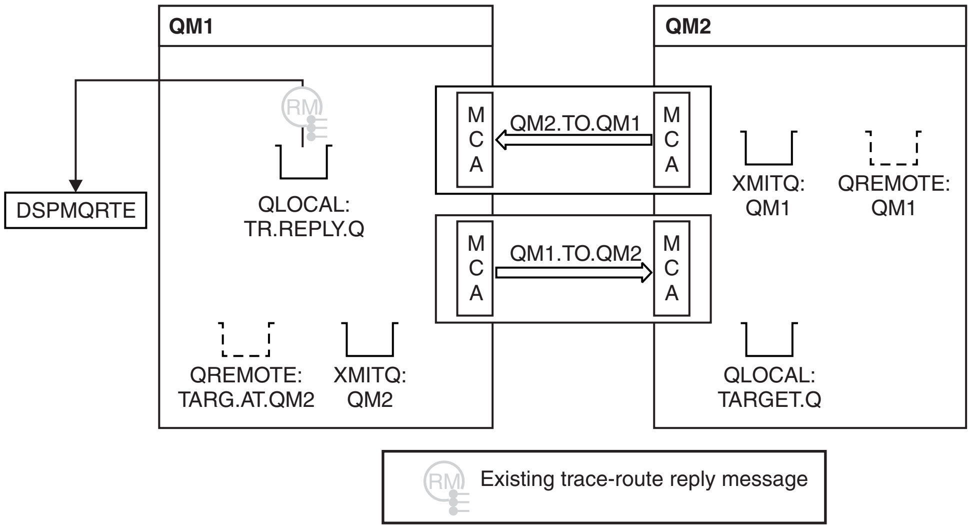


Figure 16. Requesting a trace-route reply message, Diagram 4

- Meanwhile DSPMQRTE is waiting for the trace-route reply message to return to the reply queue. When it returns, DSPMQRTE parses each activity that it contains and prints it out. The final operation is a discard operation. DSPMQRTE ends after it has been printed.

The output that is displayed follows:

```
AMQ8653: DSPMQRTE command started with options '-m QM1 -q TARG.AT.QM2 -rq
 TR.REPLY.Q'.
AMQ8659: DSPMQRTE command successfully put a message on queue 'QM2', queue
 manager 'QM1'.
AMQ8674: DSPMQRTE command is now waiting for information to display.
AMQ8666: Queue 'QM2' on queue manager 'QM1'.
AMQ8666: Queue 'TARGET.Q' on queue manager 'QM2'.
AMQ8652: DSPMQRTE command has finished.
```

Example 3 - Delivering activity reports to the system queue

This example is the same as “Example 1 - Requesting activity reports” on page 194, except that QM2 now has the value of the ACTIVREC queue manage attribute set to QUEUE. Channel QM1.TO.QM2 must have been restarted for this to take effect.

This example demonstrates how to detect when activity reports are delivered to queues other than the reply-to queue. Once detected, the WebSphere MQ display route application is used to read activity reports from another queue.

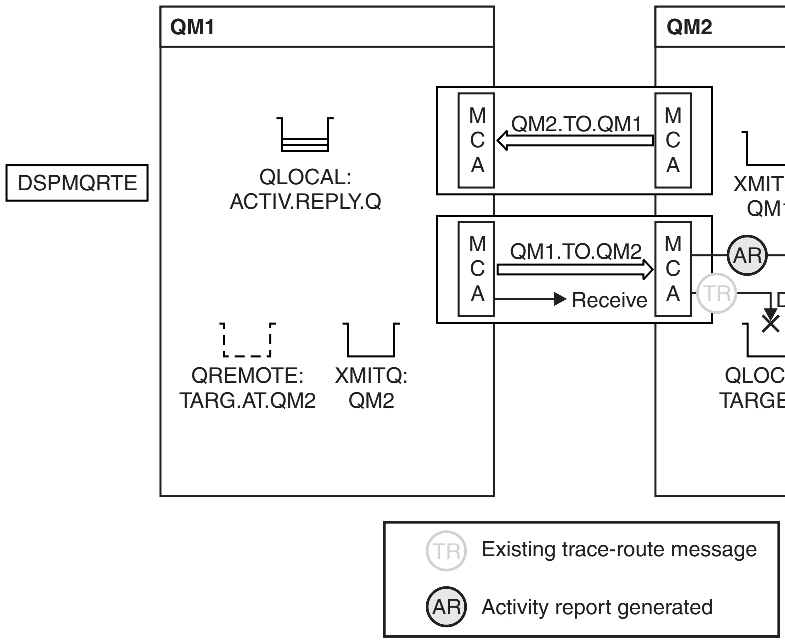


Figure 17. Delivering activity reports to the system queue, Diagram 1

- The message is a trace-route message, therefore the receiving MCA begins to record information about the activity.
- The value of the ACTIVREC queue manager attribute on QM2 is now QUEUE, therefore the MCA generates an activity report, but puts it on the system queue (SYSTEM.ADMIN.ACTIVITY.QUEUE) and not on the reply queue (ACTIV.REPLY.Q).

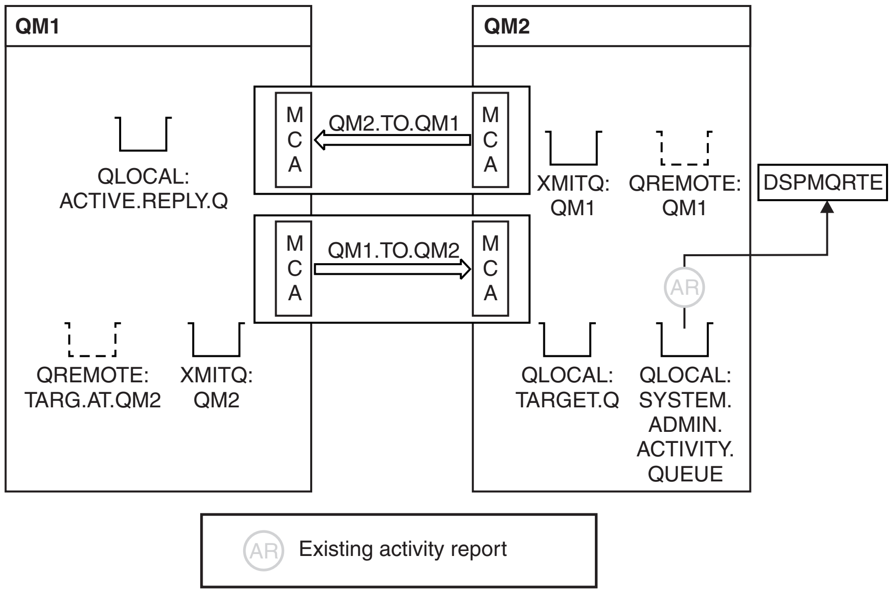


Figure 18. Delivering activity reports to the system queue, Diagram 2

- Meanwhile DSPMQRTE has been waiting for activity reports to arrive on ACTIV.REPLY.Q. Only two arrive. DSPMQRTE continues waiting for the whole 120 seconds because it seems that the route is not yet complete.

The output that is displayed follows:

```
AMQ8653: DSPMQRTE command started with options '-m QM1 -q TARG.AT.QM2 -rq
         ACTIV.REPLY.Q -v outline identifiers'.
AMQ8659: DSPMQRTE command successfully put a message on queue 'QM2', queue
         manager 'QM1'.
AMQ8674: DSPMQRTE command is now waiting for information to display.
--------------------------------------------------------------------------------
Activity:
 ApplName: 'cann\output\bin\dspmqrte.exe'

 Operation:
  OperationType: Put

  Message:

   MQMD:
    MsgId: X'414D51204C415247455120202020202020A3C9154220001502'
    CorrelId: X'414D51204C415247455120202020202020A3C9154220001503'
  QMgrName: 'QM1                                             '
  QName: 'TARG.AT.QM2                                     '
  ResolvedQName: 'QM2                                             '
  RemoteQName: 'TARGET.Q                                        '
  RemoteQMgrName: 'QM2                                             '
--------------------------------------------------------------------------------
Activity:
 ApplName: 'cann\output\bin\runmqchl.EXE'

 Operation:
  OperationType: Get

  Message:

   MQMD:
    MsgId: X'414D51204C415247455120202020202020A3C9154220001505'
    CorrelId: X'414D51204C415247455120202020202020A3C9154220001502'

   EmbeddedMQMD:
    MsgId: X'414D51204C415247455120202020202020A3C9154220001502'
    CorrelId: X'414D51204C415247455120202020202020A3C9154220001503'
  QMgrName: 'QM1                                             '
  QName: 'QM2                                             '
  ResolvedQName: 'QM2                                             '

 Operation:
  OperationType: Send

  Message:

   MQMD:
    MsgId: X'414D51204C415247455120202020202020A3C9154220001502'
    CorrelId: X'414D51204C415247455120202020202020A3C9154220001503'
  QMgrName: 'QM1                                             '
  RemoteQMgrName: 'QM2                                             '
  ChannelName: 'QM1.TO.QM2          '
  ChannelType: Sender
  XmitQName: 'QM2                                             '
--------------------------------------------------------------------------------
AMQ8652: DSPMQRTE command has finished.
```

- The last operation that DSPMQRTE observed was a Send, therefore the channel is running. Now we must work out why we did not receive any more activity reports from queue manager QM2 (as identified in RemoteQMgrName).
- To check whether there is any activity information on the system queue, start DSPMQRTE on QM2 to try and collect more activity reports. Use the following command to start DSPMQRTE:

  ```
  dspmqrte -m QM2 -q SYSTEM.ADMIN.ACTIVITY.QUEUE
           -i 414D51204C4152474551202020202020A3C9154220001502 -v outline
  ```

  where 414D51204C4152474551202020202020A3C9154220001502 is the MsgId of the trace-route message that was put.
- DSPMQRTE then performs a sequence of MQGETs again, waiting for responses on the system activity queue related to the trace-route message with the specified identifier.
- DSPMQRTE gets one more activity report, which it displays. DSPMQRTE determines that the preceding activity reports are missing, and displays a message saying this. We already know about this part of the route, however.

The output that is displayed follows:

```
AMQ8653: DSPMQRTE command started with options '-m QM2
         -q SYSTEM.ADMIN.ACTIVITY.QUEUE
         -i 414D51204C4152474551202020202020A3C915420001502 -v outline'.
AMQ8674: DSPMQRTE command is now waiting for information to display.
--------------------------------------------------------------------------------

Activity:
 Activity information unavailable.

--------------------------------------------------------------------------------
Activity:
 ApplName: 'cann\output\bin\AMQRMPPA.EXE'

 Operation:
  OperationType: Receive
  QMgrName: 'QM2                                             '
  RemoteQMgrName: 'QM1                                             '
  ChannelName: 'QM1.TO.QM2          '
  ChannelType: Receiver

 Operation:
  OperationType: Discard
  QMgrName: 'QM2                                             '
  QName: 'TARGET.Q                                        '
  Feedback: NotDelivered

--------------------------------------------------------------------------------
AMQ8652: DSPMQRTE command has finished.
```

- This activity report indicates that the route information is now complete. No problem occurred.
- Just because route information is unavailable, or because DSPMQRTE cannot display all of the route, this does not mean that the message was not delivered. For example, the queue manager attributes of different queue managers might be different, or a reply queue might not be defined to get the response back. See “Circumstances where activity information is not acquired” on page 168.

Example 4 - Diagnosing a channel problem

In this example the WebSphere MQ display route application connects to queue manager, QM1, generates a trace-route message, then attempts to deliver it to the target queue, TARGET.Q, on remote queue manager, QM2. In this example the trace-route message does not reach the target queue. The available activity report is used to diagnose the problem.

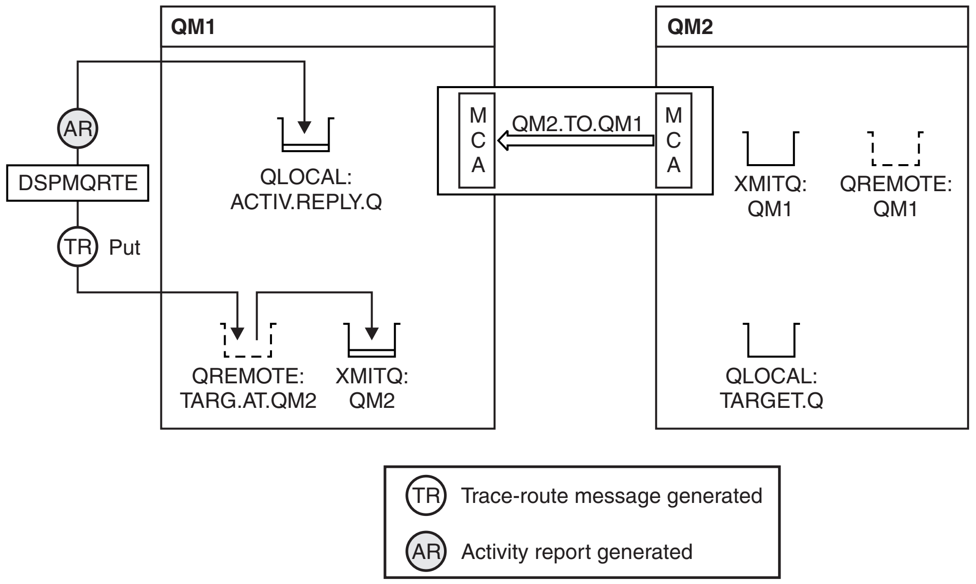


Figure 19. Diagnosing a channel problem

- In this example, the channel QM1.TO.QM2 is not running.
- DSPMQRTE puts a trace-route message (as in example 1) to the target queue and generates an activity report.
- There is no MCA to get the message from the transmission queue (QM2), therefore this is the only activity report that DSPMQRTE gets back from the reply queue. This time the fact that the route is not complete does indicate a problem. The administrator can use the transmission queue found in ResolvedQName to investigate why the transmission queue is not being serviced.

The output that is displayed follows:

```
AMQ8653: DSPMQRTE command started with options '-m QM1 -q TARG.AT.QM2
         -rq ACTIV.REPLY.Q -v outline'.
AMQ8659: DSPMQRTE command successfully put a message on queue 'QM2',
         queue manager 'QM1'.
AMQ8674: DSPMQRTE command is now waiting for information to display.
--------------------------------------------------------------------------------
Activity:
 ApplName: 'cann\output\bin\dspmqrte.exe'

 Operation:
  OperationType: Put
  QMgrName: 'QM1                                             '
  QName: 'TARG.AT.QM2                                    '
  ResolvedQName: 'QM2                                            '
  RemoteQName: 'TARGET.Q                                       '
  RemoteQMgrName: 'QM2                                            '

--------------------------------------------------------------------------------
 AMQ8652: DSPMQRTE command has finished.
```

Activity report reference

This chapter provides an overview of the activity report message format. It describes the information returned in activity reports, including returned parameters.

Activity report format

Activity reports are standard WebSphere MQ report messages containing a message descriptor and message data.

Activity reports are PCF messages generated by applications that have performed an activity on behalf of a message as it has been routed through a queue manager network. For information on activity recording see, "Activity recording" on page 163.

Activity reports contain the following:

A message descriptor
An MQMD structure

Message data
Consists of the following:
- An embedded PCF header (MQEPH).
- Activity report message data.

Activity report message data consists of the *Activity* PCF group and, if generated for a trace-route message, the *TraceRoute* PCF group.

Table 17 shows the structure of these reports, including parameters that are only returned under certain conditions.

Table 17. Activity report format

Message descriptor	Message data

Table 17. Activity report format (continued)

MQMD structure	Embedded PCF header MQEPH structure	Activity report message data
Structure identifier	Structure identifier	Activity
Structure version	Structure version	Activity application name
Report options	Structure length	Activity application type
Message type	Encoding	Activity description
Expiration time	Coded character set ID	Operation
Feedback	Message format	Operation type
Encoding	Flags	Operation date
Coded character set ID	PCF header (MQCFH)	Operation time
Message format	Structure type	Message
Priority	Structure length	Message length
Persistence	Structure version	MQMD [8]
Message identifier	Command identifier	EmbeddedMQMD
Correlation identifier	Message sequence number	Queue manager name
Backout count	Control options	Queue sharing group name
Reply-to queue	Completion code	Queue name [1 2 3 7]
Reply-to queue manager	Reason code	Resolved queue name [1 3 7]
User identifier	Parameter count	Remote queue name [3 7]
Accounting token		Remote queue manager name [2 3 4 5 7]
Application identity data		Subscription level [9]
Application type		Subscription identifier [9]
Application name		Feedback [2 10]
Put date		Channel name [4 5]
Put time		Channel type [4 5]
Application origin data		Transmission queue name [5]
Group identifier		TraceRoute [6]
Message sequence number		Detail
Offset		Recorded activities
Message flags		Unrecorded activities
Original length		Discontinuity count
		Max activities
		Accumulate
		Deliver

Note:

1. Returned for Get and Browse operations.
2. Returned for Discard operations.
3. Returned for Put, Put Reply, and Put Report operations.
4. Returned for Receive operations.
5. Returned for Send operations.
6. Returned for trace-route messages.
7. Not returned for Put operations to a topic, contained within Publish activities.
8. Not returned for Excluded Publish operations. For Publish and Discarded Publish operations, returned containing a subset of parameters.
9. Returned for Publish, Discarded Publish, and Excluded Publish operations.
10. Returned for Discarded Publish and Excluded Publish operations.

Activity report MQMD (message descriptor)

For an activity report, the MQMD structure contains these values:

StrucId

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

Version

Description: Structure version number.
Datatype: MQLONG.
Values: Copied from the original message descriptor. Possible values are:

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

MQMD_VERSION_2
Version-2 message descriptor structure, supported on AIX, HP-UX, z/OS, HP OpenVMS, i5/OS, Solaris, Linux, Windows, and all WebSphere MQ clients connected to these systems.

Report

Description: Options for further report messages.
Datatype: MQLONG.
Value: If MQRO_PASS_DISCARD_AND_EXPIRY or MQRO_DISCARD_MSG were specified in the *Report* field of the original message descriptor:

MQRO_DISCARD
The report is discarded if it cannot be delivered to the destination queue.

Otherwise:

MQRO_NONE
No reports required.

MsgType

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

Expiry

Description: Report message lifetime.
Datatype: MQLONG.
Value: If the *Report* field in the original message descriptor is specified as MQRO_PASS_DISCARD_AND_EXPIRY, the remaining expiry time from the original message is used.

Otherwise:

MQEI_UNLIMITED
The report does not have an expiry time.

Feedback

Description: Feedback or reason code.
Datatype: MQLONG.

Value:	**MQFB_ACTIVITY** Activity report.

Encoding

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

CodedCharSetId

Description:	Character set identifier of report message data.
Datatype:	MQLONG.
Value:	Set as appropriate.

Format

Description:	Format name of report message data
Datatype:	MQCHAR8.
Value:	**MQFMT_EMBEDDED_PCF** Embedded PCF message.

Priority

Description:	Report message priority.
Datatype:	MQLONG.
Value:	Copied from the original message descriptor.

Persistence

Description:	Report message persistence.
Datatype:	MQLONG.
Value:	Copied from the original message descriptor.

MsgId

Description:	Message identifier.
Datatype:	MQBYTE24.
Values:	If the *Report* field in the original message descriptor is specified as MQRO_PASS_MSG_ID, the message identifier from the original message is used. Otherwise, a unique value will be generated by the queue manager.

CorrelId

Description:	Correlation identifier.
Datatype:	MQBYTE24.
Value:	If the *Report* field in the original message descriptor is specified as MQRO_PASS_CORREL_ID, the correlation identifier from the original message is used. Otherwise, the message identifier is copied from the original message.

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 that generated the report message.

UserIdentifier

Description:	The user identifier of the application that generated the report message.
Datatype:	MQCHAR12.
Value:	Copied from the original message descriptor.

AccountingToken

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

ApplIdentityData

Description:	Application data relating to identity.
Datatype:	MQCHAR32.
Values:	Copied from the original message descriptor.

PutApplType

Description:	Type of application that put the report message.
Datatype:	MQLONG.
Value:	**MQAT_QMGR** Queue manager generated message.

PutApplName

Description:	Name of application that put the report message.
Datatype:	MQCHAR28.
Value:	Either the first 28 bytes of the queue manager name, or the name of the MCA that generated the report message.

PutDate

Description:	Date when message was put.
Datatype:	MQCHAR8.

Value:	As generated by the queue manager.

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.

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:	Copied from the original message descriptor.

MsgSeqNumber

Description:	Sequence number of logical message within group.
Datatype:	MQLONG.
Value:	Copied from the original message descriptor.

Offset

Description:	Offset of data in physical message from start of logical message.
Datatype:	MQLONG.
Value:	Copied from the original message descriptor.

MsgFlags

Description:	Message flags that specify attributes of the message or control its processing.
Datatype:	MQLONG.
Value:	Copied from the original message descriptor.

OriginalLength

Description:	Length of original message.
Datatype:	MQLONG.
Value:	Copied from the original message descriptor.

Activity report MQEPH (Embedded PCF header)

The MQEPH structure contains a description of both the PCF information that accompanies the message data of an activity report, and the application message data that follows it. For a full description of the MQEPH structure, including a description of the elementary datatype of each parameter, see “MQEPH - Embedded PCF header” on page 330.

For an activity report, the MQEPH structure contains the following values:

StrucId

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

Version

Description:	Structure version number.
Datatype:	MQLONG.
Values:	MQEPH_VERSION_1.

StrucLength

Description:	Structure length.
Datatype:	MQLONG.
Value:	Total length of the structure including the PCF parameter structures that follow it.

Encoding

Description:	Numeric encoding of the message data that follows the last PCF parameter structure.
Datatype:	MQLONG.
Value:	If any data from the original application message data is included in the report message, the value will be copied from the *Encoding* field of the original message descriptor. Otherwise, 0.

CodedCharSetId

Description:	Character set identifier of the message data that follows the last PCF parameter structure.
Datatype:	MQLONG.
Value:	If any data from the original application message data is included in the report message, the value will be copied from the *CodedCharSetId* field of the original message descriptor. Otherwise, MQCCSI_UNDEFINED.

Format

Description:	Format name of message data that follows the last PCF parameter structure.
Datatype:	MQCHAR8.
Value:	If any data from the original application message data is included in the report message, the value will be copied from the *Format* field of the original message descriptor. Otherwise, MQFMT_NONE.

Flags

Description:	Flags that specify attributes of the structure or control its processing.
Datatype:	MQLONG.
Value:	**MQEPH_CCSID_EMBEDDED** Specifies that the character set of the parameters containing character data is specified individually within the *CodedCharSetId* field in each structure.

PCFHeader

Description:	Programmable Command Format Header
Datatype:	MQCFH.
Value:	See "Activity report MQCFH (PCF header)."

Activity report MQCFH (PCF header)

The MQCFH structure describes the PCF information available in the activity report. For a full description of MQCFH, including a description of the elementary datatype of each parameter, see "MQCFH - PCF header" on page 314.

For an activity report, the MQCFH structure contains the following values:

Type

Description:	Structure type that identifies the content of the report message.
Datatype:	MQLONG.
Value:	**MQCFT_REPORT** Message is a report.

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_3

Command

Description:	Command identifier. This identifies the category of the message.
Datatype:	MQLONG.
Values:	**MQCMD_ACTIVITY_MSG** Message activity.

MsgSeqNumber

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

Datatype:	MQLONG.
Values:	1.

Control

Description:	Control options.
Datatype:	MQLONG.
Values:	MQCFC_LAST.

CompCode

Description:	Completion code.
Datatype:	MQLONG.
Values:	MQCC_OK.

Reason

Description:	Reason code qualifying completion code.
Datatype:	MQLONG.
Values:	MQRC_NONE.

ParameterCount

Description:	Count of parameter structures. This is the number of parameter structures that follow the MQCFH structure. A group structure (MQCFGR), and its included parameter structures, are counted as one structure only.
Datatype:	MQLONG.
Values:	1 or greater.

Activity report message data

Activity report message data contains the *Activity* PCF group, which contains the parameters listed here.

Activity report message data consists of the *Activity* PCF group and, if generated for a trace-route message, the *TraceRoute* PCF group. The *Activity* PCF group is detailed in this topic. For details of the *TraceRoute* PCF group, see “Trace-route message data” on page 238.

There are certain parameters, not listed here, that are returned only when specific operations have been performed. For details of these parameters, see “Operation-specific activity report message data” on page 225.

For an activity report, the activity report message data contains the following parameters:

Activity

Description:	Grouped parameters describing the activity.
Identifier:	MQGACF_ACTIVITY.
Datatype:	MQCFGR.
Included in PCF group:	None.

Parameters in PCF group:	*ActivityApplName* *ActivityApplType* *ActivityDescription* *Operation* *TraceRoute*
Returned:	Always.

ActivityApplName

Description:	Name of application that performed the activity.
Identifier:	MQCACF_APPL_NAME.
Datatype:	MQCFST.
Included in PCF group:	*Activity*.
Maximum length:	MQ_APPL_NAME_LENGTH.
Returned:	Always.

ActivityApplType

Description:	Type of application that performed the activity.
Identifier:	MQIA_APPL_TYPE.
Datatype:	MQCFIN.
Included in PCF group:	*Activity*.
Returned:	Always.

ActivityDescription

Description:	Description of activity performed by the application.
Identifier:	MQCACF_ACTIVITY_DESCRIPTION.
Datatype:	MQCFST.
Included in PCF group:	*Activity*.
Maximum length:	64
Returned:	Always.

Operation

Description:	Grouped parameters describing an operation of the activity.
Identifier:	MQGACF_OPERATION.
Datatype:	MQCFGR.
Included in PCF group:	*Activity*.
Parameters in PCF group:	*OperationType* *OperationDate* *OperationTime* *Message* *QMgrName* *QSGName* **Note:** Additional parameters are returned in this group depending on the operation type. For information on the additional parameters, see "Operation-specific activity report message data" on page 225.
Returned:	One *Operation* PCF group per operation in the activity.

OperationType

Description:	Type of operation performed.
Identifier:	MQIACF_OPERATION_TYPE.
Datatype:	MQCFIN.
Included in PCF group:	*Operation*.
Values:	MQOPER_*.
Returned:	Always.

OperationDate

Description:	Date when the operation was performed.
Identifier:	MQCACF_OPERATION_DATE.
Datatype:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_DATE_LENGTH.
Returned:	Always.

OperationTime

Description:	Time when the operation was performed.
Identifier:	MQCACF_OPERATION_TIME.
Datatype:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_TIME_LENGTH.
Returned:	Always.

Message

Description:	Grouped parameters describing the message that caused the activity.
Identifier:	MQGACF_MESSAGE.
Datatype:	MQCFGR.
Included in PCF group:	*Operation*.
Parameters in group:	*MsgLength* *MQMD* *EmbeddedMQMD*
Returned:	Always, except for Excluded Publish operations.

MsgLength

Description:	Length of the message that caused the activity, before the activity occurred.
Identifier:	MQIACF_MSG_LENGTH.
Datatype:	MQCFIN.
Included in PCF group:	*Message*.
Returned:	Always.

MQMD

Description:	Grouped parameters related to the message descriptor of the message that caused the activity.
Identifier:	MQGACF_MQMD.
Datatype:	MQCFGR.
Included in PCF group:	*Message*.
Parameters in group:	*StrucId* *Version* *Report* *MsgType* *Expiry* *Feedback* *Encoding* *CodedCharSetId* *Format* *Priority* *Persistence* *MsgId* *CorrelId* *BackoutCount* *ReplyToQ* *ReplyToQMgr* *UserIdentifier* *AccountingToken* *ApplIdentityData* *PutApplType* *PutApplName* *PutDate* *PutTime* *ApplOriginData* *GroupId* *MsgSeqNumber* *Offset* *MsgFlags* *OriginalLength*
Returned:	Always, except for Excluded Publish operations.

EmbeddedMQMD

Description:	Grouped parameters describing the message descriptor embedded within a message on a transmission queue.
Identifier:	MQGACF_EMBEDDDED_MQMD.
Datatype:	MQCFGR.
Included in PCF group:	*Message*.

Parameters in group:	*StrucId* *Version* *Report* *MsgType* *Expiry* *Feedback* *Encoding* *CodedCharSetId* *Format* *Priority* *Persistence* *MsgId* *CorrelId* *BackoutCount* *ReplyToQ* *ReplyToQMgr* *UserIdentifier* *AccountingToken* *ApplIdentityData* *PutApplType* *PutApplName* *PutDate* *PutTime* *ApplOriginData* *GroupId* *MsgSeqNumber* *Offset* *MsgFlags* *OriginalLength*
Returned:	For Get operations where the queue resolves to a transmission queue.

StrucId

Description:	Structure identifier
Identifier:	MQCACF_STRUC_ID.
Datatype:	MQCFST.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	4.
Returned:	Always, except for Excluded Publish operations and in MQMD for Publish and Discarded Publish operations.

Version

Description:	Structure version number.
Identifier:	MQIACF_VERSION.
Datatype:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	Always, except for Excluded Publish operations and in MQMD for Publish and Discarded Publish operations.

Report

Description:	Options for report messages.
Identifier:	MQIACF_REPORT.
Datatype:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	Always, except for Excluded Publish operations and in MQMD for Publish and Discarded Publish operations.

MsgType

Description:	Indicates type of message.
Identifier:	MQIACF_MSG_TYPE.
Datatype:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	Always, except for Excluded Publish operations and in MQMD for Publish and Discarded Publish operations.

Expiry

Description:	Message lifetime.
Identifier:	MQIACF_EXPIRY.
Datatype:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	Always, except for Excluded Publish operations and in MQMD for Publish and Discarded Publish operations.

Feedback

Description:	Feedback or reason code.
Identifier:	MQIACF_FEEDBACK.
Datatype:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	Always, except for Excluded Publish operations and in MQMD for Publish and Discarded Publish operations.

Encoding

Description:	Numeric encoding of message data.
Identifier:	MQIACF_ENCODING.
Datatype:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	Always, except for Excluded Publish operations and in MQMD for Publish and Discarded Publish operations.

CodedCharSetId

Description:	Character set identifier of message data.
Identifier:	MQIA_CODED_CHAR_SET_ID.
Datatype:	MQCFIN.

Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	Always, except for Excluded Publish operations and in MQMD for Publish and Discarded Publish operations.

Format

Description:	Format name of message data
Identifier:	MQCACH_FORMAT_NAME.
Datatype:	MQCFST.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	MQ_FORMAT_LENGTH.
Returned:	Always, except for Excluded Publish operations.

Priority

Description:	Message priority.
Identifier:	MQIACF_PRIORITY.
Datatype:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	Always, except for Excluded Publish operations.

Persistence

Description:	Message persistence.
Identifier:	MQIACF_PERSISTENCE.
Datatype:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	Always, except for Excluded Publish operations.

MsgId

Description:	Message identifier.
Identifier:	MQBACF_MSG_ID.
Datatype:	MQCFBS.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	MQ_MSG_ID_LENGTH.
Returned:	Always, except for Excluded Publish operations.

CorrelId

Description:	Correlation identifier.
Identifier:	MQBACF_CORREL_ID.
Datatype:	MQCFBS.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	MQ_CORREL_ID_LENGTH.
Returned:	Always, except for Excluded Publish operations.

BackoutCount

Description:	Backout counter.
Identifier:	MQIACF_BACKOUT_COUNT.
Datatype:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	Always, except for Excluded Publish operations and in MQMD for Publish and Discarded Publish operations.

ReplyToQ

Description:	Name of reply queue.
Identifier:	MQCACF_REPLY_TO_QUEUE.
Datatype:	MQCFST.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always, except for Excluded Publish Operations and in MQMD for Publish and Discarded Publish operations.

ReplyToQMgr

Description:	Name of reply queue manager.
Identifier:	MQCACF_REPLY_TO_Q_MGR.
Datatype:	MQCFST.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always, except for Excluded Publish Operations and in MQMD for Publish and Discarded Publish Operations.

UserIdentifier

Description:	The user identifier of the application that originated the message.
Identifier:	MQCACF_USER_IDENTIFIER.
Datatype:	MQCFST.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	MQ_USER_ID_LENGTH.
Returned:	Always, except for Excluded Publish Operations.

AccountingToken

Description:	Accounting token that allows an application to charge for work done as a result of the message.
Identifier:	MQBACF_ACCOUNTING_TOKEN.
Datatype:	MQCFBS.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	MQ_ACCOUNTING_TOKEN_LENGTH.
Returned:	Always, except for Excluded Publish Operations.

ApplIdentityData

Description:	Application data relating to identity.

Identifier:	MQCACF_APPL_IDENTITY_DATA.
Datatype:	MQCFST.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	MQ_APPL_IDENTITY_DATA_LENGTH.
Returned:	Always, except for Excluded Publish Operations.

PutApplType

Description:	Type of application that put the message.
Identifier:	MQIA_APPL_TYPE.
Datatype:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	Always, except for Excluded Publish Operations and in MQMD for Publish and Discarded Publish Operations.

PutApplName

Description:	Name of application that put the message.
Identifier:	MQCACF_APPL_NAME.
Datatype:	MQCFST.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	MQ_APPL_NAME_LENGTH.
Returned:	Always, except for Excluded Publish Operations and in MQMD for Publish and Discarded Publish Operations.

PutDate

Description:	Date when message was put.
Identifier:	MQCACF_PUT_DATE.
Datatype:	MQCFST.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	MQ_PUT_DATE_LENGTH.
Returned:	Always, except for Excluded Publish Operations and in MQMD for Publish and Discarded Publish Operations.

PutTime

Description:	Time when message was put.
Identifier:	MQCACF_PUT_TIME.
Datatype:	MQCFST.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	MQ_PUT_TIME_LENGTH.
Returned:	Always, except for Excluded Publish Operations and in MQMD for Publish and Discarded Publish Operations.

ApplOriginData

Description:	Application data relating to origin.
Identifier:	MQCACF_APPL_ORIGIN_DATA.
Datatype:	MQCFST.

Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	MQ_APPL_ORIGIN_DATA_LENGTH.
Returned:	Always, except for Excluded Publish Operations and in MQMD for Publish and Discarded Publish Operations.

GroupId

Description:	Identifies to which message group or logical message the physical message belongs.
Identifier:	MQBACF_GROUP_ID.
Datatype:	MQCFBS.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Maximum length:	MQ_GROUP_ID_LENGTH.
Returned:	If the *Version* is specified as MQMD_VERSION_2. Not returned in Excluded Publish Operations and in MQMD for Publish and Discarded Publish Operations.

MsgSeqNumber

Description:	Sequence number of logical message within group.
Identifier:	MQIACH_MSG_SEQUENCE_NUMBER.
Datatype:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	If *Version* is specified as MQMD_VERSION_2. Not returned in Excluded Publish Operations and in MQMD for Publish and Discarded Publish Operations.

Offset

Description:	Offset of data in physical message from start of logical message.
Identifier:	MQIACF_OFFSET.
Datatype:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	If *Version* is specified as MQMD_VERSION_2. Not returned in Excluded Publish Operations and in MQMD for Publish and Discarded Publish Operations.

MsgFlags

Description:	Message flags that specify attributes of the message or control its processing.
Identifier:	MQIACF_MSG_FLAGS.
Datatype:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	If *Version* is specified as MQMD_VERSION_2. Not returned in Excluded Publish Operations and in MQMD for Publish and Discarded Publish Operations.

OriginalLength

Description:	Length of original message.

Identifier:	MQIACF_ORIGINAL_LENGTH.
Datatype:	MQCFIN.
Included in PCF group:	*MQMD* or *EmbeddedMQMD*.
Returned:	If *Version* is specified as MQMD_VERSION_2. Not returned in Excluded Publish Operations and in MQMD for Publish and Discarded Publish Operations.

QMgrName

Description:	Name of the queue manager where the activity was performed.
Identifier:	MQCA_Q_MGR_NAME.
Datatype:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_Q_MGR_NAME_LENGTH
Returned:	Always.

QSGName

Description:	Name of the queue-sharing group to which the queue manager where the activity was performed belongs.
Identifier:	MQCA_QSG_NAME.
Datatype:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_QSG_NAME_LENGTH
Returned:	If the activity was performed on a WebSphere MQ for z/OS queue manager.

TraceRoute

Description:	Grouped parameters specifying attributes of the trace-route message.
Identifier:	MQGACF_TRACE_ROUTE.
Datatype:	MQCFGR.
Contained in PCF group:	*Activity*.
Parameters in group:	*Detail* *RecordedActivities* *UnrecordedActivities* *DiscontinuityCount* *MaxActivities* *Accumulate* *Forward* *Deliver* For details of these parameter structures, see "Trace-route message data" on page 238.
Returned:	If the activity was performed on behalf of the trace-route message.

The values of the parameters in the *TraceRoute* PCF group are those from the trace-route message at the time the activity report was generated. For details of the *TraceRoute* PCF group, see "The TraceRoute PCF group" on page 176.

Operation-specific activity report message data

Additional PCF parameters are returned in the PCF group *Operation*, depending on the value of the *OperationType* parameter. For more information on the *OperationType* parameter, see OperationType, and for more information on the PCF group *Operation*, see Operation.

The additional parameters vary depending on the following operation types:
- "Get/Browse (MQOPER_GET/MQOPER_BROWSE)."
- "Discard (MQOPER_DISCARD)."
- "Publish/Discarded Publish/Excluded Publish (MQOPER_PUBLISH/MQOPER_DISCARDED_PUBLISH/MQOPER_EXCLUDED_PUBLISH)" on page 226
- "Put/Put Reply/Put Report (MQOPER_PUT/MQOPER_PUT_REPLY/MQOPER_PUT_REPORT)" on page 227.
- "Receive (MQOPER_RECEIVE)" on page 229.
- "Send (MQOPER_SEND)" on page 229.

Get/Browse (MQOPER_GET/MQOPER_BROWSE):

The additional activity report message data parameters that are returned in the PCF group *Operation* for the Get/Browse (MQOPER_GET/MQOPER_BROWSE) operation type (a message on a queue was got, or browsed).

QName

Description:	The name of the queue that was opened.
Identifier:	MQCA_Q_NAME.
Datatype:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_Q_NAME_LENGTH
Returned:	Always.

ResolvedQName

Description:	The name that the opened queue resolves to.
Identifier:	MQCACF_RESOLVED_Q_NAME.
Datatype:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_Q_NAME_LENGTH
Returned:	Always.

Discard (MQOPER_DISCARD):

The additional activity report message data parameters that are returned in the PCF group *Operation* for the Discard (MQOPER_DISCARD) operation type (a message was discarded).

Feedback

Description:	The reason for the message being discarded.
Identifier:	MQIACF_FEEDBACK.

Datatype:	MQCFIN.
Included in PCF group:	*Operation*.
Returned:	Always.

QName

Description:	The name of the queue that was opened.
Identifier:	MQCA_Q_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_NAME_LENGTH
Included in PCF group:	*Operation*.
Returned:	If the message was discarded because it was unsuccessfully put to a queue.

RemoteQMgrName

Description:	The name of the queue manager to which the message was destined.
Identifier:	MQCA_REMOTE_Q_MGR_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH
Included in PCF group:	*Operation*.
Returned:	If the value of *Feedback* is MQFB_NOT_FORWARDED.

Publish/Discarded Publish/Excluded Publish (MQOPER_PUBLISH/ MQOPER_DISCARDED_PUBLISH/MQOPER_EXCLUDED_PUBLISH):

The additional activity report message data parameters that are returned in the PCF group *Operation* for the Publish/Discarded Publish/Excluded Publish (MQOPER_PUBLISH/MQOPER_DISCARDED_PUBLISH/ MQOPER_EXCLUDED_PUBLISH) operation type (a publish/subscribe message was delivered, discarded, or excluded).

SubId

Description:	The subscription identifier.
Identifier:	MQBACF_SUB_ID.
Datatype:	MQCFBS.
Included in PCF group:	*Operation*.
Returned:	Always.

SubLevel

Description:	The subscription level.
Identifier:	MQIACF_SUB_LEVEL.
Datatype:	MQCFIN.
Included in PCF group:	*Operation*.
Returned:	Always.

Feedback

Description:	The reason for discarding the message.

Identifier:	MQIACF_FEEDBACK.
Datatype:	MQCFIN.
Included in PCF group:	*Operation*.
Returned:	If the message was discarded because it was not delivered to a subscriber, or the message was not delivered because the subscriber was excluded.

The Publish operation MQOPER_PUBLISH provides information about a message delivered to a particular subscriber. This operation describes the elements of the onward message that might have changed from the message described in the associated Put operation. Similarly to a Put operation, it contains a message group MQGACF_MESSAGE and, inside that, an MQMD group MQGACF_MQMD. However, this MQMD group contains only the following fields, which can be overridden by a subscriber: *Format, Priority, Persistence, MsgId, CorrelId, UserIdentifier, AccountingToken, ApplIdentityData.*

The *SubId* and *SubLevel* of the subscriber are included in the operation information. You can use the *SubID* with the MQCMD_INQUIRE_SUBSCRIBER PCF command to retrieve all other attributes for a subscriber.

The Discarded Publish operation MQOPER_DISCARDED_PUBLISH is analogous to the Discard operation that is used when a message is not delivered in point-to-point messaging. A message is not delivered to a subscriber if the message was explicitly requested not to be delivered to a local destination and this subscriber specifies a local destination. A message is also considered not delivered if there is a problem getting the message to the destination queue, for example, because the queue is full.

The information in a Discarded Publish operation is the same as for a Publish operation, with the addition of a *Feedback* field that gives the reasons why the message was not delivered. This feedback field contains MQFB_* or MQRC_* values that are common with the MQOPER_DISCARD operation. The reason for discarding a publish, as opposed to excluding it, are the same as the reasons for discarding a put.

The Excluded Publish operation MQOPER_EXCLUDED_PUBLISH provides information about a subscriber that was considered for delivery of the message, because the topic on which the subscriber is subscribing matches that of the associated Put operation, but the message was not delivered to the subscriber because other selection criteria do not match with the message that is being put to the topic. As with a Discarded Publish operation, the *Feedback* field provides information about the reason why this subscription was excluded. However, unlike the Discarded Publish operation, no message-related information is provided because no message was generated for this subscriber.

Put/Put Reply/Put Report (MQOPER_PUT/MQOPER_PUT_REPLY/ MQOPER_PUT_REPORT):

The additional activity report message data parameters that are returned in the PCF group *Operation* for the Put/Put Reply/Put Report (MQOPER_PUT/ MQOPER_PUT_REPLY/MQOPER_PUT_REPORT) operation type (a message, reply message, or report message was put to a queue).

QName

Description:	The name of the queue that was opened.
Identifier:	MQCA_Q_NAME.
Datatype:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_Q_NAME_LENGTH
Returned:	Always, apart from one exception: not returned if the Put operation is to a topic, contained within a publish activity.

ResolvedQName

Description:	The name that the opened queue resolves to.
Identifier:	MQCACF_RESOLVED_Q_NAME.
Datatype:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_Q_NAME_LENGTH
Returned:	When the opened queue could be resolved. Not returned if the Put operation is to a topic, contained within a publish activity.

RemoteQName

Description:	The name of the opened queue, as it is known on the remote queue manager.
Identifier:	MQCA_REMOTE_Q_NAME.
Datatype:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_Q_NAME_LENGTH
Returned:	If the opened queue is a remote queue. Not returned if the Put operation is to a topic, contained within a publish activity.

RemoteQMgrName

Description:	The name of the remote queue manager on which the remote queue is defined.
Identifier:	MQCA_REMOTE_Q_MGR_NAME.
Datatype:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_Q_MGR_NAME_LENGTH
Returned:	If the opened queue is a remote queue. Not returned if the Put operation is to a topic, contained within a publish activity.

TopicString

Description:	The full topic string to which the message is being put.
Identifier:	MQCA_TOPIC_STRING.
Datatype:	MQCFST.
Included in PCF group:	*Operation*.
Returned:	If the Put operation is to a topic, contained within a publish activity.

Feedback

Description:	The reason for the message being put on the dead-letter queue.
Identifier:	MQIACF_FEEDBACK.
Datatype:	MQCFIN.
Included in PCF group:	*Operation*.
Returned:	If the message was put on the dead-letter queue.

Receive (MQOPER_RECEIVE):

The additional activity report message data parameters that are returned in the PCF group *Operation* for the Receive (MQOPER_RECEIVE) operation type (a message was received on a channel).

ChannelName

Description:	The name of the channel on which the message was received.
Identifier:	MQCACH_CHANNEL_NAME.
Datatype:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_CHANNEL_NAME_LENGTH
Returned:	Always.

ChannelType

Description:	The type of channel on which the message was received.
Identifier:	MQIACH_CHANNEL_TYPE.
Datatype:	MQCFIN.
Included in PCF group:	*Operation*.
Returned:	Always.

RemoteQMgrName

Description:	The name of the queue manager from which the message was received.
Identifier:	MQCA_REMOTE_Q_MGR_NAME.
Datatype:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_Q_MGR_NAME_LENGTH
Returned:	Always.

Send (MQOPER_SEND):

The additional activity report message data parameters that are returned in the PCF group *Operation* for the Send (MQOPER_SEND) operation type (a message was sent on a channel).

ChannelName

Description:	The name of the channel where the message was sent.
Identifier:	MQCACH_CHANNEL_NAME.
Datatype:	MQCFST.

Included in PCF group:	*Operation*.
Maximum length:	MQ_CHANNEL_NAME_LENGTH.
Returned:	Always.

ChannelType

Description:	The type of channel where the message was sent.
Identifier:	MQIACH_CHANNEL_TYPE.
Datatype:	MQCFIN.
Included in PCF group:	*Operation*.
Returned:	Always.

XmitQName

Description:	The transmission queue from which the message was retrieved.
Identifier:	MQCACH_XMIT_Q_NAME.
Datatype:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_Q_NAME_LENGTH.
Returned:	Always.

RemoteQMgrName

Description:	The name of the remote queue manager to which the message was sent.
Identifier:	MQCA_REMOTE_Q_MGR_NAME.
Datatype:	MQCFST.
Included in PCF group:	*Operation*.
Maximum length:	MQ_Q_MGR_NAME_LENGTH
Returned:	Always.

Trace-route message reference

This chapter provides an overview of the trace-route message format. It describes the information returned in trace-route messages, including returned parameters.

Trace-route message format

Trace-route messages are standard WebSphere MQ messages containing a message descriptor and message data. The message data contains information about the activities performed on a trace-route message as it has been routed through a queue manager network. For information on trace-route messaging, see "Trace-route messaging" on page 169.

Trace-route messages contain the following:

A message descriptor
An MQMD structure, with the *Format* field set to MQFMT_ADMIN or MQFMT_EMBEDDED_PCF.

Message data
Consists of either:

- A PCF header (MQCFH) and trace-route message data, if *Format* is set to MQFMT_ADMIN, or
- An embedded PCF header (MQEPH), trace-route message data, and additional user-specified message data, if *Format* is set to MQFMT_EMBEDDED_PCF.

When using the WebSphere MQ display route application to generate a trace-route message, *Format* is set to MQFMT_ADMIN.

Trace-route message data format

The content of the trace-route message data is determined by the *Accumulate* parameter from the *TraceRoute* PCF group, as follows:

- If *Accumulate* is set to MQROUTE_ACCUMULATE_NONE, then the trace-route message data contains the following:
  - The *TraceRoute* PCF group.
- If *Accumulate* is set to either MQROUTE_ACCUMULATE_IN_MSG or MQROUTE_ACCUMULATE_AND_REPLY, then the trace-route message data contains the following:
  - The *TraceRoute* PCF group.
  - Zero or more *Activity* PCF groups.

Example of a trace-route message

Table 18 shows the structure of a trace-route message.

Table 18. Trace-route message format

Message descriptor	Message data	
MQMD structure	Embedded PCF header MQEPH structure	Trace-route message data

Table 18. Trace-route message format (continued)

Structure identifier
Structure version
Report options
Message type
Expiration time
Feedback
Encoding
Coded character set ID
Message format
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 identifier
	Structure version
	Structure length
	Encoding
	Coded character set ID
	Message format
	Flags
	PCF header (MQCFH)
	  Structure type
	  Structure length
	  Structure version
	  Command identifier
	  Message sequence number
	  Control options
	  Completion code
	  Reason code
	  Parameter count
		TraceRoute
		  Detail
		  Recorded activities
		  Unrecorded activities
		  Discontinuity count
		  Max activities
		  Accumulate
		  Deliver

Trace-route message MQMD (message descriptor)

The MQMD structure describes the information that accompanies the message data of a trace-route 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 a trace-route 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.**

Report

Description:	Options for report messages.

Datatype:	MQLONG.
Value:	Set according to requirements. Common report options follow:

MQRO_DISCARD_MSG
The message is discarded on arrival to a local queue.

MQRO_PASS_DISCARD_AND_EXPIRY
Every response (activity reports or trace-route reply message) will have the report option MQRO_DISCARD_MSG set, and the remaining expiry passed on. This ensures that responses do not remain in the queue manager network indefinitely.

MsgType

Description:	Type of message.
Datatype:	MQLONG.
Value:	If the *Accumulate* parameter in the TraceRoute group is specified as MQROUTE_ACCUMULATE_AND_REPLY, then message type is MQMT_REQUEST

Otherwise:

MQMT_DATAGRAM.

Expiry

Description:	Message lifetime.
Datatype:	MQLONG.
Value:	Set according to requirements. This parameter can be used to ensure trace-route messages are not left in a queue manager network indefinitely.

Feedback

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

Encoding

Description:	Numeric encoding of message data.
Datatype:	MQLONG.
Value:	Set as appropriate.

CodedCharSetId

Description:	Character set identifier of message data.
Datatype:	MQLONG.
Value:	Set as appropriate.

Format

Description:	Format name of message data
Datatype:	MQCHAR8.

Value:

MQFMT_ADMIN
Admin message. No user data follows the *TraceRoute* PCF group.

MQFMT_EMBEDDED_PCF
Embedded PCF message. User data follows the *TraceRoute* PCF group.

Priority

Description:	Message priority.
Datatype:	MQLONG.
Value:	Set according to requirements.

Persistence

Description:	Message persistence.
Datatype:	MQLONG.
Value:	Set according to requirements.

MsgId

Description:	Message identifier.
Datatype:	MQBYTE24.
Value:	Set according to requirements.

CorrelId

Description:	Correlation identifier.
Datatype:	MQBYTE24.
Value:	Set according to requirements.

BackoutCount

Description:	Backout counter.
Datatype:	MQLONG.
Value:	0.

ReplyToQ

Description:	Name of reply queue.
Datatype:	MQCHAR48.
Values:	Set according to requirements.

If *MsgType* is set to MQMT_REQUEST or if *Report* has any report generating options set, then this parameter must be non-blank.

ReplyToQMgr

Description:	Name of reply queue manager.
Datatype:	MQCHAR48.
Value:	Set according to requirements.

UserIdentifier

Description:	The user identifier of the application that originated the message.

Datatype:	MQCHAR12.
Value:	Set as normal.

AccountingToken

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

ApplIdentityData

Description:	Application data relating to identity.
Datatype:	MQCHAR32.
Values:	Set as normal.

PutApplType

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

PutApplName

Description:	Name of application that put the message.
Datatype:	MQCHAR28.
Value:	Set as normal.

PutDate

Description:	Date when message was put.
Datatype:	MQCHAR8.
Value:	Set as normal.

PutTime

Description:	Time when message was put.
Datatype:	MQCHAR8.
Value:	Set as normal.

ApplOriginData

Description:	Application data relating to origin.
Datatype:	MQCHAR4.
Value:	Set as normal..

Trace-route message MQEPH (Embedded PCF header)

The MQEPH structure contains a description of both the PCF information that accompanies the message data of a trace-route message and the application message data which follows it. For a full description of the MQEPH structure, including a description of the elementary datatype of each parameter, see "MQEPH - Embedded PCF header" on page 330.

An MQEPH structure is used only if additional user message data follows the TraceRoute PCF group.

For a trace-route message, the MQEPH structure contains the following values:

StrucId

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

Version

Description:	Structure version number.
Datatype:	MQLONG.
Values:	MQEPH_VERSION_1.

StrucLength

Description:	Structure length.
Datatype:	MQLONG.
Value:	Total length of the structure including the PCF parameter structures that follow it.

Encoding

Description:	Numeric encoding of the message data that follows the last PCF parameter structure.
Datatype:	MQLONG.
Value:	The encoding of the message data.

CodedCharSetId

Description:	Character set identifier of the message data that follows the last PCF parameter structure.
Datatype:	MQLONG.
Value:	The character set of the message data.

Format

Description:	Format name of the message data that follows the last PCF parameter structure.
Datatype:	MQCHAR8.
Value:	The format name of the message data.

Flags

Description: Flags that specify attributes of the structure or control its processing.

Datatype: MQLONG.

Value:

MQEPH_NONE
: No flags specified.

MQEPH_CCSID_EMBEDDED
: Specifies that the character set of the parameters containing character data is specified individually within the *CodedCharSetId* field in each structure.

PCFHeader

Description:	Programmable Command Format Header
Datatype:	MQCFH.
Value:	See "Trace-route message MQCFH (PCF header)."

Trace-route message MQCFH (PCF header)

The MQCFH structure describes the PCF information available in the trace-route message. For a full description of MQCFH, including a description of the elementary datatype of each parameter, see "MQCFH - PCF header" on page 314.

For a trace-route message, the MQCFH structure contains the following values:

Type

Description:	Structure type that identifies the content of the message.
Datatype:	MQLONG.
Value:	**MQCFT_TRACE_ROUTE** Message is a trace-route message.

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_3

Command

Description:	Command identifier. This identifies the category of the message.
Datatype:	MQLONG.
Values:	**MQCMD_TRACE_ROUTE** Trace-route message.

MsgSeqNumber

Description:	Message sequence number. This is the sequence number of the message within a group of related messages.
Datatype:	MQLONG.
Values:	1.

Control

Description:	Control options.
Datatype:	MQLONG.
Values:	MQCFC_LAST.

CompCode

Description:	Completion code.
Datatype:	MQLONG.
Values:	MQCC_OK.

Reason

Description:	Reason code qualifying completion code.
Datatype:	MQLONG.
Values:	MQRC_NONE.

ParameterCount

Description:	Count of parameter structures. This is the number of parameter structures that follow the MQCFH structure. A group structure (MQCFGR), and its included parameter structures, are counted as one structure only.
Datatype:	MQLONG.
Values:	1 or greater.

Trace-route message data

The content of trace-route message data depends on the *Accumulate* parameter from the *TraceRoute* PCF group, see "Trace-route message data format" on page 231. Trace-route message data consists of the *TraceRoute* PCF group, and zero or more *Activity* PCF groups. The *TraceRoute* PCF group is detailed below. For details of the *Activity* PCF group, see "Activity report message data" on page 214 and "Operation-specific activity report message data" on page 225.

Trace-route message data contains the following parameters:

TraceRoute

Description:	Grouped parameters specifying attributes of the trace-route message. For a trace-route message, some of these parameters can be altered to control how it is processed.
Identifier:	MQGACF_TRACE_ROUTE.
Datatype:	MQCFGR.
Contained in PCF group:	None.
Parameters in group:	*Detail* *RecordedActivities* *UnrecordedActivities* *DiscontinuityCount* *MaxActivities* *Accumulate* *Forward* *Deliver*

Detail

Description:	The detail level that will be recorded for the activity.
Identifier:	MQIACF_ROUTE_DETAIL.

Datatype:	MQCFIN.
Contained in PCF group:	*TraceRoute*.
Values:	**MQROUTE_DETAIL_LOW** Activities performed by user-written application are recorded. **MQROUTE_DETAIL_MEDIUM** Activities specified in MQROUTE_DETAIL_LOW are recorded. Additionally, activities performed by MCAs are recorded. **MQROUTE_DETAIL_HIGH** Activities specified in MQROUTE_DETAIL_LOW, and MQROUTE_DETAIL_MEDIUM are recorded. MCAs do not record any further activity information at this level of detail. This option is only available to user-written applications that are to record further activity information.

RecordedActivities

Description:	The number of activities that the trace-route message has caused, where information was recorded.
Identifier:	MQIACF_RECORDED_ACTIVITIES.
Datatype:	MQCFIN.
Contained in PCF group:	*TraceRoute*.

UnrecordedActivities

Description:	The number of activities that the trace-route message has caused, where information was not recorded.
Identifier:	MQIACF_UNRECORDED_ACTIVITIES.
Datatype:	MQCFIN.
Contained in PCF group:	*TraceRoute*.

DiscontinuityCount

Description:	The number of times a trace-route message has been received from a queue manager that does not support trace-route messaging.
Identifier:	MQIACF_DISCONTINUITY_COUNT.
Datatype:	MQCFIN.
Contained in PCF group:	*TraceRoute*.

MaxActivities

Description:	The maximum number of activities the trace-route message can be involved in before it stops being processed.
Identifier:	MQIACF_MAX_ACTIVITIES.
Datatype:	MQCFIN.
Contained in PCF group:	*TraceRoute*.
Value:	**A positive integer** The maximum number of activities. **MQROUTE_UNLIMITED_ACTIVITIES** An unlimited number of activities.

Accumulate

Description:	Specifies whether activity information is accumulated within the trace-route message, and whether a reply message containing the accumulated activity information is generated before the trace-route message is discarded or is put on a non-transmission queue.
Identifier:	MQIACF_ROUTE_ACCUMULATION.
Datatype:	MQCFIN.
Contained in PCF group:	*TraceRoute*.
Value:	

MQROUTE_ACCUMULATE_NONE
: Activity information is not accumulated in the message data of the trace-route message.

MQROUTE_ACCUMULATE_IN_MSG
: Activity information is accumulated in the message data of the trace-route message.

MQROUTE_ACCUMULATE_AND_REPLY
: Activity information is accumulated in the message data of the trace-route message, and a trace-route reply message will be generated.

Forward

Description:	Specifies queue managers that the trace-route message can be forwarded to. Queue managers use an algorithm when determining whether to forward a message to a remote queue manager. For details of this algorithm, see Forwarding.
Identifier:	MQIACF_ROUTE_FORWARDING.
Datatype:	MQCFIN.
Contained in PCF group:	*TraceRoute*.
Value:	

MQROUTE_FORWARD_IF_SUPPORTED
: The trace-route message is only forwarded to queue managers that will honor the value of the *Deliver* parameter from the *TraceRoute* group.

MQROUTE_FORWARD_ALL
: The trace-route message is forwarded to any queue manager, regardless of whether the value of the *Deliver* parameter will be honored.

Deliver

Description:	Specifies the action to be taken if the trace-route message arrives at the destination queue successfully.
Identifier:	MQIACF_ROUTE_DELIVERY.
Datatype:	MQCFIN.
Contained in PCF group:	*TraceRoute*.
Value:	

MQROUTE_DELIVER_YES
: On arrival, the trace-route message is put on the target queue. Any application performing a destructive get on the target queue can receive the trace-route message.

MQROUTE_DELIVER_NO
: On arrival, the trace-route message is discarded.

For details of the *Activity* PCF group, see “Activity report message data” on page 214 and “Operation-specific activity report message data” on page 225.

Trace-route reply message reference

This chapter provides an overview of the trace-route reply message format. It describes the information returned in trace-route reply messages, including returned parameters.

Trace-route reply message format

Trace-route reply messages are standard WebSphere MQ messages containing a message descriptor and message data. The message data contains information about the activities performed on a trace-route message as it has been routed through a queue manager network. For information on trace-route messaging, see “Trace-route messaging” on page 169.

Trace-route reply messages contain the following:

A message descriptor
  An MQMD structure

Message data
  A PCF header (MQCFH) and trace-route reply message data

Trace-route reply message data consists of one or more *Activity* PCF groups.

When a trace-route message reaches its target queue, a trace-route reply message can be generated that contains a copy of the activity information from the trace-route message. The trace-route reply message will be delivered to a reply-to queue or to a system queue, see “Controlling queue managers for trace-route messaging” on page 172.

Table 19 shows the structure of a trace-route reply message, including parameters that are only returned under certain conditions.

Table 19. Trace-route reply message format

Message descriptor	Message data	
MQMD structure	PCF header MQCFH structure	Trace-route reply message data

Table 19. Trace-route reply message format (continued)

Structure identifier	PCF header (MQCFH)	Activity
Structure version	Structure type	Activity application name
Report options	Structure length	Activity application type
Message type	Structure version	Activity description
Expiration time	Command identifier	Operation
Feedback	Message sequence number	Operation type
Encoding	Control options	Operation date
Coded character set ID	Completion code	Operation time
Message format	Reason code	Message
Priority	Parameter count	Message length
Persistence		MQMD
Message identifier		EmbeddedMQMD
Correlation identifier		Queue manager name
Backout count		Queue sharing group name
Reply-to queue		Queue name [1] [2] [3]
Reply-to queue manager		Resolved queue name [1] [3]
User identifier		Remote queue name [3]
Accounting token		Remote queue manager-name [2] [3] [4] [5]
Application identity data		Feedback [2]
Application type		Channel name [4] [5]
Application name		Channel type [4] [5]
Put date		Transmission queue name [5]
Put time		TraceRoute
Application origin data		Detail
Group identifier		Recorded activities
Message sequence number		Unrecorded activities
Offset		Discontinuity count
Message flags		Max activities
Original length		Accumulate
		Deliver

Note:

1. Returned for Get and Browse operations.
2. Returned for Discard operations.
3. Returned for Put, Put Reply, and Put Report operations.
4. Returned for Receive operations.
5. Returned for Send operations.

Trace-route reply message MQMD (message descriptor)

The MQMD structure describes the information that accompanies the message data of a trace-route reply 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 a trace-route reply message, the MQMD structure contains the parameters as described in Activity report message descriptor. Some of the parameter values in a trace-route reply message descriptor are different from those in an activity report message descriptor, as follows:

MsgType

Description: Type of message.

Datatype:	MQLONG.
Value:	**MQMT_REPLY**

Feedback

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

Encoding

Description:	Numeric encoding of message data.
Datatype:	MQLONG.
Value:	Copied from trace-route message descriptor.

CodedCharSetId

Description:	Character set identifier of message data.
Datatype:	MQLONG.
Value:	Copied from trace-route message descriptor.

Format

Description:	Format name of message data
Datatype:	MQCHAR8.
Value:	**MQFMT_ADMIN** Admin message.

Trace-route reply message MQCFH (PCF header)

The MQCFH structure describes the PCF information available in the trace-route reply message. For a full description of MQCFH, including a description of the elementary datatype of each parameter, see "MQCFH - PCF header" on page 314.

The PCF header (MQCFH) for a trace-route reply message is the same as for a trace-route message, see "Trace-route message MQCFH (PCF header)" on page 237.

Trace-route reply message data

The trace-route reply message data of a trace-route reply message is a duplicate of the trace-route message data from the trace-route message for which it was generated. The trace-route reply message data contains one or more *Activity* groups.

The trace-route reply message data contains the parameters as described in "Activity report message data" on page 214.

Chapter 4. Accounting and statistics messages

Accounting and statistics messages

Accounting and statistics messages are generated intermittently by queue managers to record information about MQI operations performed by WebSphere MQ applications or to record information about the activities occurring in a WebSphere MQ system.

An introduction to accounting and statistics messages

Accounting and *statistics* messages are generated intermittently by queue managers to record information about the MQI operations performed by WebSphere MQ applications, or to record information about the activities occurring in a WebSphere MQ system.

Accounting messages
Accounting messages are used to record information about the MQI operations performed by WebSphere MQ applications, see “Accounting messages.”

Statistics messages
Statistics messages are used to record information about the activities occurring in a WebSphere MQ system, see “Statistics messages” on page 249.

Accounting messages and statistics messages as described here are not available on WebSphere MQ for z/OS, however equivalent functionality is available through the System Management Facility (SMF), see the WebSphere MQ for z/OS System Setup Guide.

Accounting and statistics messages are delivered to two system queues. Users or user applications can retrieve messages from these system queues and use the recorded information for various tasks such as resource charging or system monitoring.

Accounting messages

Accounting messages are used to record information about the MQI operations performed by WebSphere MQ applications. An accounting message is a PCF message that contains a number of PCF structures.

When an application disconnects from a queue manager, an accounting message is generated and delivered to the system accounting queue (SYSTEM.ADMIN.ACCOUNTING.QUEUE). For long running WebSphere MQ applications, intermediate accounting messages are generated as follows:

- When the time since the connection was established exceeds the configured interval.
- When the time since the last intermediate accounting message exceeds the configured interval.

The information contained within accounting messages can be used for the following:

- Account for application resource use.
- Record application activity.
- Detect problems in your queue manager network.
- Assist in determining the causes of problems in your queue manager network.
- Improve the efficiency of your queue manager network.
- Familiarize yourself with the running of your queue manager network.
- Confirm your queue manager network is running correctly.

Accounting message types

The are two categories of accounting message, as follows:

MQI accounting messages
MQI accounting messages contain information relating to the number of MQI calls made using a connection to a queue manager.

Queue accounting messages
Queue accounting messages contain information relating to the number of MQI calls made using connections to a queue manager, grouped by queue.

Each queue accounting message can contain up to 100 records, with every record relating to an activity performed by the application with respect to a specific queue.

Accounting messages are recorded only for local queues. If an application makes an MQI call against an alias queue, the accounting data is recorded against the base queue, and, for a remote queue, the accounting data is recorded against the transmission queue.

Controlling accounting messages

The collection of accounting information is controlled by a set of queue manager, and queue attributes.

Collecting MQI accounting information:

The collection of MQI accounting information is controlled by the queue manager attribute, ACCTMQI. To change the value of this attribute you can use the MQSC command, `ALTER QMGR`, and specify the parameter `ACCTMQI`. Accounting messages are generated only for connections which begin after accounting is enabled.

The ACCTMQI parameter can have the following values:

ON MQI accounting information is collected for every connection to the queue manager.

OFF MQI accounting information is not collected. This is the default value.

For example, to enable MQI accounting information collection use the following MQSC command:

```
ALTER QMGR ACCTMQI(ON)
```

Collecting queue accounting information:

The collection of queue accounting information is controlled by the queue attribute, ACCTQ, and the queue manager attribute, ACCTQ. To change the value of the queue attribute, you can use the MQSC command, `ALTER QLOCAL` and specify the parameter `ACCTQ`. Accounting messages are generated only for connections which begin after accounting is enabled.

The queue attribute, ACCTQ, can have the following values:

ON
: Queue accounting information for this queue is collected for every connection to the queue manager that opens the queue.

OFF
: Queue accounting information for this queue is not collected.

QMGR
: The collection of queue accounting information for this queue is controlled according to the value of the queue manager attribute, ACCTQ. This is the default value.

The queue manager attribute, ACCTQ, can have the following values:

ON
: Queue accounting information is collected for queues that have the queue attribute ACCTQ set as QMGR.

OFF
: Queue accounting information is not collected for queues that have the queue attribute ACCTQ set as QMGR. This is the default value.

NONE
: The collection of queue accounting information is disabled for all queues, regardless of the queue attribute ACCTQ.

To enable accounting information collection for the queue, `Q1`, use the following MQSC command:

```
ALTER QLOCAL(Q1) ACCTQ(ON)
```

To enable accounting information collection for all queues that specify the queue attribute ACCTQ as QMGR, use the following MQSC command:

```
ALTER QMGR ACCTQ(ON)
```

If the queue manager attribute, ACCTQ, is set to NONE, the collection of queue accounting information is disabled for all queues, regardless of the queue attribute ACCTQ.

Controlling accounting information collection using MQCONNX:

The collection of both MQI and queue accounting information can also be modified at the connection level by specifying the ConnectOpts parameter on the MQCONNX call. By altering the value of ConnectOpts, it is possible to override the effective value of the queue manager attributes ACCTMQI and ACCTQ.

ConnectOpts can have the following values:

MQCNO_ACCOUNTING_MQI_ENABLED
: If the value of the queue manager attribute ACCTMQI is specified as OFF, then MQI accounting is enabled for this connection. This is equivalent of the queue manager attribute ACCTMQI being specified as ON.

  If the value of the queue manager attribute ACCTMQI is not specified as OFF, then this attribute has no effect.

MQCNO_ACCOUNTING_MQI_DISABLED
: If the value of the queue manager attribute ACCTMQI is specified as ON,

then MQI accounting is disabled for this connection. This is equivalent of the queue manager attribute ACCTMQI being specified as OFF.

If the value of the queue manager attribute ACCTMQI is not specified as ON, then this attribute has no effect.

MQCNO_ACCOUNTING_Q_ENABLED
If the value of the queue manager attribute ACCTQ is specified as OFF, then queue accounting is enabled for this connection. All queues with ACCTQ specified as QMGR, are enabled for queue accounting. This is equivalent of the queue manager attribute ACCTQ being specified as ON.

If the value of the queue manager attribute ACCTQ is not specified as OFF, then this attribute has no effect.

MQCNO_ACCOUNTING_Q_DISABLED
If the value of the queue manager attribute ACCTQ is specified as ON, queue accounting is disabled for this connection. This is equivalent of the queue manager attribute ACCTQ being specified as OFF.

If the value of the queue manager attribute ACCTQ is not specified as ON, then this attribute has no effect.

These overrides are by disabled by default. To enable them, set the queue manager attribute ACCTCONO to ENABLED. To enable accounting overrides per connection use the following MQSC command:

```
ALTER QMGR ACCTCONO(ENABLED)
```

Generating accounting messages:

Accounting messages are generated upon the disconnection of the application from the queue manager, either upon the execution of the MQDISC operation, or implicity by the queue manager upon the recognition of the termination of the application.

Intermediate accounting messages are also written for long running WebSphere MQ applications when the interval since the connection was established or since the last intermediate accounting message that was written exceeds the configured interval. The queue manager attribute, ACCTINT, specifies the time, in seconds, after which intermediate accounting messages can be automatically written. Accounting messages are only generated when the application interacts with the queue manager, so applications that remain connected to the queue manager for long periods without executing MQI requests will not generate accounting messages until the execution of the first MQI request following the completion of the accounting interval.

The default accounting interval is 1800 seconds (30 minutes). For example, to change the accounting interval to 900 seconds (15 minutes) use the following MQSC command:

```
ALTER QMGR ACCTINT(900)
```

Format of accounting messages

Accounting messages are constructed as a set of PCF fields that consist of the following:

A message descriptor
An accounting message MQMD (message descriptor)

Accounting message data

- An accounting message MQCFH (PCF header)
- Accounting message data that is always returned
- Accounting message data that is returned if available

The accounting message MQCFH (PCF header) contains information about the application, and the interval for which the accounting data was recorded.

Accounting message data is comprised of PCF parameters that store the accounting information. The content of accounting messages depends on the message category as follows:

MQI accounting message

MQI accounting message data consists of a number of PCF parameters, but no PCF groups.

The parameters contained in MQI accounting message data are described in "MQI accounting message data" on page 264.

Queue accounting message

Queue accounting message data consists of a number of PCF parameters, and between one and one hundred *QAccountingData* PCF groups.

There is one *QAccountingData* PCF group for every queue that had accounting data collected. If an application accesses more than 100 queues, multiple accounting messages are generated. Each message has the *SeqNumber* in the MQCFH (PCF header) updated accordingly, and the last message in the sequence has the *Control* parameter in the MQCFH specified as `MQCFC_LAST`.

The parameters contained in queue accounting message data are described in "Queue accounting message data" on page 270.

Statistics messages

Statistics messages are used to record information about the activities occurring in a WebSphere MQ system. An statistics messages is a PCF message that contains a number of PCF structures. Statistics messages are delivered to the system queue (SYSTEM.ADMIN.STATISTICS.QUEUE) at configured intervals.

The information contained within statistics messages can be used for the following:

- Account for application resource use.
- Record application activity.
- Capacity planning.
- Detect problems in your queue manager network.
- Assist in determining the causes of problems in your queue manager network.
- Improve the efficiency of your queue manager network.
- Familiarize yourself with the running of your queue manager network.
- Confirm your queue manager network is running correctly.

Statistics message types

The various types of statistics message follow:

MQI statistics messages

MQI statistics messages contain information relating to the number of MQI calls made during a configured interval. For example, the information can include the number of MQI calls actioned by a queue manager.

Queue statistics messages
Queue statistics messages contain information relating to the activity of a queue during a configured interval. The information includes the number of messages put on, and retrieved from, the queue, and the total number of bytes processed by a queue.

Each queue statistics message can contain up to 100 records, with each record relating to the activity per queue for which statistics were collected.

Statistics messages are recorded only for local queues. If an application makes an MQI call against an alias queue, the statistics data is recorded against the base queue, and, for a remote queue, the statistics data is recorded against the transmission queue.

Channel statistics messages
Channel statistics messages contain information relating to the activity of a channel during a configured interval. For example the information might be the number of messages transferred by the channel, or the number of bytes transferred by the channel.

Each channel statistics message contains up to 100 records, with each record relating to the activity per channel for which statistics were collected.

Controlling statistics messaging

The collection of statistics data is controlled by queue manager, queue, and channel attributes.

Collecting MQI statistics information:

The collection of MQI statistics information is controlled by the queue manager attribute, STATMQI. To change the value of this r attribute, you can use the MQSC command, `ALTER QMGR` and specify the parameter `STATMQI`. Statistics messages are generated only for queues which are opened after statistics collection has been enabled.

`STATMQI` can have the following values:

ON MQI statistics information is collected for every connection to the queue manager.

OFF MQI statistics information is not collected. This is the default value.

For example, to enable MQI statistics use the following MQSC command:

```
ALTER QMGR STATMQI(ON)
```

Collecting queue statistics information:

Queue statistics information collection can be enabled or disabled for individual queues, or for multiple queues. To control individual queues, set the queue attribute STATQ. Queue statistics information collection can be enabled or disabled at the queue manager level using the queue manager attribute STATQ. For all queues that have the queue attribute STATQ specified with the value QMGR queue statistics information collection is controlled at the queue manager level. Statistics messages are generated only for queues which are opened after statistics collection has been enabled.

To change the value of the queue attribute STATQ, you can use the MQSC command, `ALTER QLOCAL` and specify the parameter `STATQ`. To change the value of the queue manager attribute STATQ, you can use the MQSC command, `ALTER QMGR` and specify the parameter `STATQ`.

The queue attribute, STATQ, can have the following values:

ON
: Queue statistics information is collected for every connection to the queue manager that opens the queue.

OFF
: Queue statistics information for this queue is not collected.

QMGR
: The collection of queue statistics information for this queue is controlled according to the value of the queue manager attribute, STATQ. This is the default value.

The queue manager attribute, STATQ, can have the following values:

ON
: Queue statistics information is collected for queues that have the queue attribute STATQ set as QMGR

OFF
: Queue statistics information is not collected for queues that have the queue attribute STATQ set as QMGR. This is the default value.

NONE
: The collection of queue statistics information is disabled for all queues, regardless of the queue attribute STATQ.

To enable statistics information collection for the queue, `Q1`, use the following MQSC command:

```
ALTER QLOCAL(Q1) STATQ(ON)
```

To enable statistics information collection for all queues that specify the queue attribute STATQ as QMGR, use the following MQSC command:

```
ALTER QMGR STATQ(ON)
```

If the queue manager attribute, STATQ, is set to NONE, the collection of queue statistics information is disabled for all queues, regardless of the queue attribute STATQ.

Collecting channel statistics information:

Channel statistics information collection can be enabled or disabled for individual channels, or for multiple channels. To control individual channels, the channel attribute STATCHL must be set to enable or disable channel statistic information collection. To control many channels together, channel statistics information collection can be enabled or disabled at the queue manager level using the queue manager attribute STATCHL. For all channels that have the channel attribute STATCHL specified with the value, QMGR, channel statistics information collection is controlled at the queue manager level.

Automatically defined cluster-sender channels are not WebSphere MQ objects, so do not have attributes in the same way as channel objects. To control automatically defined cluster-sender channels, use the queue manager attribute, STATACLS. This attribute determines whether automatically defined cluster-sender channels within a queue manager are enabled or disabled for channel statistics information collection.

Channel statistics information collection can be set to one of the three monitoring levels, low, medium or high. This is set at either object level, or at the queue manager level. The choice of which level to use is dependant on your system. Collecting statistics information data may require the execution of some relatively expensive instructions, so in order to reduce the impact of channel statistics information collection, the medium and low monitoring options measure a sample of the data at regular intervals rather than collecting data all the time. Table 20 summarizes the levels available with channel statistics information collection:

Table 20. Detail level of channel statistics information collection

Level	Description	Usage
Low	Measure a small sample of the data, at regular intervals.	For objects that process a high volume of messages.
Medium	Measure a sample of the data, at regular intervals.	For most objects.
High	Measure all data, at regular intervals.	For objects that process only a few messages per second, on which the most current information is important.

To change the value of the channel attribute STATCHL, you can use the MQSC command, `ALTER CHANNEL` and specify the parameter `STATCHL`.

To change the value of the queue manager attribute STATCHL, you can use the MQSC command, `ALTER QMGR` and specify the parameter `STATCHL`.

To change the value of the queue manager attribute STATACLS, you can use the MQSC command, `ALTER QMGR` and specify the parameter `STATACLS`.

The channel attribute, STATCHL, can have the following values:

LOW Channel statistics information is collected with a low level of detail.

MEDIUM
Channel statistics information is collected with a medium level of detail.

HIGH Channel statistics information is collected with a high level of detail.

OFF Channel statistics information is not collected for this channel.

This is the default value.

QMGR
The channel attribute is set as QMGR. The collection of statistics information for this channel is controlled by the value of the queue manager attribute, STATCHL.

The queue manager attribute, STATCHL, can have the following values:

LOW Channel statistics information is collected with a low level of detail, for all channels that have the channel attribute STATCHL set as QMGR.

MEDIUM
Channel statistics information is collected with a medium level of detail, for all channels that have the channel attribute STATCHL set as QMGR.

HIGH Channel statistics information is collected with a high level of detail, for all channels that have the channel attribute STATCHL set as QMGR.

OFF Channel statistics information is not collected for all channels that have the channel attribute STATCHL set as QMGR.

This is the default value.

NONE
The collection of channel statistics information is disabled for all channel, regardless of the channel attribute STATCHL.

The queue manager attribute, STATACLS, can have the following values:

LOW Statistics information is collected with a low level of detail for automatically defined cluster-sender channels.

MEDIUM
Statistics information is collected with a medium level of detail for automatically defined cluster-sender channels.

HIGH Statistics information is collected with a high level of detail for automatically defined cluster-sender channels.

OFF Statistics information is not for automatically defined cluster-sender channels.

This is the default value.

QMGR
The collection of statistics information for automatically defined cluster-sender channels is controlled by the value of the queue manager attribute, STATCHL.

For example, to enable statistics information collection, with a medium level of detail, for the sender channel `QM1.TO.QM2`, use the following MQSC command:

```
ALTER CHANNEL(QM1.TO.QM2) CHLTYPE(SDR) STATCHL(MEDIUM)
```

To enable statistics information collection, at a medium level of detail, for all channels that specify the channel attribute STATCHL as QMGR, use the following MQSC command:

```
ALTER QMGR STATCHL(MEDIUM)
```

To enable statistics information collection, at a medium level of detail, for all automatically defined cluster-sender channels, use the following MQSC command:

```
ALTER QMGR STATACLS(MEDIUM)
```

Generating statistics messages:

Statistics messages are generated at configured intervals, and when a queue manager shuts down in a controlled fashion.

The configured interval is controlled by the STATINT queue manager attribute. STATINT specifies the interval, in seconds, between the generation of statistics messages. The default statistics interval is 1800 seconds (30 minutes). To change the statistics interval you can you the MQSC command `ALTER QMGR` and specify the `STATINT` parameter. For example, to change the statistics interval to 900 seconds (15 minutes) use the following MQSC command:

```
ALTER QMGR STATINT(900)
```

To write the currently collected statistics data to the statistics queue before the statistics collection interval is due to expire, you can use issue the MQSC command

RESET QMGR TYPE(STATISTICS). This causes the collected statistics data to be written to the statistics queue and a new statistics data collection interval to begin.

Format of statistics messages

Statistics messages are constructed as a set of PCF fields that consist of the following:

A message descriptor
A statistics message MQMD (message descriptor)

Accounting message data
- A statistics message MQCFH (PCF header)
- Statistics message data that is always returned
- Statistics message data that is returned if available

The statistics message MQCFH (PCF header) contains information about the interval for which the statistics data was recorded.

Statistics message data is comprised of PCF parameters that store the statistics information. The content of statistics messages depends on the message category as follows:

MQI statistics message
MQI statistics message data consists of a number of PCF parameters, but no PCF groups.

The parameters contained in MQI statistics message data are described in "MQI statistics message data" on page 279.

Queue statistics message
Queue statistics message data consists of a number of PCF parameters, and between one and one hundred *QStatisticsData* PCF groups.

There is one *QStatisticsData* PCF group for every queue was active in the interval. If more than 100 queues were active in the interval, multiple statistics messages are generated. Each message has the *SeqNumber* in the MQCFH (PCF header) updated accordingly, and the last message in the sequence has the *Control* parameter in the MQCFH specified as MQCFC_LAST.

The parameters contained in queue statistics message data are described in "Queue statistics message data" on page 284.

Channel statistics message
Channel statistics message data consists of a number of PCF parameters, and between one and one hundred *ChlStatisticsData* PCF groups.

There is one *ChlStatisticsData* PCF group for every channel that was active in the interval. If more than 100 channels were active in the interval, multiple statistics messages are generated. Each message has the *SeqNumber* in the MQCFH (PCF header) updated accordingly, and the last message in the sequence has the *Control* parameter in the MQCFH specified as MQCFC_LAST.

The parameters contained in channel statistics message data are described in "Channel statistics message data" on page 290.

Displaying accounting and statistics information

Accounting and statistics messages are written to the system accounting and statistics queues. In order to use the information recorded in these messages, you must use an application to transform the recorded information into a format suitable to be used.

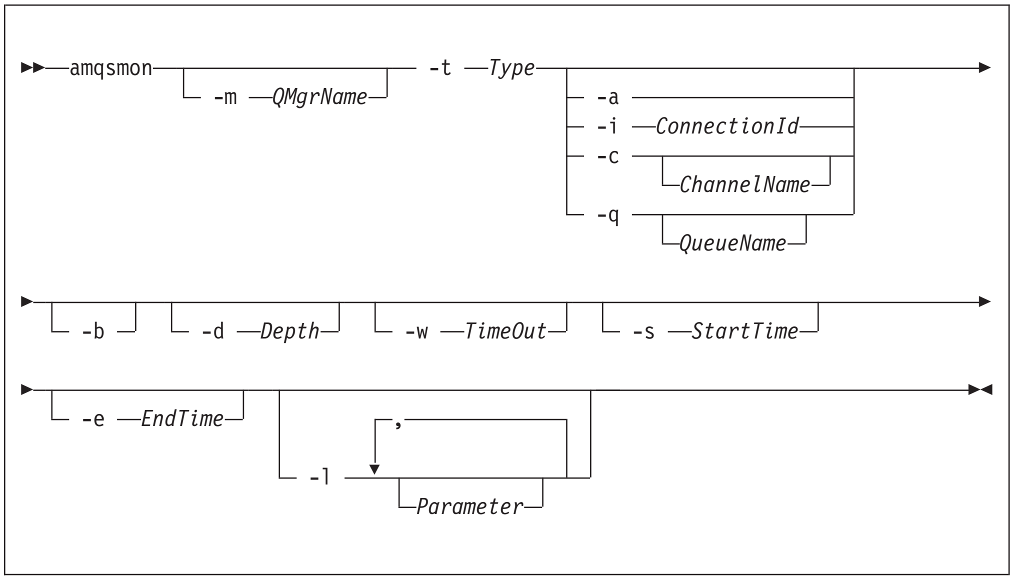
amqsmon is a sample supplied with WebSphere MQ which processes messages from the accounting and statistics queues and displays the information to the screen in a readable form.

As **amqsmon** is a sample program, you can use the supplied source code as template for writing your own application to process accounting or statistics messages, or modify the **amqsmon** source code to meet your own particular requirements.

amqsmon (Display formatted monitoring information)

Use **amqsmon** to display the information contained within accounting and statistics messages in a formatted form. Accounting messages are read from the accounting queue, SYSTEM.ADMIN.ACCOUNTING.QUEUE. Statistics messages are read from the statistics queue, SYSTEM.ADMIN.STATISTICS.QUEUE.

Syntax:



Required parameters:

-t *Type*
: The type of messages to process. Specify *Type* as one of the following:

accounting
: Accounting records are processed. Messages are read from the system queue, SYSTEM.ADMIN.ACCOUNTING.QUEUE.

statistics
: Statistics records are processed. Messages are read from the system queue, SYSTEM.ADMIN.STATISTICS.QUEUE.

Optional Parameters:

-m *QMgrName*
: The name of the queue manager from which accounting or statistics messages are to be processed.

  If you do not specify this parameter, the default queue manager is used.

-a Process messages containing MQI records only.

  Only display MQI records. Messages not containing MQI records will always be left on the queue they were read from.

-q *QueueName*
: *QueueName* is an optional parameter.

If *QueueName* is not supplied:	Displays queue accounting and queue statistics records only.
If *QueueName* is supplied:	Displays queue accounting and queue statistics records for the queue specified by *QueueName* only. If *-b* is not specified then the accounting and statistics messages from which the records came are discarded. Since accounting and statistics messages can also contain records from other queues, if *-b* is not specified then unseen records can be discarded.

-c *ChannelName*
: *ChannelName* is an optional parameter.

If *ChannelName* is not supplied:	Displays channel statistics records only.
If *ChannelName* is supplied:	Displays channel statistics records for the channel specified by *ChannelName* only. If *-b* is not specified then the statistics messages from which the records came are discarded. Since statistics messages can also contain records from other channels, if *-b* is not specified then unseen records can be discarded.

  This parameter is available when displaying statistics messages only, (*-t statistics*).

-i *ConnectionId*
: Displays records related to the connection identifier specified by *ConnectionId* only.

  This parameter is available when displaying accounting messages only, (*-t accounting*).

  If *-b* is not specified then the statistics messages from which the records came are discarded. Since statistics messages can also contain records from other channels, if *-b* is not specified then unseen records can be discarded.

-b Browse messages.

  Messages are retrieved non-destructively.

-d *Depth*
: The maximum number of messages that can be processed.

  If you do not specify this parameter, then an unlimited number of messages can be processed.

-w *TimeOut*
: Time maximum number of seconds to wait for a message to become available.

  If you do not specify this parameter, **amqsmon** will end once there are no more messages to process.

-s *StartTime*
: Process messages put after the specified *StartTime* only.

  StartTime is specified in the format `yyyy-mm-dd hh.mm.ss`. If a date is specified without a time, then the time will default to `00.00.00` on the date specified. Times are in GMT.

  For the effect of not specifying this parameter, see Note 1.

-e *EndTime*
: Process messages put before the specified *EndTime* only.

  The *EndTime* is specified in the format `yyyy-mm-dd hh.mm.ss`. If a date is specified without a time, then the time will default to `23.59.59` on the date specified. Times are in GMT.

  For the effect of not specifying this parameter, see Note 1.

-l *Parameter*
: Only display the selected fields from the records processed. *Parameter* is a comma-separated list of integer values, with each integer value mapping to the numerical constant of a field, see amqsmon example 5.

  If you do not specify this parameter, then all available fields are displayed.

Note:

1. If you do not specify *-s StartTime* or *-e EndTime*, then the messages that can be processed are not restricted by put time.

Examples:

1. The following command displays all MQI statistics messages from queue manager `saturn.queue.manager`:

   ```
   amqsmon -m saturn.queue.manager -t statistics -a
   ```

   The output from this command follows:

   ```
   RecordType: MQIStatistics
   QueueManager: 'saturn.queue.manager'
   IntervalStartDate: '2005-04-30'
   IntervalStartTime: '15.09.02'
   IntervalEndDate: '2005-04-30'
   IntervalEndTime: '15.39.02'
   CommandLevel: 600
   ConnCount: 23
   ConnFailCount: 0
   ConnHighwater: 8
   DiscCount: [17, 0, 0]
   OpenCount: [0, 80, 1, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0]
   OpenFailCount: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
   CloseCount: [0, 73, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
   CloseFailCount: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
   InqCount: [4, 2102, 0, 0, 0, 46, 0, 0, 0, 0, 0, 0, 0]
   InqFailCount: [0, 31, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
   SetCount: [0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0]
   SetFailCount: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
   PutCount: [26, 1]
   PutFailCount: 0
   Put1Count: [40, 0]
   ```

```
   Put1FailCount: 0
   PutBytes: [57064, 12320]
   GetCount: [18, 1]
   GetBytes: [52, 12320]
   GetFailCount: 2254
   BrowseCount: [18, 60]
   BrowseBytes: [23784, 30760]
   BrowseFailCount: 9
   CommitCount: 0
   CommitFailCount: 0
   BackCount: 0
   ExpiredMsgCount: 0
   PurgeCount: 0
```

2. The following command displays all queue statistics messages for queue `LOCALQ` on queue manager `saturn.queue.manager`:

```
amqsmon -m saturn.queue.manager -t statistics -q LOCALQ
```

The output from this command follows:

```
   RecordType: QueueStatistics
   QueueManager: 'saturn.queue.manager'
   IntervalStartDate: '2005-04-30'
   IntervalStartTime: '15.09.02'
   IntervalEndDate: '2005-04-30'
   IntervalEndTime: '15.39.02'
   CommandLevel: 600
   ObjectCount: 3
   QueueStatistics:
     QueueName: 'LOCALQ'
     CreateDate: '2005-03-08'
     CreateTime: '17.07.02'
     QueueType: Predefined
     QueueDefinitionType: Local
     QMinDepth: 0
     QMaxDepth: 18
     AverageQueueTime: [29827281, 0]
     PutCount: [26, 0]
     PutFailCount: 0
     Put1Count: [0, 0]
     Put1FailCount: 0
     PutBytes: [88, 0]
     GetCount: [18, 0]
     GetBytes: [52, 0]
     GetFailCount: 0
     BrowseCount: [0, 0]
     BrowseBytes: [0, 0]
     BrowseFailCount: 1
     NonQueuedMsgCount: 0
     ExpiredMsgCount: 0
     PurgedMsgCount: 0
```

3. The following command displays all of the statistics messages recorded since 15:30 on 30 April 2005 from queue manager `saturn.queue.manager`.

```
amqsmon -m saturn.queue.manager -t statistics -s "2005-04-30 15.30.00"
```

The output from this command follows:

```
   RecordType: MQIStatistics
   QueueManager: 'saturn.queue.manager'
   IntervalStartDate: '2005-04-30'
   IntervalStartTime: '15.09.02'
   IntervalEndDate: '2005-04-30'
   IntervalEndTime: '15.39.02'
   CommandLevel: 600
   ConnCount: 23
   ConnFailCount: 0
```

```
ConnHighwater: 8
DiscCount: [17, 0, 0]
OpenCount: [0, 80, 1, 0, 0, 3, 0, 0, 0, 0, 0, 0, 0]
  ...
RecordType: QueueStatistics
QueueManager: 'saturn.queue.manager'
IntervalStartDate: '2005-04-30'
IntervalStartTime: '15.09.02'
IntervalEndDate: '2005-04-30'
IntervalEndTime: '15.39.02'
CommandLevel: 600
ObjectCount: 3
QueueStatistics: 0
  QueueName: 'LOCALQ'
  CreateDate: '2005-03-08'
  CreateTime: '17.07.02'
  QueueType: Predefined
    ...
QueueStatistics: 1
  QueueName: 'SAMPLEQ'
  CreateDate: '2005-03-08'
  CreateTime: '17.07.02'
  QueueType: Predefined
    ...
```

4. The following command displays all accounting messages recorded on 30 April 2005 from queue manager `saturn.queue.manager`:

   ```
   amqsmon -m saturn.queue.manager -t accounting -s "2005-04-30" -e "2005-04-30"
   ```

   The output from this command follows:

   ```
   RecordType: MQIAccounting
   QueueManager: 'saturn.queue.manager'
   IntervalStartDate: '2005-04-30'
   IntervalStartTime: '15.09.29'
   IntervalEndDate: '2005-04-30'
   IntervalEndTime: '15.09.30'
   CommandLevel: 600
   ConnectionId: x'414d5143545245563120202020202020208d0b3742010a0020'
   SeqNumber: 0
   ApplicationName: 'amqsput'
   ApplicationPid: 8572
   ApplicationTid: 1
   UserId: 'admin'
   ConnDate: '2005-03-16'
   ConnTime: '15.09.29'
   DiscDate: '2005-03-16'
   DiscTime: '15.09.30'
   DiscType: Normal
   OpenCount: [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
   OpenFailCount: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
   CloseCount: [0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
   CloseFailCount: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
   PutCount: [1, 0]
   PutFailCount: 0
   PutBytes: [4, 0]
   GetCount: [0, 0]
   GetFailCount: 0
   GetBytes: [0, 0]
   BrowseCount: [0, 0]
   BrowseFailCount: 0
   BrowseBytes: [0, 0]
   CommitCount: 0
   CommitFailCount: 0
   BackCount: 0
   InqCount: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
   InqFailCount: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
   ```

```
SetCount: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
SetFailCount: [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

RecordType: MQIAccounting
QueueManager: 'saturn.queue.manager'
IntervalStartDate: '2005-03-16'
IntervalStartTime: '15.16.22'
IntervalEndDate: '2005-03-16'
IntervalEndTime: '15.16.22'
CommandLevel: 600
ConnectionId: x'414d5143545245563120202020202020208d0b3742010c0020'
SeqNumber: 0
ApplicationName: 'runmqsc'
ApplicationPid: 8615
ApplicationTid: 1
    ...
```

5. The following command browses the accounting queue and displays the application name and connection identifier of every application for which MQI accounting information is available:

   ```
   amqsmon -m saturn.queue.manager -t accounting -b -a -l 7006,3024
   ```

   The output from this command follows:

   ```
   ConnectionId: x'414d5143545245563120202020202020208d0b374203090020'
   ApplicationName: 'runmqsc'

   ConnectionId: x'414d5143545245563120202020202020208d0b3742010a0020'
   ApplicationName: 'amqsput'

   ConnectionId: x'414d5143545245563120202020202020208d0b3742010c0020'
   ApplicationName: 'runmqsc'

   ConnectionId: x'414d5143545245563120202020202020208d0b3742010d0020'
   ApplicationName: 'amqsput'

   ConnectionId: x'414d5143545245563120202020202020208d0b3742150d0020'
   ApplicationName: 'amqsget'

   5 Records Processed.
   ```

Accounting and statistics message reference

This section provides an overview of the accounting and statistics message format. It describes the information returned in accounting messages, and in statistics messages.

Accounting and statistics message format

Accounting and statistics message messages are standard WebSphere MQ messages containing a message descriptor and message data. The message data contains information about the MQI operations performed by WebSphere MQ applications, or information about the activities occurring in a WebSphere MQ system. For information on accounting and statistics messages, see “Accounting and statistics messages” on page 245.

Accounting and statistics messages contain the following:

A message descriptor
An MQMD structure

Message data
- A PCF header (MQCFH)

- Accounting or statistics message data that is always returned
- Accounting or statistics message data that is returned if available

For example, the structure of an MQI accounting message is shown in Table 21.

Table 21. MQI accounting message structure

Message descriptor	Message data	
MQMD structure	Accounting message header MQCFH structure	MQI accounting message data [1]
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 Command identifier Message sequence number Control options Completion code Reason code Parameter count	Queue manager Interval start date Interval start time Interval end date Interval end time Command level Connection identifier Sequence number Application name Application process identifier Application thread identifier User identifier Connection date Connection time Connection name Channel name Disconnect date Disconnect time Disconnect type Open count Open fail count Close count Close fail count Put count Put fail count Put1 count Put1 fail count Put bytes Get count Get fail count Get bytes Browse count Browse fail count Browse bytes Commit count Commit fail count Backout count Inquire count Inquire fail count Set count Set fail count
Note: 1. The parameters shown are those returned for an MQI accounting message. The actual accounting or statistics message data depends on the message category.		

Accounting and statistics message MQMD (message descriptor)

The parameters and values in the message descriptor of accounting and statistics message are the same as in the message descriptor of event messages, with the following exception:

Format

Description:	Format name of message data.
Datatype:	MQCHAR8.
Value:	**MQFMT_ADMIN** Admin message.

Some of the parameters contained in the message descriptor of accounting and statistics message contain fixed data supplied by the queue manager that generated the message.

The parameters that make up the MQMD structure of event messages are described in “Event message MQMD (message descriptor)” on page 51. 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 message was put on the accounting, or statistics, queue.

Message data in accounting and statistics messages

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

The message data in accounting and statistics messages consists of two parts:
- A PCF header (MQCFH), see “Accounting and statistics message MQCFH (PCF header).”
- An accounting or statistics report, see “Accounting and statistics message data” on page 263.

Accounting and statistics message MQCFH (PCF header)

The message header of accounting and statistics messages is an MQCFH structure. The parameters and values in the message header of accounting and statistics message are the same as in the message header of event messages, with the following exceptions:

Command

Description:	Command identifier. This identifies the accounting or statistics message category.
Datatype:	MQLONG.

Values:	**MQCMD_ACCOUNTING_MQI** MQI accounting message. **MQCMD_ACCOUNTING_Q** Queue accounting message. **MQCMD_STATISTICS_MQI** MQI statistics message. **MQCMD_STATISTICS_Q** Queue statistics message. **MQCMD_STATISTICS_CHANNEL** Channel statistics message.

Version

Description:	Structure version number.
Datatype:	MQLONG.
Value:	**MQCFH_VERSION_3** Version-3 for accounting and statistics messages.

The parameters that make up the MQCFH structure of event messages are described in "Event message MQCFH (PCF header)" on page 56.

Accounting and statistics message data

The content of accounting and statistics message data is dependent on the category of the accounting or statistics message, as follows:

MQI accounting message
MQI accounting message data consists of a number of PCF parameters, but no PCF groups.

The parameters contained in MQI accounting message data are described in "MQI accounting message data" on page 264.

Queue accounting message
Queue accounting message data consists of a number of PCF parameters, and between one and one hundred *QAccountingData* PCF groups.

The parameters contained in queue accounting message data are described in "Queue accounting message data" on page 270.

MQI statistics message
MQI statistics message data consists of a number of PCF parameters, but no PCF groups.

The parameters contained in MQI statistics message data are described in "MQI statistics message data" on page 279.

Queue statistics message
Queue statistics message data consists of a number of PCF parameters, and between one and one hundred *QStatisticsData* PCF groups.

The parameters contained in queue statistics message data are described in "Queue statistics message data" on page 284.

Channel statistics message
Channel statistics message data consists of a number of PCF parameters, and between one and one hundred *ChlStatisticsData* PCF groups.

The parameters contained in channel statistics message data are described in “Channel statistics message data” on page 290.

MQI accounting message data

Message name:	MQI accounting message.
Platforms:	All, except WebSphere MQ for z/OS.
System queue:	SYSTEM.ADMIN.ACCOUNTING.QUEUE.

Accounting message data

QueueManager

Description:	The name of the queue manager
Identifier:	MQCA_Q_MGR_NAME
Datatype:	MQCFST
Maximum length:	MQ_Q_MGR_NAME_LENGTH
Returned:	Always

IntervalStartDate

Description:	The date of the start of the monitoring period
Identifier:	MQCAMO_START_DATE
Datatype:	MQCFST
Maximum length:	MQ_DATE_LENGTH
Returned:	Always

IntervalStartTime

Description:	The time of the start of the monitoring period
Identifier:	MQCAMO_START_TIME
Datatype:	MQCFST
Maximum length:	MQ_TIME_LENGTH
Returned:	Always

IntervalEndDate

Description:	The date of the end of the monitoring period
Identifier:	MQCAMO_END_DATE
Datatype:	MQCFST
Maximum length:	MQ_DATE_LENGTH
Returned:	Always

IntervalEndTime

Description:	The time of the end of the monitoring period
Identifier:	MQCAMO_END_TIME
Datatype:	MQCFST
Maximum length:	MQ_TIME_LENGTH
Returned:	Always

CommandLevel

Description:	The queue manager command level
Identifier:	MQIA_COMMAND_LEVEL

Datatype:	MQCFIN
Returned:	Always

ConnectionId

Description:	The connection identifier for the WebSphere MQ connection
Identifier:	MQBACF_CONNECTION_ID
Datatype:	MQCFBS
Maximum length:	MQ_CONNECTION_ID_LENGTH
Returned:	Always

SeqNumber

Description:	The sequence number. This value is incremented for each subsequent record for long running connections.
Identifier:	MQIACF_SEQUENCE_NUMBER
Datatype:	MQCFIN
Returned:	Always

ApplicationName

Description:	The name of the application. The contents of this field are equivalent to the contents of the *PutApplName* field in the message descriptor.
Identifier:	MQCACF_APPL_NAME
Datatype:	MQCFST
Maximum length:	MQ_APPL_NAME_LENGTH
Returned:	Always

ApplicationPid

Description:	The operating system process identifier of the application
Identifier:	MQIACF_PROCESS_ID
Datatype:	MQCFIN
Returned:	Always

ApplicationTid

Description:	The WebSphere MQ thread identifier of the connection in the application
Identifier:	MQIACF_THREAD_ID
Datatype:	MQCFIN
Returned:	Always

UserId

Description:	The user identifier context of the application
Identifier:	MQCACF_USER_IDENTIFIER
Datatype:	MQCFST
Maximum length:	MQ_USER_ID_LENGTH
Returned:	Always

ConnDate

Description:	Date of MQCONN operation
Identifier:	MQCAMO_CONN_DATE
Datatype:	MQCFST

Maximum length:	MQ_TIME_LENGTH
Returned:	When available

ConnTime

Description:	Time of MQCONN operation
Identifier:	MQCAMO_CONN_TIME
Datatype:	MQCFST
Maximum length:	MQ_TIME_LENGTH
Returned:	When available

ConnName

Description:	Connection name for client connection
Identifier:	MQCAMO_CONNECTION_NAME
Datatype:	MQCFST
Maximum length:	MQ_CONN_NAME_LENGTH
Returned:	When available

ChannelName

Description:	Channel name for client connection
Identifier:	MQCACH_CHANNEL_NAME
Datatype:	MQCFST
Maximum length:	MQ_CHANNEL_NAME_LENGTH
Returned:	When available

DiscDate

Description:	Date of MQDISC operation
Identifier:	MQCAMO_DISC_DATE
Datatype:	MQCFST
Maximum length:	MQ_DATE_LENGTH
Returned:	When available

DiscTime

Description:	Time of MQDISC operation
Identifier:	MQCAMO_DISC_TIME
Datatype:	MQCFST
Maximum length:	MQ_TIME_LENGTH
Returned:	When available

DiscType

Description:	Type of disconnect
Identifier:	MQIAMO_DISC_TYPE
Datatype:	MQCFIN

Values:	The possible values are:
	MQDISCONNECT_NORMAL Requested by application
	MQDISCONNECT_IMPLICIT Abnormal application termination
	MQDISCONNECT_Q_MGR Connection broken by queue manager
Returned:	When available

OpenCount

Description:	The number of objects opened. This parameter is an integer list indexed by object type, see Note 1.
Identifier:	MQIAMO_OPENS
Datatype:	MQCFIL
Returned:	When available

OpenFailCount

Description:	The number of unsuccessful attempts to open an object. This parameter is an integer list indexed by object type, see Note 1.
Identifier:	MQIAMO_OPENS_FAILED
Datatype:	MQCFIL
Returned:	When available

CloseCount

Description:	The number of objects closed. This parameter is an integer list indexed by object type, see Note 1.
Identifier:	MQIAMO_CLOSES
Datatype:	MQCFIL
Returned:	When available

CloseFailCount

Description:	The number of unsuccessful attempts to close an object. This parameter is an integer list indexed by object type, see Note 1.
Identifier:	MQIAMO_CLOSES_FAILED
Datatype:	MQCFIL
Returned:	When available

PutCount

Description:	The number persistent and nonpersistent messages successfully put to a queue, with the exception of messages put using the MQPUT1 call. This parameter is an integer list indexed by persistence value, see Note 2.
Identifier:	MQIAMO_PUTS
Datatype:	MQCFIL
Returned:	When available

PutFailCount

Description:	The number of unsuccessful attempts to put a message
Identifier:	MQIAMO_PUTS_FAILED

Datatype:	MQCFIN
Returned:	When available

Put1Count

Description:	The number of persistent and nonpersistent messages successfully put to the queue using MQPUT1 calls. This parameter is an integer list indexed by persistence value, see Note 2.
Identifier:	MQIAMO_PUT1S
Datatype:	MQCFIL
Included in PCF group:	*QAccountingData*
Returned:	When available

Put1FailCount

Description:	The number of unsuccessful attempts to put a message using MQPUT1 calls
Identifier:	MQIAMO_PUT1S_FAILED
Datatype:	MQCFIN
Included in PCF group:	*QAccountingData*
Returned:	When available

PutBytes

Description:	The number bytes written using put calls for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Note 2.
Identifier:	MQIAMO64_PUT_BYTES
Datatype:	MQCFIL64
Returned:	When available

GetCount

Description:	The number of successful destructive MQGET calls for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Note 2.
Identifier:	MQIAMO_GETS
Datatype:	MQCFIL
Returned:	When available

GetFailCount

Description:	The number of failed destructive MQGET calls
Identifier:	MQIAMO_GETS_FAILED
Datatype:	MQCFIN
Returned:	When available

GetBytes

Description:	Total number of bytes retrieved for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Note 2.
Identifier:	MQIAMO64_GET_BYTES
Datatype:	MQCFIL64

Returned:	When available

BrowseCount

Description:	The number of successful non-destructive MQGET calls for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Note 2.
Identifier:	MQIAMO_BROWSES
Datatype:	MQCFIL
Returned:	When available

BrowseFailCount

Description:	The number of unsuccessful non-destructive MQGET calls
Identifier:	MQIAMO_BROWSES_FAILED
Datatype:	MQCFIN
Returned:	When available

BrowseBytes

Description:	Total number of bytes browsed for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Note 2.
Identifier:	MQIAMO64_BROWSE_BYTES
Datatype:	MQCFIL64
Returned:	When available

CommitCount

Description:	The number of successful transactions. This number includes those transactions committed implicitly by the connected application. Commit requests where there is no outstanding work are included in this count.
Identifier:	MQIAMO_COMMITS
Datatype:	MQCFIN
Returned:	When available

CommitFailCount

Description:	The number of unsuccessful attempts to complete a transaction
Identifier:	MQIAMO_COMMITS_FAILED
Datatype:	MQCFIN
Returned:	When available

BackCount

Description:	The number of backouts processed, including implicit backouts due to abnormal disconnection
Identifier:	MQIAMO_BACKOUTS
Datatype:	MQCFIN
Returned:	When available

InqCount

Description:	The number of successful objects inquired upon. This parameter is an integer list indexed by object type, see Note 1.

Identifier:	MQIAMO_INQS
Datatype:	MQCFIL
Returned:	When available

InqFailCount

Description:	The number of unsuccessful object inquire attempts. This parameter is an integer list indexed by object type, see Note 1.
Identifier:	MQIAMO_INQS_FAILED
Datatype:	MQCFIL
Returned:	When available

SetCount

Description:	The number of successful MQSET calls. This parameter is an integer list indexed by object type, see Note 1.
Identifier:	MQIAMO_SETS
Datatype:	MQCFIL
Returned:	When available

SetFailCount

Description:	The number of unsuccessful MQSET calls. This parameter is an integer list indexed by object type, see Note 1.
Identifier:	MQIAMO_SETS_FAILED
Datatype:	MQCFIL
Returned:	When available

Queue accounting message data

Message name:	Queue accounting message.
Platforms:	All, except WebSphere MQ for z/OS.
System queue:	SYSTEM.ADMIN.ACCOUNTING.QUEUE.

Accounting message data

QueueManager

Description:	The name of the queue manager
Identifier:	MQCA_Q_MGR_NAME
Datatype:	MQCFST
Maximum length:	MQ_Q_MGR_NAME_LENGTH
Returned:	Always

IntervalStartDate

Description:	The date of the start of the monitoring period
Identifier:	MQCAMO_START_DATE
Datatype:	MQCFST
Maximum length:	MQ_DATE_LENGTH
Returned:	Always

IntervalStartTime

Description:	The time of the start of the monitoring period
Identifier:	MQCAMO_START_TIME
Datatype:	MQCFST
Maximum length:	MQ_TIME_LENGTH
Returned:	Always

IntervalEndDate

Description:	The date of the end of the monitoring period
Identifier:	MQCAMO_END_DATE
Datatype:	MQCFST
Maximum length:	MQ_DATE_LENGTH
Returned:	Always

IntervalEndTime

Description:	The time of the end of the monitoring period
Identifier:	MQCAMO_END_TIME
Datatype:	MQCFST
Maximum length:	MQ_TIME_LENGTH
Returned:	Always

CommandLevel

Description:	The queue manager command level
Identifier:	MQIA_COMMAND_LEVEL
Datatype:	MQCFIN
Returned:	Always

ConnectionId

Description:	The connection identifier for the WebSphere MQ connection
Identifier:	MQBACF_CONNECTION_ID
Datatype:	MQCFBS
Maximum length:	MQ_CONNECTION_ID_LENGTH
Returned:	Always

SeqNumber

Description:	The sequence number. This value is incremented for each subsequent record for long running connections.
Identifier:	MQIACF_SEQUENCE_NUMBER
Datatype:	MQCFIN
Returned:	Always

ApplicationName

Description:	The name of the application. The contents of this field are equivalent to the contents of the PutApplName field in the message descriptor.
Identifier:	MQCACF_APPL_NAME
Datatype:	MQCFST
Maximum length:	MQ_APPL_NAME_LENGTH
Returned:	Always

ApplicationPid

Description:	The operating system process identifier of the application
Identifier:	MQIACF_PROCESS_ID
Datatype:	MQCFIN
Returned:	Always

ApplicationTid

Description:	The WebSphere MQ thread identifier of the connection in the application
Identifier:	MQIACF_THREAD_ID
Datatype:	MQCFIN
Returned:	Always

UserId

Description:	The user identifier context of the application
Identifier:	MQCACF_USER_IDENTIFIER
Datatype:	MQCFST
Maximum length:	MQ_USER_ID_LENGTH
Returned:	Always

ObjectCount

Description:	The number of queues accessed in the interval for which accounting data has been recorded. This value is set to the number of *QAccountingData* PCF groups contained in the message.
Identifier:	MQIAMO_OBJECT_COUNT
Datatype:	MQCFIN
Returned:	Always

QAccountingData

Description:	Grouped parameters specifying accounting details for a queue
Identifier:	MQGACF_Q_ACCOUNTING_DATA
Datatype:	MQCFGR

Parameters in group:	*QName* *CreateDate* *CreateDate* *QType* *QDefinitionType* *OpenCount* *OpenDate* *OpenTime* *CloseDate* *CloseTime* *PutCount* *PutFailCount* *Put1Count* *Put1FailCount* *PutBytes* *PutMinBytes* *PutMaxBytes* *GetCount* *GetFailCount* *GetBytes* *GetMinBytes* *GetMaxBytes* *BrowseCount* *BrowseFailCount* *BrowseBytes* *BrowseMinBytes* *BrowseMaxBytes* *TimeOnQMin* *TimeOnQAvg* *TimeOnQMax*
Returned:	Always

QName

Description:	The name of the queue
Identifier:	MQCA_Q_NAME
Datatype:	MQCFST
Included in PCF group:	*QAccountingData*
Maximum length:	MQ_Q_NAME_LENGTH
Returned:	When available

CreateDate

Description:	The date the queue was created
Identifier:	MQCA_CREATION_DATE
Datatype:	MQCFST
Included in PCF group:	*QAccountingData*
Maximum length:	MQ_DATE_LENGTH

Returned:	When available

CreateTime

Description:	The time the queue was created
Identifier:	MQCA_CREATION_TIME
Datatype:	MQCFST
Included in PCF group:	*QAccountingData*
Maximum length:	MQ_TIME_LENGTH
Returned:	When available

QType

Description:	The type of the queue
Identifier:	MQIA_Q_TYPE
Datatype:	MQCFIN
Included in PCF group:	*QAccountingData*
Value:	MQQT_LOCAL
Returned:	When available

QDefinitionType

Description:	The queue definition type
Identifier:	MQIA_DEFINITION_TYPE
Datatype:	MQCFIN
Included in PCF group:	*QAccountingData*
Values:	Possible values are: **MQQDT_PREDEFINED** **MQQDT_PERMANENT_DYNAMIC** **MQQDT_TEMPORARY_DYNAMIC**
Returned:	When available

OpenCount

Description:	The number of times this queue was opened by the application in this interval
Identifier:	MQIAMO_OPENS
Datatype:	MQCFIL
Included in PCF group:	*QAccountingData*
Returned:	When available

OpenDate

Description:	The date the queue was first opened in this recording interval. If the queue was already open at the start of this interval, this value reflects the date the queue was originally opened.
Identifier:	MQCAMO_OPEN_DATE
Datatype:	MQCFST

Included in PCF group:	*QAccountingData*
Returned:	When available

OpenTime

Description:	The time the queue was first opened in this recording interval. If the queue was already open at the start of this interval, this value reflects the time the queue was originally opened.
Identifier:	MQCAMO_OPEN_TIME
Datatype:	MQCFST
Included in PCF group:	*QAccountingData*
Returned:	When available

CloseDate

Description:	The date of the final close of the queue in this recording interval. If the queue is still open then the value is not returned.
Identifier:	MQCAMO_CLOSE_DATE
Datatype:	MQCFST
Included in PCF group:	*QAccountingData*
Returned:	When available

CloseTime

Description:	The time of final close of the queue in this recording interval. If the queue is still open then the value is not returned.
Identifier:	MQCAMO_CLOSE_TIME
Datatype:	MQCFST
Included in PCF group:	*QAccountingData*
Returned:	When available

PutCount

Description:	The number of persistent and nonpersistent messages successfully put to the queue, with the exception of MQPUT1 calls. This parameter is an integer list indexed by persistence value, see Note 2.
Identifier:	MQIAMO_PUTS
Datatype:	MQCFIL
Included in PCF group:	*QAccountingData*
Returned:	When available

PutFailCount

Description:	The number of unsuccessful attempts to put a message, with the exception of MQPUT1 calls
Identifier:	MQIAMO_PUTS_FAILED
Datatype:	MQCFIN
Included in PCF group:	*QAccountingData*
Returned:	When available

Put1Count

Description:	The number of persistent and nonpersistent messages successfully put to the queue using MQPUT1 calls. This parameter is an integer list indexed by persistence value, see Note 2.
Identifier:	MQIAMO_PUT1S
Datatype:	MQCFIL
Included in PCF group:	*QAccountingData*
Returned:	When available

Put1FailCount

Description:	The number of unsuccessful attempts to put a message using MQPUT1 calls
Identifier:	MQIAMO_PUT1S_FAILED
Datatype:	MQCFIN
Included in PCF group:	*QAccountingData*
Returned:	When available

PutBytes

Description:	The total number of bytes put for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Note 2.
Identifier:	MQIAMO64_PUT_BYTES
Datatype:	MQCFIL64
Included in PCF group:	*QAccountingData*
Returned:	When available

PutMinBytes

Description:	The smallest persistent and nonpersistent message size placed on the queue. This parameter is an integer list indexed by persistence value, see Note 2.
Identifier:	MQIAMO_PUT_MIN_BYTES
Datatype:	MQCFIL
Included in PCF group:	*QAccountingData*
Returned:	When available

PutMaxBytes

Description:	The largest persistent and nonpersistent message size placed on the queue. This parameter is an integer list indexed by persistence value, see Note 2.
Identifier:	MQIAMO_PUT_MAX_BYTES
Datatype:	MQCFIL
Included in PCF group:	*QAccountingData*
Returned:	When available

GetCount

Description:	The number of successful destructive MQGET calls for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Note 2.
Identifier:	MQIAMO_GETS
Datatype:	MQCFIL
Included in PCF group:	*QAccountingData*
Returned:	When available

GetFailCount

Description:	The number of failed destructive MQGET calls
Identifier:	MQIAMO_GETS_FAILED
Datatype:	MQCFIN
Included in PCF group:	*QAccountingData*
Returned:	When available

GetBytes

Description:	The number of bytes read in destructive MQGET calls for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Note 2.
Identifier:	MQIAMO64_GET_BYTES
Datatype:	MQCFIL64
Included in PCF group:	*QAccountingData*
Returned:	When available

GetMinBytes

Description:	The size of the smallest persistent and nonpersistent message retrieved rom the queue. This parameter is an integer list indexed by persistence value, see Note 2.
Identifier:	MQIAMO_GET_MIN_BYTES
Datatype:	MQCFIL
Included in PCF group:	*QAccountingData*
Returned:	When available

GetMaxBytes

Description:	The size of the largest persistent and nonpersistent message retrieved rom the queue. This parameter is an integer list indexed by persistence value, see Note 2.
Identifier:	MQIAMO_GET_MAX_BYTES
Datatype:	MQCFIL
Included in PCF group:	*QAccountingData*
Returned:	When available

BrowseCount

Description:	The number of successful non-destructive MQGET calls for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Note 2.
Identifier:	MQIAMO_BROWSES
Datatype:	MQCFIL
Included in PCF group:	*QAccountingData*
Returned:	When available

BrowseFailCount

Description:	The number of unsuccessful non-destructive MQGET calls
Identifier:	MQIAMO_BROWSES_FAILED
Datatype:	MQCFIN
Included in PCF group:	*QAccountingData*
Returned:	When available

BrowseBytes

Description:	The number of bytes read in non-destructive MQGET calls that returned persistent messages
Identifier:	MQIAMO64_BROWSE_BYTES
Datatype:	MQCFIL64
Included in PCF group:	*QAccountingData*
Returned:	When available

BrowseMinBytes

Description:	The size of the smallest persistent and nonpersistent message browsed from the queue. This parameter is an integer list indexed by persistence value, see Note 2.
Identifier:	MQIAMO_BROWSE_MIN_BYTES
Datatype:	MQCFIL
Included in PCF group:	*QAccountingData*
Returned:	When available

BrowseMaxBytes

Description:	The size of the largest persistent and nonpersistent message browsed from the queue. This parameter is an integer list indexed by persistence value, see Note 2.
Identifier:	MQIAMO_BROWSE_MAX_BYTES
Datatype:	MQCFIL
Included in PCF group:	*QAccountingData*
Returned:	When available

TimeOnQMin

Description:	The shortest time a persistent and nonpersistent message remained on the queue before being destructively retrieved, in microseconds. For messages retrieved under syncpoint this value does not included the time before the get operation is committed. This parameter is an integer list indexed by persistence value, see Note 2.
Identifier:	MQIAMO_Q_TIME_MIN
Datatype:	MQCFIL64
Included in PCF group:	*QAccountingData*
Returned:	When available

TimeOnQAvg

Description:	The average time a persistent and nonpersistent message remained on the queue before being destructively retrieved, in microseconds. For messages retrieved under syncpoint this value does not included the time before the get operation is committed. This parameter is an integer list indexed by persistence value, see Note 2.
Identifier:	MQIAMO_Q_TIME_AVG
Datatype:	MQCFIL64
Included in PCF group:	*QAccountingData*
Returned:	When available

TimeOnQMax

Description:	The longest time a persistent and nonpersistent message remained on the queue before being destructively retrieved, in microseconds. For messages retrieved under syncpoint this value does not included the time before the get operation is committed. This parameter is an integer list indexed by persistence value, see Note 2.
Identifier:	MQIAMO_Q_TIME_MAX
Datatype:	MQCFIL64
Included in PCF group:	*QAccountingData*
Returned:	When available

MQI statistics message data

Message name:	MQI statistics message.
Platforms:	All, except WebSphere MQ for z/OS.
System queue:	SYSTEM.ADMIN.STATISTICS.QUEUE.

Statistics message data

QueueManager

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

IntervalStartDate

Description:	The date at the start of the monitoring period.
Identifier:	MQCAMO_START_DATE.
Datatype:	MQCFST.
Maximum length:	MQ_DATE_LENGTH
Returned:	Always.

IntervalStartTime

Description:	The time at the start of the monitoring period.
Identifier:	MQCAMO_START_TIME.
Datatype:	MQCFST.
Maximum length:	MQ_TIME_LENGTH
Returned:	Always.

IntervalEndDate

Description:	The date at the end of the monitoring period.
Identifier:	MQCAMO_END_DATE.
Datatype:	MQCFST.
Maximum length:	MQ_DATE_LENGTH
Returned:	Always.

IntervalEndTime

Description:	The time at the end of the monitoring period.
Identifier:	MQCAMO_END_TIME.
Datatype:	MQCFST.
Maximum length:	MQ_TIME_LENGTH
Returned:	Always.

CommandLevel

Description:	The queue manager command level.
Identifier:	MQIA_COMMAND_LEVEL.
Datatype:	MQCFIN.
Returned:	Always.

ConnCount

Description:	The number of successful connections to the queue manager.
Identifier:	MQIAMO_CONNS.
Datatype:	MQCFIN.
Returned:	When available.

ConnFailCount

Description:	The number of unsuccessful connection attempts.
Identifier:	MQIAMO_CONNS_FAILED.
Datatype:	MQCFIN.
Returned:	When available.

ConnsMax

Description:	The maximum number of concurrent connections in the recording interval.
Identifier:	MQIAMO_CONNS_MAX.
Datatype:	MQCFIN.
Returned:	When available.

DiscCount

Description:	The number of disconnects from the queue manager. This is an integer array, indexed by the following constants: • MQDISCONNECT_NORMAL • MQDISCONNECT_IMPLICIT • MQDISCONNECT_Q_MGR
Identifier:	MQIAMO_DISCS.
Datatype:	MQCFIL.
Returned:	When available.

OpenCount

Description:	The number of objects successfully opened. This parameter is an integer list indexed by object type, see Note 1.
Identifier:	MQIAMO_OPENS.
Datatype:	MQCFIL.
Returned:	When available.

OpenFailCount

Description:	The number of unsuccessful open object attempts. This parameter is an integer list indexed by object type, see Note 1.
Identifier:	MQIAMO_OPENS_FAILED.
Datatype:	MQCFIL.
Returned:	When available.

CloseCount

Description:	The number of objects successfully closed. This parameter is an integer list indexed by object type, see Note 1.
Identifier:	MQIAMO_CLOSES.
Datatype:	MQCFIL.
Returned:	When available.

CloseFailCount

Description:	The number of unsuccessful close object attempts. This parameter is an integer list indexed by object type, see Note 1.
Identifier:	MQIAMO_CLOSES_FIALED.
Datatype:	MQCFIL.
Returned:	When available.

InqCount

Description:	The number of objects successfully inquired upon. This parameter is an integer list indexed by object type, see Note 1.

Identifier:	MQIAMO_INQS.
Datatype:	MQCFIL.
Returned:	When available.

InqFailCount

Description:	The number of unsuccessful object inquire attempts. This parameter is an integer list indexed by object type, see Note 1.
Identifier:	MQIAMO_INQS_FAILED.
Datatype:	MQCFIL.
Returned:	When available.

SetCount

Description:	The number of objects successfully updated (SET). This parameter is an integer list indexed by object type, see Note 1.
Identifier:	MQIAMO_SETS.
Datatype:	MQCFIL.
Returned:	When available.

SetFailCount

Description:	The number of unsuccessful SET attempts. This parameter is an integer list indexed by object type, see Note 1.
Identifier:	MQIAMO_SETS_FAILED.
Datatype:	MQCFIL.
Returned:	When available.

PutCount

Description:	The number of persistent and nonpersistent messages successfully put to a queue, with the exception of MQPUT1 requests. This parameter is an integer list indexed by persistence value, see Note 2.
Identifier:	MQIAMO_PUTS.
Datatype:	MQCFIL.
Returned:	When available.

PutFailCount

Description:	The number of unsuccessful put message attempts.
Identifier:	MQIAMO_PUTS_FAILED.
Datatype:	MQCFIN.
Returned:	When available.

Put1Count

Description:	The number of persistent and nonpersistent messages successfully put to a queue using MQPUT1 requests. This parameter is an integer list indexed by persistence value, see Note 2
Identifier:	MQIAMO_PUT1S.
Datatype:	MQCFIL.
Returned:	When available.

Put1FailCount

Description:	The number of unsuccessful attempts to put a persistent and nonpersistent message to a queue using MQPUT1 requests. This parameter is an integer list indexed by persistence value, see Note 2
Identifier:	MQIAMO_PUT1S_FAILED.
Datatype:	MQCFIN.
Returned:	When available.

PutBytes

Description:	The number bytes for persistent and nonpersistent messages written in using put requests. This parameter is an integer list indexed by persistence value, see Note 2
Identifier:	MQIAMO64_PUT_BYTES.
Datatype:	MQCFIL64.
Returned:	When available.

GetCount

Description:	The number of successful destructive get requests for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Note 2
Identifier:	MQIAMO_GETS.
Datatype:	MQCFIL.
Returned:	When available.

GetFailCount

Description:	The number of unsuccessful destructive get requests.
Identifier:	MQIAMO_GETS_FAILED.
Datatype:	MQCFIN.
Returned:	When available.

GetBytes

Description:	The number of bytes read in destructive gets requests for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Note 2
Identifier:	MQIAMO64_GET_BYTES.
Datatype:	MQCFIL64.
Returned:	When available.

BrowseCount

Description:	The number of successful non-destructive get requests for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Note 2
Identifier:	MQIAMO_BROWSES.
Datatype:	MQCFIL.
Returned:	When available.

BrowseFailCount

Description:	The number of unsuccessful non-destructive get requests.
Identifier:	MQIAMO_BROWSES_FAILED.

Datatype:	MQCFIN.
Returned:	When available.

BrowseBytes

Description:	The number of bytes read in non-destructive get requests for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Note 2
Identifier:	MQIAMO64_BROWSE_BYTES.
Datatype:	MQCFIL64.
Returned:	When available.

CommitCount

Description:	The number of transactions successfully completed. This number includes transactions committed implicitly by the application disconnecting, and commit requests where there is no outstanding work.
Identifier:	MQIAMO_COMMITS.
Datatype:	MQCFIN.
Returned:	When available.

CommitFailCount

Description:	The number of unsuccessful attempts to complete a transaction.
Identifier:	MQIAMO_COMMITS_FAILED.
Datatype:	MQCFIN.
Returned:	When available.

BackCount

Description:	The number of backouts processed, including implicit backout upon abnormal disconnect.
Identifier:	MQIAMO_BACKOUTS.
Datatype:	MQCFIN.
Returned:	When available.

ExpiredMsgCount

Description:	The number of persistent and nonpersistent messages that were discarded because they had expired, before they could be retrieved. This parameter is an integer list indexed by persistence value, see Note 2
Identifier:	MQIAMO_MSGS_EXPIRED.
Datatype:	MQCFIL.
Returned:	When available.

Queue statistics message data

Message name:	Queue statistics message.
Platforms:	All, except WebSphere MQ for z/OS.
System queue:	SYSTEM.ADMIN.STATISTICS.QUEUE.

Statistics message data

QueueManager

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

IntervalStartDate

Description:	The date at the start of the monitoring period
Identifier:	MQCAMO_START_DATE
Datatype:	MQCFST
Maximum length:	MQ_DATE_LENGTH
Returned:	Always

IntervalStartTime

Description:	The time at the start of the monitoring period
Identifier:	MQCAMO_START_TIME
Datatype:	MQCFST
Maximum length:	MQ_TIME_LENGT
Returned:	Always

IntervalEndDate

Description:	The date at the end of the monitoring period
Identifier:	MQCAMO_END_DATE
Datatype:	MQCFST
Maximum length:	MQ_DATE_LENGTH
Returned:	Always

IntervalEndTime

Description:	The time at the end of the monitoring period
Identifier:	MQCAMO_END_TIME
Datatype:	MQCFST
Maximum length:	MQ_TIME_LENGTH
Returned:	Always

CommandLevel

Description:	The queue manager command level
Identifier:	MQIA_COMMAND_LEVEL
Datatype:	MQCFIN
Returned:	Always

ObjectCount

Description:	The number of queue objects accessed in the interval for which statistics data has been recorded. This value is set to the number of QStatisticsData PCF groups contained in the message.
Identifier:	MQIAMO_OBJECT_COUNT
Datatype:	MQCFIN

Returned:	Always

QStatisticsData

Description:	Grouped parameters specifying statistics details for a queue
Identifier:	MQGACF_Q_STATISTICS_DATA
Datatype:	MQCFGR
Parameters in group:	*QName* *CreateDate* *CreateTime* *QType* *QDefinitionType* *QMinDepth* *QMaxDepth* *AvgTimeOnQ* *PutCount* *PutFailCount* *Put1Count* *Put1FailCount* *PutBytes* *GetCount* *GetFailCount* *GetBytes* *BrowseCount* *BrowseFailCount* *BrowseBytes* *ExpiredMsgCount* *NonQueuedMsgCount*
Returned:	Always

QName

Description:	The name of the queue
Identifier:	MQCA_Q_NAME
Datatype:	MQCFST
Maximum length:	MQ_Q_NAME_LENGTH
Returned:	Always

CreateDate

Description:	The date when the queue was created
Identifier:	MQCA_CREATION_DATE
Datatype:	MQCFST
Maximum length:	MQ_DATE_LENGTH
Returned:	Always

CreateTime

Description:	The time when the queue was created
Identifier:	MQCA_CREATION_TIME
Datatype:	MQCFST

Maximum length:	MQ_TIME_LENGTH
Returned:	Always

QType

Description:	The type of the queue
Identifier:	MQIA_Q_TYPE
Datatype:	MQCFIN
Value:	MQOT_LOCAL
Returned:	Always

QDefinitionType

Description:	The queue definition type
Identifier:	MQIA_DEFINITION_TYPE
Datatype:	MQCFIN
Values:	Possible values are • MQQDT_PREDEFINED • MQQDT_PERMANENT_DYNAMIC • MQQDT_TEMPORARY_DYNAMIC
Returned:	When available

QMinDepth

Description:	The minimum queue depth during the monitoring period
Identifier:	MQIAMO_Q_MIN_DEPTH
Datatype:	MQCFIN
Included in PCF group:	*QStatisticsData*
Returned:	When available

QMaxDepth

Description:	The maximum queue depth during the monitoring period
Identifier:	MQIAMO_Q_MAX_DEPTH
Datatype:	MQCFIN
Included in PCF group:	*QStatisticsData*
Returned:	When available

AvgTimeOnQ

Description:	The average latency, in microseconds, of messages destructively retrieved from the queue during the monitoring period. This parameter is an integer list indexed by persistence value, see Note 2.
Identifier:	MQIAMO_AVG_Q_TIME
Datatype:	MQCFIL64
Included in PCF group:	*QStatisticsData*
Returned:	When available

PutCount

Description:	The number of persistent and nonpersistent messages successfully put to the queue, with exception of MQPUT1 requests. This parameter is an integer list indexed by persistence value. See Note 2.
Identifier:	MQIAMO_PUTS
Datatype:	MQCFIL
Included in PCF group:	*QStatisticsData*
Returned:	When available

PutFailCount

Description:	The number of unsuccessful attempts to put a message to the queue
Identifier:	MQIAMO_PUTS_FAILED
Datatype:	MQCFIN
Included in PCF group:	*QStatisticsData*
Returned:	When available

Put1Count

Description:	The number of persistent and nonpersistent messages successfully put to the queue using MQPUT1 calls. This parameter is an integer list indexed by persistence value. See Note 2.
Identifier:	MQIAMO_PUT1S
Datatype:	MQCFIL
Included in PCF group:	*QStatisticsData*
Returned:	When available

Put1FailCount

Description:	The number of unsuccessful attempts to put a message using MQPUT1 calls
Identifier:	MQIAMO_PUT1S_FAILED
Datatype:	MQCFIN
Included in PCF group:	*QStatisticsData*
Returned:	When available

PutBytes

Description:	The number of bytes written in put requests to the queue
Identifier:	MQIAMO64_PUT_BYTES
Datatype:	MQCFIL64
Included in PCF group:	*QStatisticsData*
Returned:	When available

GetCount

Description:	The number of successful destructive get requests for persistent and nonpersistent messages.This parameter is an integer list indexed by persistence value. See Note 2.
Identifier:	MQIAMO_GETS

Datatype:	MQCFIL
Included in PCF group:	*QStatisticsData*
Returned:	When available

GetFailCount

Description:	The number of unsuccessful destructive get requests
Identifier:	MQIAMO_GETS_FAILED
Datatype:	MQCFIN
Included in PCF group:	*QStatisticsData*
Returned:	When available

GetBytes

Description:	The number of bytes read in destructive put requests for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value. See Note 2.
Identifier:	MQIAMO64_GET_BYTES
Datatype:	MQCFIL64
Included in PCF group:	*QStatisticsData*
Returned:	When available

BrowseCount

Description:	The number of successful non-destructive get requests for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value. See Note 2.
Identifier:	MQIAMO_BROWSES
Datatype:	MQCFIL
Included in PCF group:	*QStatisticsData*
Returned:	When available

BrowseFailCount

Description:	The number of unsuccessful non-destructive get requests
Identifier:	MQIAMO_BROWSES_FAILED
Datatype:	MQCFIN
Included in PCF group:	*QStatisticsData*
Returned:	When available

BrowseBytes

Description:	The number of bytes read in non-destructive get requests for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value. See Note 2.
Identifier:	MQIAMO64_BROWSE_BYTES
Datatype:	MQCFIL64
Included in PCF group:	*QStatisticsData*
Returned:	When available

ExpiredMsgCount

Description:	The number of persistent and nonpersistent messages that were discarded because they had expired before they could be retrieved. This parameter is an integer list indexed by persistence value. See Note 2.
Identifier:	MQIAMO_MSGS_EXPIRED
Datatype:	MQCFIL
Included in PCF group:	*QStatisticsData*
Returned:	When available

NonQueuedMsgCount

Description:	The number of messages that bypassed the queue and were transferred directly to a waiting application. Bypassing a queue can only occur in certain circumstances. This number represents how many times WebSphere MQ was able to bypass the queue, and not the number of times an application was waiting.
Identifier:	MQIAMO_MSGS_NOT_QUEUED
Datatype:	MQCFIL
Included in PCF group:	*QStatisticsData*
Returned:	When available

Channel statistics message data

Message name:	Channel statistics message.
Platforms:	All, except WebSphere MQ for z/OS.
System queue:	SYSTEM.ADMIN.STATISTICS.QUEUE.

Statistics message data

QueueManager

Description:	The name of the queue manager.
Identifier:	MQCA_Q_MGR_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH.
Returned:	Always.

IntervalStartDate

Description:	The date at the start of the monitoring period.
Identifier:	MQCAMO_START_DATE.
Datatype:	MQCFST.
Maximum length:	MQ_DATE_LENGTH.
Returned:	Always.

IntervalStartTime

Description:	The time at the start of the monitoring period.
Identifier:	MQCAMO_START_TIME.
Datatype:	MQCFST.
Maximum length:	MQ_TIME_LENGTH.

Returned:	Always.

IntervalEndDate

Description:	The date at the end of the monitoring period
Identifier:	MQCAMO_END_DATE.
Datatype:	MQCFST.
Maximum length:	MQ_DATE_LENGTH.
Returned:	Always.

IntervalEndTime

Description:	The time at the end of the monitoring period
Identifier:	MQCAMO_END_TIME.
Datatype:	MQCFST.
Maximum length:	MQ_TIME_LENGTH
Returned:	Always.

CommandLevel

Description:	The queue manager command level.
Identifier:	MQIA_COMMAND_LEVEL.
Datatype:	MQCFIN.
Returned:	Always.

ObjectCount

Description:	The number of Channel objects accessed in the interval for which statistics data has been recorded. This value is set to the number of ChlStatisticsData PCF groups contained in the message.
Identifier:	MQIAMO_OBJECT_COUNT
Datatype:	MQCFIN.
Returned:	Always.

ChlStatisticsData

Description:	Grouped parameters specifying statistics details for a channel.
Identifier:	MQGACF_CHL_STATISTICS_DATA.
Datatype:	MQCFGR.

Parameters in group:	*ChannelName* *ChannelType* *RemoteQmgr* *ConnectionName* *MsgCount* *TotalBytes* *NetTimeMin* *NetTimeAvg* *NetTimeMax* *ExitTimeMin* *ExitTimeAvg* *ExitTimeMax* *FullBatchCount* *IncmplBatchCount* *AverageBatchSize* *PutRetryCount*
Returned:	Always.

ChannelName

Description:	The name of the channel.
Identifier:	MQCACH_CHANNEL_NAME.
Datatype:	MQCFST.
Maximum length:	MQ_CHANNEL_NAME_LENGTH.
Returned:	Always.

ChannelType

Description:	The channel type.
Identifier:	MQIACH_CHANNEL_TYPE.
Datatype:	MQCFIN.
Values:	Possible values are: **MQCHT_SENDER** Sender channel. **MQCHT_SERVER** Server channel. **MQCHT_RECEIVER** Receiver channel. **MQCHT_REQUESTER** Requester channel. **MQCHT_CLUSRCVR** Cluster receiver channel. **MQCHT_CLUSSDR** Cluster sender channel.
Returned:	Always.

RemoteQmgr

Description:	The name of the remote queue manager.
Identifier:	MQCA_REMOTE_Q_MGR_NAME.

Datatype:	MQCFST.
Maximum length:	MQ_Q_MGR_NAME_LENGTH
Returned:	When available.

ConnectionName

Description:	Connection name of remote queue manager.
Identifier:	MQCACH_CONNECTION_NAME.
Datatype:	MQCFST
Maximum length:	MQ_CONN_NAME_LENGTH
Returned:	When available.

MsgCount

Description:	The number of persistent and nonpersistent messages sent or received.
Identifier:	MQIAMO_MSGS.
Datatype:	MQCFIN
Returned:	When available.

TotalBytes

Description:	The number of bytes sent or received for persistent and nonpersistent messages. This parameter is an integer list indexed by persistence value, see Note 2
Identifier:	MQIAMO64_BYTES.
Datatype:	MQCFIL64.
Returned:	When available.

NetTimeMin

Description:	The shortest recorded channel round trip measured in the recording interval, in microseconds.
Identifier:	MQIAMO_NET_TIME_MIN.
Datatype:	MQCFIN.
Returned:	When available.

NetTimeAvg

Description:	The average recorded channel round trip measured in the recording interval, in microseconds.
Identifier:	MQIAMO_NET_TIME_AVG.
Datatype:	MQCFIN.
Returned:	When available.

NetTimeMax

Description:	The longest recorded channel round trip measured in the recording interval, in microseconds.
Identifier:	MQIAMO_NET_TIME_MAX.
Datatype:	MQCFIN.
Returned:	When available.

ExitTimeMin

Description:	The shortest recorded time, in microseconds, spent executing a user exit in the recording interval,
Identifier:	MQIAMO_EXIT_TIME_MIN.
Datatype:	MQCFIN.
Returned:	When available.

ExitTimeAvg

Description:	The average recorded time, in microseconds, spent executing a user exit in the recording interval. Measured in microseconds.
Identifier:	MQIAMO_EXIT_TIME_AVG.
Datatype:	MQCFIN.
Returned:	When available.

ExitTimeMax

Description:	The longest recorded time, in microseconds, spent executing a user exit in the recording interval. Measured in microseconds.
Identifier:	MQIAMO_EXIT_TIME_MAX.
Datatype:	MQCFIN.
Returned:	When available.

FullBatchCount

Description:	The number of batches processed by the channel, that were sent because the value of the channel attribute BATCHSZ was reached.
Identifier:	MQIAMO_FULL_BATCHES.
Datatype:	MQCFIN.
Returned:	When available.

IncmplBatchCount

Description:	The number of batches processed by the channel, that were sent without the value of the channel attribute BATCHSZ being reached.
Identifier:	MQIAMO_INCOMPLETE_BATCHES.
Datatype:	MQCFIN.
Returned:	When available.

AverageBatchSize

Description:	The average batch size of batches processed by the channel.
Identifier:	MQIAMO_AVG_BATCHE_SIZE.
Datatype:	MQCFIN.
Returned:	When available.

PutRetryCount

Description:	The number of times in the time interval that a message failed to be put, and entered a retry loop.
Identifier:	MQIAMO_PUT_RETRIES.
Datatype:	MQCFIN.
Returned:	When available.

Reference notes

These notes are applicable to the following sections:

- "MQI accounting message data" on page 264
- "Queue accounting message data" on page 270
- "MQI statistics message data" on page 279
- "Queue statistics message data" on page 284
- "Channel statistics message data" on page 290

1. This parameter relates to WebSphere MQ objects. This parameter is an array of values (MQCFIL or MQCFIL64) indexed by the following constants:

Table 22. Array indexed by object type

Object type	**Value context**
MQOT_Q (1)	Contains the value relating to queue objects.
MQOT_NAMELIST (2)	Contains the value relating to namelist objects.
MQOT_PROCESS (3)	Contains the value relating to process objects.
MQOT_Q_MGR (5)	Contains the value relating to queue manager objects.
MQOT_CHANNEL (6)	Contains the value relating to channel objects.
MQOT_AUTH_INFO (7)	Contains the value relating to authentication information objects.
MQOT_TOPIC (8)	Contains the value relating to topic objects.

2. This parameter relates to WebSphere MQ messages. This parameter is an array of values (MQCFIL or MQCFIL64) indexed by the following constants:

Table 23. Array indexed by persistence value

Constant	**Value**
1	Contains the value for nonpersistent messages.
2	Contains the value for persistent messages.

Note: The index for each of these arrays starts at zero, so an index of 1 refers to the second row of the array. Elements of these arrays not listed in these tables contain no accounting or statistics information.

Chapter 5. Real-time monitoring

An introduction to real-time monitoring

Real-time monitoring is a technique that allows you to determine the current state of queues and channels within a queue manager. A number of commands are available that, when issued, return real-time information about queues and channels.

Real-time monitoring

Real-time monitoring is a technique that allows you to determine the current state of queues and channels within a queue manager. A number of commands are available that when issued return real-time information about queues and channels. Varying amounts of information can be returned for a single queue or channel, or for multiple queues or channels. The information returned is accurate at the moment the command was issued. Real-time monitoring can be used to:

- Help system administrators understand the steady state of their WebSphere MQ system. This helps with problem diagnosis if a problem occurs in the system.
- Determine the condition of your queue manager at any moment, even if no specific event or problem has been detected.
- Assist in determining the cause of a problem in your system.

When using real-time monitoring, information can be returned for either queues or channels. The amount of real-time information returned is controlled by queue manager, queue, and channel attributes. This chapter discusses how to enable these attributes, and how to display real-time information, as follows:

- "Controlling real-time monitoring"
- "Displaying queue and channel monitoring data" on page 299

Real-time monitoring for queues and channels is in addition to, and separate from, performance and channel event monitoring, see "An introduction to instrumentation events" on page 5 and "Understanding performance events" on page 19.

"Monitoring queues" on page 300 outlines a number of questions that you can ask about a queue to ensure it is being serviced properly, and the commands you can use for this purpose. Some of the fields used are among these monitoring fields and must be enabled to allow their use.

"Monitoring channels" on page 303 outlines a number of questions that you can ask about a channel to ensure it is running properly, and the commands you can use for this purpose. Some of the fields used are among these monitoring fields and must be enabled to allow their use.

Controlling real-time monitoring

There are a number of queue and channel status attributes that will hold monitoring information if real-time monitoring is enabled. If real-time monitoring is not enabled, then no monitoring information will be held in the monitoring attributes.

Real-time monitoring can be enabled or disabled for individual queues or channels, or for multiple queue or channels. To control individual queues or channels, the queue attribute MONQ, or the channel attribute MONCHL, must be set to enable or disable real-time monitoring. To control many queues or channels together, real-time monitoring can be enabled or disabled at the queue manager level using the queue manager attributes MONQ and MONCHL. For all queue and channel objects whose monitoring attribute is specified with the default value, QMGR, real-time monitoring is controlled at the queue manager level.

Automatically defined cluster-sender channels are not WebSphere MQ objects, so do not have attributes in the same way as channel objects. To control automatically defined cluster-sender channels, use the queue manager attribute, MONACLS. This attribute determines whether automatically defined cluster-sender channels within a queue manager are enabled or disabled for channel monitoring.

Real-time monitoring of channels (MONCHL) can be set to one of the three monitoring levels, low, medium or high. This level is either set at the object level, or at the queue manager level. The choice of which level to use is dependant on your system. Collecting monitoring data may require the execution of some relatively expensive instructions (for example, obtaining system time), so in order to reduce the impact of real-time monitoring, the medium and low monitoring options measure a sample of the data at regular intervals rather than collecting data all the time. Table 24 summarizes the monitoring levels available with real-time monitoring of channels:

Table 24. Monitoring levels

Level	Description	Usage
Low	Measure a small sample of the data, at regular intervals.	For objects that process a high volume of messages.
Medium	Measure a sample of the data, at regular intervals.	For most objects.
High	Measure all data, at regular intervals.	For objects that process only a few messages per second, on which the most current information is important.

Real-time monitoring of queues (MONQ) can also be set to one of the three monitoring levels, low, medium or high. However, there is no distinction between these values. The values all turn data collection on, but do not affect the size of the sample.

Examples of controlling real-time monitoring

A number of examples follow that show how to set the necessary queue, channel, and queue manager attributes to control the level of monitoring. In all the examples, when monitoring is enabled, queue and channel objects have a medium level of monitoring.

1. To enable both queue and channel monitoring for all queues and channels at the queue manager level, use the following commands:
```
ALTER QMGR MONQ(MEDIUM) MONCHL(MEDIUM)
ALTER QL(Q1) MONQ(QMGR)
ALTER CHL(QM1.TO.QM2) CHLTYPE(SDR) MONCHL(QMGR)
```
2. To enable monitoring for all queues and channels, with the exception of local queue, Q1, and sender channel, QM1.TO.QM2, use the following commands:
```
ALTER QMGR MONQ(MEDIUM) MONCHL(MEDIUM)
ALTER QL(Q1) MONQ(OFF)
ALTER CHL(QM1.TO.QM2) CHLTYPE(SDR) MONCHL(OFF)
```

3. To disable both queue and channel monitoring for all queues and channels, with the exception of local queue, `Q1`, and sender channel, `QM1.TO.QM2`, use the following commands:

   ```
   ALTER QMGR MONQ(OFF) MONCHL(OFF)
   ALTER QL(Q1) MONQ(MEDIUM)
   ALTER CHL(QM1.TO.QM2) CHLTYPE(SDR) MONCHL(MEDIUM)
   ```
4. To disable both queue and channel monitoring for all queues and channels, regardless of individual object attributes, use the following command:

   ```
   ALTER QMGR MONQ(NONE) MONCHL(NONE)
   ```

To control the monitoring capabilities of automatically defined cluster-sender channels use the following command:

```
ALTER QMGR MONACLS(MEDIUM)
```

To specify that automatically defined cluster-sender channels are to use the queue manager setting for channel monitoring, use the following command:

```
ALTER QMGR MONACLS(QMGR)
```

Displaying queue and channel monitoring data

To display real-time monitoring information for a queue either use the WebSphere MQ Explorer, or the MQSC command `DISPLAY QSTATUS` specifying the optional parameter `MONITOR`.

To display real-time monitoring information for a channel either use the WebSphere MQ Explorer, or the MQSC command `DISPLAY CHSTATUS` specifying the optional parameter `MONITOR`.

For examples of using the MQSC commands `DISPLAY QSTATUS` and `DISPLAY CHSTATUS`, see “Examples of displaying monitoring levels.”

Examples of displaying monitoring levels

The queue, `Q1`, has the attribute MONQ set as the default value, QMGR, and the queue manager where it resides has the attribute MONQ set as MEDIUM. To display the monitoring fields collected for this queue, use the following command:

```
DISPLAY QSTATUS(Q1) MONITOR
```

This displays the monitoring fields, and monitoring level, of queue, `Q1`:

```
QSTATUS(Q1)
TYPE(QUEUE)
MONQ(MEDIUM)
QTIME(11892157,24052785)
MSGAGE(37)
LPUTDATE(2005-03-02)
LPUTTIME(09.52.13)
LGETDATE(2005-03-02)
LGETTIME(09.51.02)
```

For the meaning of these fields see the WebSphere MQ Script (MQSC) Command Reference manual.

The sender channel, `QM1.TO.QM2`, has the attribute MONCHL set as the default value, QMGR, and the queue manager where it resides has the attribute MONCHL set as MEDIUM. To display the monitoring fields collected for this sender channel, use the following command:

```
DISPLAY CHSTATUS(QM1.TO.QM2) MONITOR
```

This displays the monitoring fields, and monitoring level, of sender channel, `QM1.TO.QM2`:

```
CHSTATUS(QM1.TO.QM2)
XMITQ(Q1)
CONNAME(127.0.0.1)
CURRENT
CHLTYPE(SDR)
STATUS(RUNNING)
SUBSTATE(MQGET)
MONCHL(MEDIUM)
XQTIME(755394737,755199260)
NETTIME(13372,13372)
EXITTIME(0,0)
XBATCHSZ(50,50)
COMPTIME(0,0)
STOPREQ(NO)
RQMNAME(QM2)
```

Monitoring Indicator Values

Some monitoring fields display a pair of values separated by a comma. These pairs are short term and long term indicators for the time measured since monitoring was enabled for the object, or from when the queue manager was started:

- The short term indicator is the first value in the pair and is calculated in a way such that more recent measurements are given a higher weighting and will have a greater effect on this value. This gives an indication of recent trend in measurements taken.
- The long term indicator in the second value in the pair and is calculated in a way such that more recent measurements are not given such a high weighting. This gives an indication of the longer term activity on performance of a resource.

These indicator values are most useful to detect changes in the operation of your queue manager. This requires knowledge of the times these indicators show when in normal use, in order to detect increases in these times. By collecting and checking these values regularly you can detect fluctuations in the operation of your queue manager. This can indicate a change in performance.

Monitoring queues

Very often the first sign of a problem with a queue that is being serviced is that the number of messages on the queue (CURDEPTH) is increasing. This section discusses the various monitoring options that are available to determine the problem with a queue and the application servicing it.

Of course, the increasing number of messages might not be a sign of a problem if it is expected at certain times of day or under certain workloads. However, if there is no known explanation for the increasing number of messages, it should be investigated.

You might have an application queue where there is a problem with the application, or a transmission queue where there is a problem with the channel.

"Monitoring channels" on page 303 discusses additional monitoring options that are available when the application that services the queue is a channel.

This section discusses the questions that you should ask, and the fields that you should look at in the output of various commands, when investigating problems with a particular queue. In all the examples, the queue being examined is called Q1.

Does your application have the queue open?

To determine whether your application has the queue open, do the following:

1. Ensure that the application that is running against the queue is the application that you expect. Issue the following command for the queue in question:
   ```
   DISPLAY QSTATUS(Q1) TYPE(HANDLE) ALL
   ```
   In the output, look at the APPLTAG field, and check that the name of your application is shown. If the name of your application is not shown, or if there is no output at all, start your application.
2. If the queue is a transmission queue, look in the output at the CHANNEL field. If the channel name is not shown in the CHANNEL field, then see "Is the channel running?" on page 303
3. Ensure that the application that is running against the queue has the queue open for input. Issue the following command:
   ```
   DISPLAY QSTATUS(Q1) TYPE(QUEUE) ALL
   ```
   In the output, look at the IPPROCS field to see if any application has the queue open for input. If the value is 0 and this is a user application queue, make sure that the application opens the queue for input to get the messages off the queue.

Are the messages on the queue available?

If there is a large number of messages on the queue and your application is not processing any of them, follow this procedure.

Perform the following steps to investigate why your application is not processing messages from the queue:

1. Ensure that your application is not asking for a specific message ID or correlation ID when it should be processing all the messages on the queue.
2. Although the current depth of the queue might show that there is an increasing number of messages on the queue, some messages on the queue might not be available to be got by an application, because they are not committed; the current depth includes the number of uncommitted MQPUTs of messages to the queue. Issue the following command:
   ```
   DISPLAY QSTATUS(Q1) TYPE(QUEUE) ALL
   ```
   In the output, look at the UNCOM field to see whether there are any uncommitted messages on the queue.
3. If your application is attempting to get any messages from the queue, check whether the putting application is committing the messages correctly. Issue the following command to find out the names of applications that are putting messages to this queue:
   ```
   DISPLAY QSTATUS(Q1) TYPE(HANDLE) OPENTYPE(OUTPUT)
   ```
4. Then issue the following command, inserting in <appltag> the APPLTAG value from the output of the previous command:
   ```
   DISPLAY CONN(*) WHERE(APPLTAG EQ <appltag>) UOWSTDA UOWSTTI
   ```

This shows when the unit of work was started and will help you discover whether the application is creating a long running unit of work. If the putting application is a channel, see "Does a batch take a long time to complete?" on page 306

Is your application getting messages off the queue?

To check whether your application is getting messages off the queue, do the following:

1. Ensure that the application that is running against the queue is actually processing messages from the queue. Issue the following command:
   ```
   DISPLAY QSTATUS(Q1) TYPE(QUEUE) ALL
   ```
   In the output, look at the LGETDATE and LGETTIME fields which show when the last get was done from the queue.
2. If the last get that was done on this queue was longer ago than expected, ensure that the application is processing correctly. If the application is a channel, see "Is the channel moving messages?" on page 305 for more details.

Can the application process messages fast enough?

If messages are building up on the queue, and there are none of the processing problems that are described in the previous sections, it might simply be that the application cannot process messages fast enough. If the application is a channel, see "Can the channel process messages fast enough?" on page 307.

To determine whether messages are being processed fast enough, do the following:

1. Issue the following command periodically to gather performance data about the queue:
   ```
   DISPLAY QSTATUS(Q1) TYPE(QUEUE) ALL
   ```
   If the values in the QTIME indicators are high, or are increasing over this period, and you have already ruled out the possibility of long running Units of Work as discussed in "Are the messages on the queue available?" on page 301, this can indicate that the getting application is not keeping up with the putting applications.
2. If your getting application cannot keep up with the putting applications, consider adding another getting application to process the queue. Whether you can do this depends on the design of the application and whether the queue can be shared by more than one application. Features such as message grouping or getting by correlation ID, might help to ensure that two applications can process a queue simultaneously. See WebSphere MQ Application Programming Guide for more details.

What about when the current depth is not increasing?

If the current depth of your queue is not increasing, it might still be useful to monitor the queue to check whether your application is processing messages correctly. to do this, issue the following command periodically to gather performance data about the queue:

```
DISPLAY QSTATUS(Q1) TYPE(QUEUE) MSGAGE QTIME
```

In the output, if the value in MSGAGE increases over the period of time, and your application is designed to process all messages, this might indicate that some messages are not being processed at all.

Monitoring channels

Very often the first sign of a problem with a queue that is being serviced is that the current depth of the queue is increasing. Of course, the fact that the current depth of the queue is increasing might not be a sign of a problem if it is expected at certain times of day or under certain workloads. However, if there is no known explanation for the current depth increasing, it is worth investigating.

This section discusses the various channel monitoring options that are available to help determine the problem with a transmission queue and the channel servicing it. In all the examples, the transmission queue in question is called QM2 and the channel is called QM1.TO.QM2. QM1.TO.QM2 is used to send messages from queue manager, QM1, to queue manager, QM2. The channel definition at queue manager QM1 is either a sender or server channel, and the channel definition at queue manager, QM2, is either a receiver or requester channel. Cluster channels are discussed in "Cluster channels" on page 308.

Is the channel running?

To determine whether a channel is running and processing this transmission queue, check the status of the channel by doing the following:

1. Issue the following command to find out which channel is supposed to be processing the transmission queue QM2:
   ```
   DIS CHANNEL(*) WHERE(XMITQ EQ QM2)
   ```
   In this example, the output of this command shows that the channel servicing the transmission queue is QM1.TO.QM2.
2. Issue the following command to determine the status of the channel, QM1.TO.QM2:
   ```
   DIS CHSTATUS(QM1.TO.QM2) ALL
   ```
3. Look at the STATUS field of the output.
   - If the value of the STATUS field is `RUNNING`, see "Is the channel moving messages?" on page 305.
   - If the value of the STATUS field is not `RUNNING`, see one of the topics in this section of the documentation.

If the channel has stopped

Having issued the command `DIS CHSTATUS(QM1.TO.QM2) ALL`, if the output shows the status of the channel is `STOPPED`, then the channel could either have stopped because of an error or because of a command. Do the following:

1. Determine why the channel stopped by checking the error logs. If the channel stopped due to an error, the error must be fixed.

   In addition, ensure that the channel has values specified for the retry attributes, *SHORTRTY*, and *LONGRTY*, so that in the event of transient failures such as network errors, the channel will attempt to restart automatically. For more information on channel retry, see the WebSphere MQ Intercommunication manual.

   The channel must be manually started for the changes to take effect.

2. Issue the following command to start the channel again:
   ```
   START CHANNEL(QM1.TO.QM2)
   ```

On WebSphere MQ for z/OS, you can detect when a user stops a channel by using command event messages. For more information, see "Understanding command events" on page 40.

If the channel is inactive

Having issued the command `DIS CHSTATUS(QM1.TO.QM2) ALL`, if the output shows no status, then the channel is inactive. Do the following:

1. If the channel should have been automatically started by a trigger, check the following:
   - Check that the messages on the transmission queue are committed. For more information, see "Are the messages on the queue available?" on page 301.
   - If there are available messages on the transmission queue, check that the trigger settings on the transmission queue are correct. For details of how to set up triggering on a transmission queue to start a channel, see the WebSphere MQ Intercommunication manual.
2. For now, the channel must be manually started. Issue the following command:
   ```
   START CHANNEL(QM1.TO.QM2)
   ```

If the channel is in retry state

Having issued the command `DIS CHSTATUS(QM1.TO.QM2) ALL`, if the output shows the status of the channel is `RETRY`, then the channel has detected an error. Do the following:

1. Identify, and fix, the error by checking the error logs.
2. Start the channel using one of the following:
   - Start the channel manually, by issuing the following command:
     ```
     START CHANNEL(QM1.TO.QM2)
     ```
   - Wait for the channel to connect successfully on its next retry,

If the channel in another state

Having issued the command `DIS CHSTATUS(QM1.TO.QM2) ALL`, if the output shows the status of the channel is `BINDING` or `REQUESTING`, then it has not yet successfully connected to the partner. Do the following:

1. Issue the following command, at both ends of the channel, to determine the substate of the channel:
   ```
   DIS CHSTATUS(QM1.TO.QM2) ALL
   ```

   Note:
   a. In some cases there might be a substate at one end of the channel only.
   b. Many substates are transitory, so issue the command a few times to detect whether a channel is stuck in a particular substate.
2. Check Table 25 on page 305 to determine what to do next:

Table 25. Substates seen with status binding or requesting

Initiating MCA substate [1]	Responding MCA substate [2]	Notes
NAMESERVER		The initiating MCA is waiting for a nameserver request to complete. Ensure that the correct hostname has been specified in the channel attribute, CONNAME, and that your name servers are set up correctly.
SCYEXIT	SCYEXIT	The MCAs are currently *in conversation* through a security exit. For more information, see "Are there exits processing?" on page 307.
	CHADEXIT	The channel autodefinition exit is currently executing. For more information, see "Are there exits processing?" on page 307.
RCVEXIT SENDEXIT MSGEXIT MREXIT	RCVEXIT SENDEXIT MSGEXIT MREXIT	Exits are called at channel startup for MQXR_INIT. Review the processing in this part of your exit if this takes a long time. For more information, see "Are there exits processing?" on page 307.
SERIALIZE	SERIALIZE	This substate only applies to channels with a disposition of SHARED.
NETCONNECT		This substate is shown if there is a delay in connecting due to incorrect network configuration.
SSLHANDSHAKE	SSLHANDSHAKE	An SSL handshake is made up of a number of sends and receives. If network times are slow, or connection to lookup CRLs are slow, this affects the time taken to do the handshake. On WebSphere MQ for z/OS this substate can also be indicative of not having enough SSLTASKS.

Note:

1. The initiating MCA is the end of the channel which started the conversation. This can be senders, cluster-senders, fully-qualified servers and requesters. In a server-requester pair, it is the end from which you started the channel.
2. The responding MCA is the end of the channel which responded to the request to start the conversation. This can be receivers, cluster-receivers, requesters (when the server or sender is started), servers (when the requester is started) and senders (in a requester-sender call-back pair of channels).

Is the channel moving messages?

Having issued the command `DIS CHSTATUS(QM1.TO.QM2) ALL`, if the output shows the status of the channel is `RUNNING`, then it has successfully connected to the partner system.

Providing there are no uncommitted messages on the transmission queue, see "Are the messages on the queue available?" on page 301, then there are messages available for the channel to get and send. Do the following:

1. In the output from the display channel status command, `DIS CHSTATUS(QM1.TO.QM2) ALL`, look at the following fields:

   MSGS

   Number of messages sent or received (or, for server-connection channels, the number of MQI calls handled) during this session (since the channel was started).

BUFSSENT
Number of transmission buffers sent. This includes transmissions to send control information only.

BYTSSENT
Number of bytes sent during this session (since the channel was started). This includes control information sent by the message channel agent.

LSTMSGDA
Date when the last message was sent or MQI call was handled, see LSTMSGTI.

LSTMSGTI
Time when the last message was sent or MQI call was handled. For a sender or server, this is the time the last message (the last part of it if it was split) was sent. For a requester or receiver, it is the time the last message was put to its target queue. For a server-connection channel, it is the time when the last MQI call completed.

CURMSGS
For a sending channel, this is the number of messages that have been sent in the current batch. For a receiving channel, it is the number of messages that have been received in the current batch. The value is reset to zero, for both sending and receiving channels, when the batch is committed.

Determine whether the channel has sent any messages since it started. If any have been sent, determine when the last message was sent.

2. If the channel has started a batch, which has not yet completed (indicated by a non-zero value in CURMSGS), it could be waiting for the other end of the channel to acknowledge the batch. Look at the SUBSTATE field in the output and refer to Table 26:

Table 26. Sender and receiver MCA substates

Sender SUBSTATE	Receiver SUBSTATE	Notes
MQGET	RECEIVE	Normal states of a channel at rest.
SEND	RECEIVE	SEND is usually a transitory state. If SEND is seen it indicates that the communication protocol buffers have filled. This can indicate a network problem.
RECEIVE		If the sender is seen in RECEIVE substate for any length of time, it is waiting on a response, either to a batch completion or a heartbeat. See “Does a batch take a long time to complete?” for more details.
Note: Any substates not mentioned here are discussed in “Can the channel process messages fast enough?” on page 307.		

Does a batch take a long time to complete?

There could be a number of different reasons why a batch can take a long time to complete. Once a sender channel has sent a batch of messages it waits for confirmation of that batch from the receiver (the exception to this is pipelined channels, see the WebSphere MQ Intercommunication manual). Various things can affect how long the sender waits for.

Is the network slow?

If the network is slow, this can affect the time it takes to complete a batch. The measurements, described in "Is the network slow?" on page 308, that result in the indicators for the NETTIME field are measured at the end of a batch. The first batch affected by a slowdown in the network is not be indicated with a change in the NETTIME value because it is measured at the end of the batch.

Is the channel using message retry?

If the receiver channel fails to put a message to a target queue it may use message retry processing, rather than put the message to a dead-letter immediately. This can cause the batch to slow down. In between MQPUT attempts, the channel will have STATUS(PAUSED) indicating that it is waiting for the message retry interval to pass.

Can the channel process messages fast enough?

If there are messages building up on the transmission queue, yet you have found no processing problems, it can simply be that the channel cannot process messages fast enough. To identify if this is the case, do the following:

1. Issue the following command repeatedly over a period of time to gather performance data about the channel:

   ```
   DIS CHSTATUS(QM1.TO.QM2) ALL
   ```
2. Providing you have confirmed that there are no uncommitted messages on the transmission queue (detailed in "Are the messages on the queue available?" on page 301), check the XQTIME field. When the value of the XQTIME indicators are consistently high, or are increasing over this period, this indicates that the channel is not keeping pace with the putting applications. In this situation, try to determine why the channel is not keeping pace by asking the questions handled in the topics in this section of the documentation.

Are there exits processing?

If exits are used on the channel that is delivering these messages, they may add to the time spent processing messages. To identify if this is the case, do the following:

1. In the output of the command `DIS CHSTATUS(QM1.TO.QM2) ALL`, check the EXITTIME field.

   If the time spent in exits is higher than expected, review the processing in your exits for any unnecessary loops or extra processing especially in message, send, and receive exits, as they will affect all messages moved across the channel.
2. In the output of the command `DIS CHSTATUS(QM1.TO.QM2) ALL`, check the SUBSTATE field.

   If you see the channel has a substate of one of the following for any appreciable time, you should also review your exits.
   - SCYEXIT
   - RCVEXIT
   - SENDEXIT
   - MSGEXIT
   - MREXIT

Is the network slow?

If messages are not moving fast enough across a channel, this could be because the network is slow. To identify if this is the case, do the following:

1. In the output of the command `DIS CHSTATUS(QM1.TO.QM2) ALL`, check the NETTIME field.

   These indicators are measured when the sending channel asks its partner for a response. This happens at the end of each batch and, when a channel is idle during heartbeating.
2. If this indicator shows that round trips are taking longer than expected, use other network monitoring tools to investigate the performance of your network.

Is the channel using compression?

If the channel is using compression, this will add to the time spent processing messages.

If only one compression algorithm is being used, do the following:

1. In the output of the command `DIS CHSTATUS(QM1.TO.QM2) ALL`, check the COMPTIME field.

   These indicators show the time spent during compression or decompression.
2. If the chosen compression is not reducing the amount of data to send by the expected amount, then change the compression algorithm.

If multiple compression algorithms are being used, do the following:

1. In the output of the command `DIS CHSTATUS(QM1.TO.QM2) ALL`, check the COMPTIME, COMPHDR, and COMPMSG fields.
2. Change the compression algorithms specified on the channel definition, or consider writing a message exit to override the channel's choice of compression algorithm for particular messages if the rate of compression, or choice of algorithm, is not providing the required compression or performance.

Cluster channels

The advice in this chapter is also applicable to cluster channels. If the queue that is showing signs of a build up of messages is the SYSTEM.CLUSTER.TRANSMIT.QUEUE, the first step in diagnosing the problem is discovering which channel, or channels, are having a problem delivering messages. To do this, do the following:

1. Issue the following command:

   ```
   DIS CHSTATUS(*) WHERE(XQMSGSA GT 1)
   ```

   Note: If you have a busy cluster that has many messages moving, consider issuing this command with a higher number to eliminate the channels that have only a few messages available to deliver.
2. Look through the output for the channel, or channels, that have large values in the field XQMSGSA. Apply the methods outlined in this chapter to determine why the channel is not moving messages, or is not moving them fast enough.

The Windows performance monitor

Administrators of WebSphere MQ for Windows can monitor the performance of local queues using the Windows performance monitor.

The performance monitor displays an object type called `MQSeries Queues` in which performance data for local queues is stored.

Active local queues defined in running queue managers are displayed as `QueueName:QMName` in the performance monitor instance list when you select the `MQSeries Queues` object type. `QMName` denotes the name of the queue manager owning the queue, and `QueueName` denotes the name of the local queue.

For each queue, you can view information relating to the following:
- The current queue depth
- The queue depth as a percentage of the maximum queue depth
- The number of messages being placed on the queue each second
- The number of messages being removed from the queue each second

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

In the case of segmented messages, the performance monitor counts the appropriate number of segments.

Performance data is obtained from statistical data maintained by the WebSphere MQ queue managers for each local queue. However, performance data is available only for queues that are accessed ***after*** the performance monitor has started.

You can monitor the performance of queues on computers other than that on which the performance monitor is running, by selecting your target computer from the performance monitor, which works using the Windows Network Neighborhood hierarchy.

Chapter 6. Structure datatypes

In this appendix, the structures MQCFBS, MQCFGR, MQCFH, MQCFIL, MQCFIL64, MQCFIN, MQCFIN64, MQCFSL, MQCFST, and MQEPH are described in a language-independent form. The declarations are shown in the following programming languages:

- C
- COBOL
- PL/I
- RPG (ILE) (i5/OS only)
- S/390® assembler (z/OS only)
- Visual Basic (Windows platforms only)

MQCFBS - Byte string parameter

The MQCFBS structure describes a byte string parameter.

Type

Description:	This indicates that the structure is an MQCFBS structure describing a byte string parameter.
Datatype:	MQLONG.
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).
Datatype:	MQLONG.

Parameter

Description:	This identifies the parameter whose value is contained in the structure.
Datatype:	MQLONG.

StringLength

Description:	This is the length in bytes of the data in the *String* field, and is zero or greater.
Datatype:	MQLONG.

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*.

C language declaration (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 */
 } MQCFBS;
```

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 */
```

RPG/ILE language declaration (MQCFBS) (i5/OS only)

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQCFBS Structure
D*
D* Structure type
D  BSTYP                 1      4I 0 INZ(9)
D* Structure length
D  BSLEN                 5      8I 0 INZ(16)
D* Parameter identifier
D  BSPRM                 9     12I 0 INZ(0)
D* Length of string
D  BSSTL                13     16I 0 INZ(0)
D* String value -- first byte
D  BSSRA                17     17    INZ
```

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)
```

MQCFGR - Group parameter

The MQCFGR structure describes a group parameter in which the subsequent parameter structures are grouped together as a single logical unit. The number of subsequent structures that are included is given by *ParameterCount*. This structure, and the parameter structures it includes, are counted as one structure only in the *ParameterCount* parameter in the PCF header (MQCFH) and the group parameter (MQCFGR).

Type

Description:	Indicates that the structure type is MQCFGR describing which parameters are in this group.
Datatype:	MQLONG.
Value:	**MQCFT_GROUP** Structure defining a group of parameters.

StrucLength

Description:	Length in bytes of the MQCFGR structure.
Datatype:	MQLONG.
Value:	**MQCFGR_STRUC_LENGTH** Length of the command format group-parameter structure.

Parameter

Description:	This identifies the type of group parameter.
Datatype:	MQLONG.

ParameterCount

Description:	The number of parameter structures following the MQCFGR structure that are contained within the group identified by the *Parameter* field. If the group itself contains one or more groups, each group and its parameters count as one structure only.
Datatype:	MQLONG.

C language declaration (MQCFGR)

```
typedef struct tagMQCFGR {
  MQLONG  Type;            /* Structure type */
  MQLONG  StrucLength;     /* Structure length */
  MQLONG  Parameter;       /* Parameter identifier */
  MQLONG  ParameterCount;  /* Count of the grouped parameter structures */
 } MQCFGR;
```

COBOL language declaration (MQCFGR)

```
**   MQCFGR structure
  10 MQCFGR.
**    Structure type
   15 MQCFGR-TYPE           PIC S9(9) BINARY.
**    Structure length
   15 MQCFGR-STRUCLENGTH    PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFGR-PARAMETER      PIC S9(9) BINARY.
**    Count of grouped parameter structures
   15 MQCFGR-PARAMETERCOUNT PIC S9(9) BINARY.
```

PL/I language declaration (MQCFGR) (z/OS and Windows only)

```
dcl
 1 MQCFGR based,
  3 Type           fixed bin(31), /* Structure type */
  3 StrucLength    fixed bin(31), /* Structure length */
  3 Parameter      fixed bin(31), /* Parameter identifier */
  3 ParameterCount fixed bin(31), /* Count of grouped parameter structures */
```

RPG/ILE declaration (MQCFGR) (i5/OS only)

```
     D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
  D* MQCFGR Structure
     D*
     D* Structure type
     D  GRTYP                 1      4I INZ(20)
     D* Structure length
     D  GRLEN                 5      8I INZ(16)
     D* Parameter identifier
     D  GRPRM                 9     12I INZ(0)
     D* Count of grouped parameter structures
     D  GRCNT                13     16I INZ(0)
     D*
```

System/390 assembler-language declaration (MQCFGR) (z/OS only)

```
MQCFGR                           DSECT
MQCFGR_TYPE                      DS   F        Structure type
MQCFGR_STRUCLENGTH               DS   F        Structure length
MQCFGR_PARAMETER                 DS   F        Parameter identifier
MQCFGR_PARAMETERCOUNT            DS   F        Count of grouped parameter structures
MQCFGR_LENGTH                    EQU  *-MQCFGR Length of structure
                                 ORG  MQCFGR
MQCFGR_AREA                      DS   CL(MQCFGR_LENGTH)
```

Visual Basic language declaration (MQCFGR) (Windows only)

```
Type MQCFGR
  Type As Long            ' Structure type
  StrucLength As Long     ' Structure length
  Parameter As Long       ' Parameter identifier
  ParameterCount As Long  ' Count of grouped parameter structures
  End Type
```

MQCFH - PCF header

The MQCFH structure describes the information that is present at the start of the message data of a monitoring message.

Type

Description:	Structure type This indicates the content of the message.
Datatype:	MQLONG.

Values:

MQCFT_ACCOUNTING
: Message is an accounting message.

MQCFT_EVENT
: Message is reporting an event.

MQCFT_REPORT
: Message is an activity report.

MQCFT_RESPONSE
: Message is a response to a command.

MQCFT_STATISTICS
: Message is a statistics message.

MQCFT_TRACE_ROUTE
: Message is a trace-route message.

StrucLength

Description:	This is the length in bytes of the MQCFH structure
Datatype:	MQLONG.
Value:	**MQCFH_STRUC_LENGTH** Length of command format header structure.

Version

Description:	Structure version number.
Datatype:	MQLONG.
Value:	**MQCFH_VERSION_1** Version number for all events except configuration and command events.
	MQCFH_VERSION_2 Version number for configuration events.
	MQCFH_VERSION_3 Version number for command events, activity reports, trace-route messages, accounting and statistics messages.

Command

Description: Specifies the category of the message.
Datatype: MQLONG.
Value: See the following sections for applicable values:

- "Event message MQCFH (PCF header)" on page 56.
- "Activity report MQCFH (PCF header)" on page 213.
- "Trace-route message MQCFH (PCF header)" on page 237.
- "Accounting and statistics message MQCFH (PCF header)" on page 262.

MsgSeqNumber

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

Control

Description:	Control options.
Datatype:	MQLONG.
Value:	**MQCFC_LAST** Last message in the set. **MQCFC_NOT_LAST** Not the last message in the set.

CompCode

Description:	Completion code.
Datatype:	MQLONG.
Value:	**MQCC_OK** Events reporting OK condition, activity reports, trace-route messages, accounting messages, or statistics messages. **MQCC_WARNING** Event reporting warning condition.

Reason

Description:	Reason code qualifying completion code.
Datatype:	MQLONG.
Value:	For event messages: **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. For activity reports, trace-route messages, accounting messages, and statistics messages: **MQRC_NONE**

ParameterCount

Description:	Count of parameter structures. This is the number of parameter structures that follow the MQCFH structure.
Datatype:	MQLONG.
Value:	0 or greater.

Language declarations

This structure is available in the following languages:

C language declaration

```
typedef struct tagMQCFH {
  MQLONG  Type;           /* Structure type */
  MQLONG  StrucLength;    /* Structure length */
  MQLONG  Version;        /* Structure version number */
  MQLONG  Command;        /* Command identifier */
  MQLONG  MsgSeqNumber;   /* Message sequence number */
  MQLONG  Control;        /* Control options */
  MQLONG  CompCode;        /* Completion code */
  MQLONG  Reason;          /* Reason code qualifying completion code */
  MQLONG  ParameterCount;  /* Count of parameter structures */
 } MQCFH;
```

COBOL language declaration

```
**   MQCFH structure
  10 MQCFH.
**    Structure type
   15 MQCFH-TYPE           PIC S9(9) BINARY.
**    Structure length
   15 MQCFH-STRUCLENGTH    PIC S9(9) BINARY.
**    Structure version number
   15 MQCFH-VERSION        PIC S9(9) BINARY.
**    Command identifier
   15 MQCFH-COMMAND        PIC S9(9) BINARY.
**    Message sequence number
   15 MQCFH-MSGSEQNUMBER   PIC S9(9) BINARY.
**    Control options
   15 MQCFH-CONTROL        PIC S9(9) BINARY.
**    Completion code
   15 MQCFH-COMPCODE       PIC S9(9) BINARY.
**    Reason code qualifying completion code
   15 MQCFH-REASON         PIC S9(9) BINARY.
**    Count of parameter structures
   15 MQCFH-PARAMETERCOUNT PIC S9(9) BINARY.
```

PL/I language declaration (z/OS and Windows)

```
dcl
 1 MQCFH based,
  3 Type           fixed bin(31), /* Structure type */
  3 StrucLength    fixed bin(31), /* Structure length */
  3 Version        fixed bin(31), /* Structure version number */
  3 Command        fixed bin(31), /* Command identifier */
  3 MsgSeqNumber   fixed bin(31), /* Message sequence number */
  3 Control        fixed bin(31), /* Control options */
  3 CompCode       fixed bin(31), /* Completion code */
  3 Reason         fixed bin(31), /* Reason code qualifying completion
                                     code */
  3 ParameterCount fixed bin(31); /* Count of parameter structures */
```

RPG language declaration (i5/OS only)

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

System/390 assembler-language declaration (z/OS only)

```
MQCFH                          DSECT
MQCFH_TYPE                     DS   F        Structure type
MQCFH_STRUCLENGTH              DS   F        Structure length
MQCFH_VERSION                  DS   F        Structure version number
MQCFH_COMMAND                  DS   F        Command identifier
MQCFH_MSGSEQNUMBER             DS   F        Message sequence number
MQCFH_CONTROL                  DS   F        Control options
MQCFH_COMPCODE                 DS   F        Completion code
MQCFH_REASON                   DS   F        Reason code qualifying
*                                            completion code
MQCFH_PARAMETERCOUNT           DS   F        Count of parameter
*                                            structures
MQCFH_LENGTH                   EQU  *-MQCFH  Length of structure
                               ORG  MQCFH
MQCFH_AREA                     DS   CL(MQCFH_LENGTH)
```

Visual Basic language declaration (Windows only)

```
Type MQCFH
  Type As Long            'Structure type
  StrucLength As Long     'Structure length
  Version As Long         'Structure version number
  Command As Long         'Command identifier
  MsgSeqNumber As Long    'Message sequence number
  Control As Long         'Control options
  CompCode As Long        'Completion code
  Reason As Long          'Reason code qualifying completion code
  ParameterCount As Long  'Count of parameter structures
End Type
```

MQCFIL - Integer list parameter

The MQCFIL structure describes an integer list parameter.

Type

Description:	Indicates that the structure type is MQCFIL and describes an integer-list parameter.
Datatype:	MQLONG.
Value:	**MQCFT_INTEGER_LIST** Structure defining an integer list.

StrucLength

Description:	Length in bytes of the MQCFIL structure, including the array of integers at the end of the structure (the *values* field).
Datatype:	MQLONG.

Parameter

Description:	Identifies the parameter whose value is contained in the structure.
Datatype:	MQLONG.

Count

Description:	Number of elements in the *Values* array.
Datatype:	MQLONG.
Values:	Zero or greater.

Values

Description:	Array of values for the parameter identified by the *Parameter* field.
Datatype:	MQLONG×*Count*.

The way that this field is declared depends on the programming language:

- For the C programming language, the field is declared as an array with one element. Storage for the structure must be allocated dynamically, and pointers used to address the fields within it.
- For the COBOL, PL/I, RPG, and System/390® assembler programming languages, the field is omitted from the structure declaration. When an instance of the structure is declared, you must include MQCFIL in a larger structure, and declare additional fields following MQCFIL, to represent the Values field as required.

C language declaration (MQCFIL)

```
typedef struct tagMQCFIL {
  MQLONG  Type;         /* Structure type */
  MQLONG  StrucLength;  /* Structure length */
  MQLONG  Parameter;    /* Parameter identifier */
  MQLONG  Count;        /* Count of parameter values */
  MQLONG  Values[1];    /* Parameter values - first element */
 } MQCFIL;
```

COBOL language declaration (MQCFIL)

```
**   MQCFIL structure
  10 MQCFIL.
**    Structure type
   15 MQCFIL-TYPE        PIC S9(9) BINARY.
**    Structure length
   15 MQCFIL-STRUCLENGTH PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFIL-PARAMETER   PIC S9(9) BINARY.
**    Count of parameter values
   15 MQCFIL-COUNT       PIC S9(9) BINARY.
```

PL/I language declaration (MQCFIL)

```
dcl
 1 MQCFIL based,
  3 Type        fixed bin(31), /* Structure type */
  3 StrucLength fixed bin(31), /* Structure length */
  3 Parameter   fixed bin(31), /* Parameter identifier */
  3 Count       fixed bin(31); /* Count of parameter values */
```

RPG/ILE declaration (MQCFIL) (i5/OS only)

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQCFIL Structure
D*
D* Structure type
D  ILTYP                  1      4I 0
D* Structure length
D  ILLEN                  5      8I 0
D* Parameter identifier
D  ILPRM                  9     12I 0
D* Count of paramter valuee
D  ILCNT                 13     16I 0
```

System/390 assembler-language declaration (MQCFIL)

```
MQCFIL                       DSECT
MQCFIL_TYPE                  DS   F        Structure type
MQCFIL_STRUCLENGTH           DS   F        Structure length
MQCFIL_PARAMETER             DS   F        Parameter identifier
MQCFIL_COUNT                 DS   F        Count of parameter values
MQCFIL_LENGTH                EQU  *-MQCFIL Length of structure
                             ORG  MQCFIL
MQCFIL_AREA                  DS   CL(MQCFIL_LENGTH)
```

Visual Basic language declaration (MQCFIL)

```
Type MQCFIL
  Type As Long         ' Structure type
  StrucLength As Long  ' Structure length
  Parameter As Long    ' Parameter identifier
  Count As Long        ' Count of parameter value
End Type
```

MQCFIL64 - 64–bit integer list parameter

The MQCFIL64 structure describes a 64–bit integer list parameter.

Type

Description:	Indicates that the structure is a MQCFIL64 structure describing a 64–bit integer list parameter.
Datatype:	MQLONG.
Value:	**MQCFT_INTEGER64_LIST** Structure defining a 64–bit integer list.

StrucLength

Description:	Length in bytes of the MQCFIL64 structure, including the array of integers at the end of the structure (the *Values* field).
Datatype:	MQLONG.

Parameter

Description:	Identifies the parameter whose value is contained in the structure.
Datatype:	MQLONG.

Count

Description:	Number of elements in the *Values* array.
Datatype:	MQLONG.
Values:	0 or greater.

Values

Description:	Array of values for the parameter identified by the *Parameter* field.
Datatype:	(MQINT64×*Count*)

The way that this field is declared depends on the programming language:

- For the C programming language, the field is declared as an array with one element. Storage for the structure must be allocated dynamically, and pointers used to address the fields within it.

- For the COBOL, PL/I, RPG, and System/390 assembler programming languages, the field is omitted from the structure declaration. When an instance of the structure is declared, you must include MQCFIL64 in a larger structure, and declare additional fields following MQCFIL64, to represent the *Values* field as required.

For COBOL, additional fields should be declared as:

```
PIC  S9(18)
```

For PL/I, additional fields should be declared as `FIXED BINARY SIGNED` with a precision of 63.

For System/390 assembler, additional fields should be declared `D` (double word) in the `DS` declaration.

C language declaration (MQCFIL64)

```
typedef struct tagMQCFIN64 {
  MQLONG  Type;         /* Structure type */
  MQLONG  StrucLength;  /* Structure length */
  MQLONG  Parameter;    /* Parameter identifier */
  MQLONG  Count;        /* Count of parameter values */
  MQINT64 Values[1];        /* Parameter value */
 } MQCFIL64;
```

COBOL language declaration (MQCFIL64)

```
**   MQCFIL64 structure
  10 MQCFIL64.
**    Structure type
   15 MQCFIL64-TYPE        PIC S9(9) BINARY.
**    Structure length
   15 MQCFIL64-STRUCLENGTH PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFIL64-PARAMETER   PIC S9(9) BINARY.
**    Count of parameter values
   15 MQCFIL64-COUNT       PIC S9(9) BINARY.
```

PL/I language declaration (MQCFIL64)

```
dcl
 1 MQCFIL64 based,
  3 Type        fixed bin(31), /* Structure type */
  3 StrucLength fixed bin(31), /* Structure length */
  3 Parameter   fixed bin(31), /* Parameter identifier */
  3 Count       fixed bin(31)  /* Count of parameter values */
```

RPG/ILE language declaration (MQCFIL64) (i5/OS only)

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQCFIL64 Structure
D*
D* Structure type
D  IL64TYP               1      4I 0 INZ(25)
D* Structure length
D  IL64LEN               5      8I 0 INZ(16)
D* Parameter identifier
D  IL64PRM               9     12I 0 INZ(0)
D* Count of parameter values
D  IL64CNT              13     16I 0 INZ(0)
D* Parameter values -- first element
D  IL64VAL              17     16    INZ(0)
```

System/390 assembler-language declaration (MQCFIL64) (z/OS only)

```
MQCFIL64                          DSECT
MQCFIL64_TYPE                     DS   F        Structure type
MQCFIL64_STRUCLENGTH              DS   F        Structure length
MQCFIL64_PARAMETER                DS   F        Parameter identifier
MQCFIL64_COUNT                    DS   F        Parameter value high
MQCFIL64_LENGTH                   EQU  *-MQCFIL64 Length of structure
                                ORG  MQCFIL64
MQCFIL64_AREA                     DS   CL(MQCFIL64_LENGTH)
```

MQCFIN - Integer parameter

The MQCFIN structure describes an integer parameter.

Type

Description:	Indicates that the structure type is MQCFIN and describes an integer parameter.
Datatype:	MQLONG.
Value:	**MQCFT_INTEGER** Structure defining an integer.

StrucLength

Description:	Length in bytes of the MQCFIN structure.
Datatype:	MQLONG.
Value:	**MQCFIN_STRUC_LENGTH** Length of MQCFIN structure.

Parameter

Description:	Identifies the parameter whose value is contained in the structure.
Datatype:	MQLONG.

Value

Description:	Value of parameter identified by the *Parameter* field.
Datatype:	MQLONG.

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) (i5/OS 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
```

MQCFIN64 - 64–bit integer parameter

The MQCFIN64 structure describes a 64–bit integer parameter.

Type

Description:	Indicates that the structure is a MQCFIN64 structure describing a 64–bit integer parameter.
Datatype:	MQLONG.
Value:	**MQCFT_INTEGER64** Structure defining a 64–bit integer.

StrucLength

Description:	Length in bytes of the MQCFIN64 structure.
Datatype:	MQLONG.
Value:	**MQCFIN64_STRUC_LENGTH** Length of 64–bit integer parameter structure.

Parameter

Description:	Identifies the parameter whose value is contained in the structure.
Datatype:	MQLONG.

Values

Description:	This is the value of the parameter identified by the *Parameter* field.
Datatype:	(MQINT64)

C language declaration (MQCFIN64)

```
typedef struct tagMQCFIN64 {
  MQLONG  Type;         /* Structure type */
  MQLONG  StrucLength;  /* Structure length */
  MQLONG  Parameter;    /* Parameter identifier */
  MQLONG  Reserved;     /* Reserved */
  MQINT64 Value;        /* Parameter value */
 } MQCFIN64;
```

COBOL language declaration (MQCFIN64)

```
**   MQCFIN64 structure
  10 MQCFIN64.
**    Structure type
   15 MQCFIN64-TYPE        PIC S9(9) BINARY.
**    Structure length
   15 MQCFIN64-STRUCLENGTH PIC S9(9) BINARY.
**    Parameter identifier
   15 MQCFIN64-PARAMETER   PIC S9(9) BINARY.
**    Reserved
   15 MQCFIN64-RESERVED    PIC S9(9) BINARY.
**    Parameter value
   15 MQCFIN64-VALUE       PIC S9(18) BINARY.
```

PL/I language declaration (MQCFIN64)

```
dcl
 1 MQCFIN64 based,
  3 Type        fixed bin(31), /* Structure type */
  3 StrucLength fixed bin(31), /* Structure length */
  3 Parameter   fixed bin(31), /* Parameter identifier */
  3 Reserved    fixed bin(31)  /* Reserved */
  3 Value       fixed bin(63); /* Parameter value */
```

RPG/ILE language declaration (MQCFIN64) (i5/OS only)

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQCFIN64 Structure
D*
D* Structure type
D  IN64TYP               1      4I 0 INZ(23)
D* Structure length
D  IN64LEN               5      8I 0 INZ(24)
D* Parameter identifier
```

```
D  IN64PRM              9     12I 0 INZ(0)
D* Reserved field
D  IN64RSV             13     16I 0 INZ(0)
D* Parameter value
D  IN64VAL             17     16    INZ(0)
```

System/390 assembler-language declaration (MQCFIN64) (z/OS only)

```
MQCFIN64                         DSECT
MQCFIN64_TYPE                    DS   F        Structure type
MQCFIN64_STRUCLENGTH             DS   F        Structure length
MQCFIN64_PARAMETER               DS   F        Parameter identifier
MQCFIN64_RESERVED                DS   F        Reserved
MQCFIN64_VALUE                   DS   D        Parameter value
MQCFIN64_LENGTH                  EQU  *-MQCFIN64 Length of structure
                               ORG  MQCFIN64
MQCFIN64_AREA                    DS   CL(MQCFIN64_LENGTH)
```

MQCFSL - String list parameter

The MQCFSL structure describes a string list parameter.

Type

Description:	This indicates that the structure is an MQCFSL structure describing a string-list parameter.
Datatype:	MQLONG.
Value:	**MQCFT_STRING_LIST** Structure defining a string list.

StrucLength

Description:	This is the length in bytes of the MQCFSL structure, including the array of strings at the end of the structure (the *Strings* field).
Datatype:	MQLONG.

Parameter

Description:	This identifies the parameter whose values are contained in the structure.
Datatype:	MQLONG.

CodedCharSetId

Description:	This specifies the coded character set identifier of the data in the *Strings* field.
Datatype:	MQLONG.

Count

Description:	This is the number of strings present in the *Strings* field; zero or greater.
Datatype:	MQLONG.

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.
Datatype:	MQLONG.

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 can be 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).

Datatype: MQCHAR × *StringLength*×*Count*.

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) (i5/OS 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.

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.
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).
Datatype:	MQLONG.

Parameter

Description:	Identifies the parameter whose value is contained in the structure.
Datatype:	MQLONG.
Values:	Dependent on the event message.

CodedCharSetId

Description:	Coded character set identifier of the data in the *String* field.
Datatype:	MQLONG.

StringLength

Description:	Length in bytes of the data in the *String* field; zero or greater.
Datatype:	MQLONG.

String

Description: The value of the parameter identified by the *Parameter* field.

In MQFMT_EVENT messages, trailing blanks can be 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*.

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.

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;
```

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) (i5/OS 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
```

MQEPH - Embedded PCF header

The MQEPH structure describes the additional data that is present in a message when that message is a programmable command format (PCF) message. The additional data consists of the MQEPH structure followed by an array of PCF parameter structures. To include the MQEPH structure in a message, the *Format* parameter in the message descriptor is set to MQFMT_EMBEDDED.

StrucId

Description:	Structure identifier.
Datatype:	MQCHAR4.
Value:	**MQEPH_STRUC_ID** Identifier for distribution header structure.

Version

Description:	Structure version number.
Datatype:	MQLONG.
Value:	**MQEPH_VERSION_1** Version number for embedded PCF header structure.

StrucLength

Description:	Structure length. This is the length in bytes of the MQEPH structure and is set to the amount of data preceding the next header structure.
Datatype:	MQLONG.

Encoding

Description:	Numeric encoding. This specifies the numeric encoding of the data that follows the last PCF parameter structure.
Datatype:	MQLONG.

CodedCharSetId

Description:	Coded character set identifier. This specifies the coded character set identifier of the data that follows the last PCF parameter structure.
Datatype:	MQLONG.

Format

Description:	Format. This specifies the format name of the data that follows the last PCF parameter structure.
Datatype:	MQCHAR8.

Flags

Description:	Flags. This is a reserved field.
Datatype:	MQLONG.

Value:	**MQEPH_NONE** No flags have been specified. **MQEPH_CCSID_EMBEDDED** The character set of the parameters containing character data is specified individually within the CodedCharSetId field in each structure. The character set of the StrucId and Format fields is defined by the CodedCharSetId field in the header structure that precedes the MQEPH structure, or by the CodedCharSetId field in the MQMD if the MQEPH is at the start of the message.

PCFHeader

Description:	Command format header.
Datatype:	MQCFH.

Language declarations

This structure is available in the following languages:

C language declaration

```
struct tagMQEPH  {
  MQCHAR4 StrucId;         /* Structure identifier */
  MQLONG  Version;         /* Structure version number */
  MQLONG  StrucLength      /* Structure length */
  MQLONG  Encoding;        /* Numeric encoding */
  MQLONG  CodedCharSetId;  /* Coded character set identifier */
  MQCHAR8 Format;          /* Data format */
  MQLONG  Flags;           /* Flags */
  MQCFH   PCFHeader;       /* PCF header */
 } MQEPH;
```

COBOL language declaration

```
**   MQEPH structure
  10 MQEPH.
**    Structure identifier
   15 MQEPH-STRUCID       PIC X(4).
**    Structure version number
   15 MQEPH-VERSION       PIC S9(9) BINARY.
**    Structure length
   15 MQEPH-STRUCLENGTH   PIC S9(9) BINARY.
**    Numeric encoding
   15 MQEPH-ENCODING      PIC S9(9) BINARY.
**    Coded characeter set identifier
   15 MQEPH-CODEDCHARSETID PIC S9(9) BINARY.
**    Data format
   15 MQEPH-FORMAT        PIC X(8).
**    Flags
   15 MQEPH-FLAGS         PIC S9(9) BINARY.
**    PCF header
   15 MQEPH-PCFHEADER.
**    Structure type
    20 MQEPH-PCFHEADER-TYPE            PIC S9(9) BINARY.
**    Structure length
    20 MQEPH-PCFHEADER-STRUCLENGTH     PIC S9(9) BINARY.
**    Structure version number
    20 MQEPH-PCFHEADER-VERSION         PIC S9(9) BINARY.
**    Command identifier
    20 MQEPH-PCFHEADER-COMMAND         PIC S9(9) BINARY.
**    Message sequence number
    20 MQEPH-PCFHEADER-MSGSEQNUMBER    PIC S9(9) BINARY.
```

```
**    Control options
    20 MQEPH-PCFHEADER-CONTROL         PIC S9(9) BINARY.
**    Completion code
    20 MQEPH-PCFHEADER-COMPCODE        PIC S9(9) BINARY.
**    Reason code qualifying completion code
    20 MQEPH-PCFHEADER-REASON          PIC S9(9) BINARY.
**    Count of parameter structures
    20 MQEPH-PCFHEADER-PARAMETERCOUNT  PIC S9(9) BINARY.
```

PL/I language declaration (z/OS and Windows)

```
dcl
 1 MQEPH based,
  3 StrucId         char(4),       /* Structure identifier */
  3 Version         fixed bin(31), /* Structure version number */
  3 StrucLength     fixed bin(31), /* Structure length */
  3 Encoding        fixed bin(31), /* Numeric encoding */
  3 CodedCharSetId  fixed bin(31), /* Coded character set identifier */
  3 Format          char(8),       /* Data format */
  3 Flags           fixed bin(31), /* Flags */
  3 PCFHeader,                     /* PCF header */
   5 Type           fixed bin(31), /* Structure type */
   5 StrucLength    fixed bin(31), /* Structure length */
   5 Version        fixed bin(31), /* Structure version number */
   5 Command        fixed bin(31), /* Command identifier */
   5 MsgSeqNumber   fixed bin(31), /* Message sequence number */
   5 Control        fixed bin(31), /* Control options */
   5 CompCode       fixed bin(31), /* Completion code */
   5 Reason         fixed bin(31), /* Reason code qualifying completion
                                      code */
   5 ParameterCount fixed bin(31); /* Count of parameter structures */
```

RPG language declaration (i5/OS only)

```
D*..1....:....2....:....3....:....4....:....5....:....6....:....7..
D* MQEPH Structure
D*
D* Structure identifier
D  EPSID                  1      4    INZ('EPH ')
D* Structure version number
D  EPVER                  5      8I 0 INZ(1)
D* Structure length
D  EPLEN                  9     12I 0 INZ(68)
D* Numeric encoding
D  EPENC                 13     16I 0 INZ(0)
D* Coded character set identifier
D  EPCSI                 17     20I 0 INZ(0)
D* Format name
D  EPFMT                 21     28I 0 INZ('        ')
D* Flags
D  EPFLG                 29     32I 0 INZ(0)
D* Programmable Command Format Header
D*
D* Structure type
D  EP1TYPE               33     36I 0 INZ(0)
D* Structure length
D  EP1LEN                37     40I 0 INZ(36)
D* Structure version number
D  EP1VER                41     44I 0 INZ(3)
D* Command identifier
D  EP1CMD                45     48I 0 INZ(0)
D* Message sequence number
D  EP1SEQ                49     52I 0 INZ(1)
D* Control options
D  EP1CTL                53     56I 0 INZ(1)
D* Completion code
D  EP1CMP                57     60I 0 INZ(0)
D* Reason code qualifying completion code
```

```
D  EP1REA                61     64I 0 INZ(0)
D* Count of parameter structures
D  EP1CNT                65     68I 0 INZ(0)
```

System/390 assembler-language declaration (z/OS only)

```
MQEPH                        DSECT
MQEPH_STRUCID                DS   CL4      Structure identifier
MQEPH_VERSION                DS   F        Structure version number
MQEPH_STRUCLENGTH            DS   F        Structure length
MQEPH_ENCODING               DS   F        Numeric encoding
MQEPH_CODEDCHARSETID         DS   F        Coded character set identifier
MQEPH_FORMAT                 DS   CL8      Data format
MQEPH_FLAGS                  DS   F        Flags
MQEPH_PCFHEADER              DS   0F       Force fullword alignment
MQEPH_PCFHEADER_TYPE         DS   F        Structure type
MQEPH_PCFHEADER_STRUCLENGTH  DS   F        Structure length
MQEPH_PCFHEADER_VERSION      DS   F        Structure version number
MQEPH_PCFHEADER_COMMAND      DS   F        Command identifier
MQEPH_PCFHEADER_MSGSEQNUMBER DS   F        Message sequence number
MQEPH_PCFHEADER_CONTROL      DS   F        Control options
MQEPH_PCFHEADER_COMPCODE     DS   F        Completion code
MQEPH_PCFHEADER_REASON       DS   F        Reason code qualifying completion code
MQEPH_PCFHEADER_PARAMETERCOUNT DS   F        Count of parameter structures
MQEPH_PCFHEADER_LENGTH       EQU  *-MQEPH_PCFHEADER
                             ORG  MQEPH_PCFHEADER
MQEPH_PCFHEADER_AREA         DS   CL(MQEPH_PCFHEADER_LENGTH)
*
MQEPH_LENGTH                 EQU  *-MQEPH
                             ORG  MQEPH
MQEPH_AREA                   DS   CL(MQEPH_LENGTH)
```

Visual Basic language declaration (Windows only)

```
Type MQEPH
  StrucId As String*4     'Structure identifier
  Version As Long         'Structure version number
  StrucLength As Long     'Structure length
  Encoding As Long        'Numeric encoding
  CodedCharSetId As Long  'Coded characetr set identifier
  Format As String*8      'Format name
  Flags As Long           'Flags
  Reason As Long          'Reason code qualifying completion code
  PCFHeader As MQCFH      'PCF header
End Type
```

Chapter 7. Event data for object attributes

This appendix specifies the object attributes 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

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.

AuthInfoConnName **(MQCFST)**
Authentication information connection name (parameter identifier: MQCA_AUTH_INFO_CONN_NAME).

The maximum length of the string is 48.

AuthInfoDesc **(MQCFST)**
Authentication information description (parameter identifier: MQCA_AUTH_INFO_DESC).

The maximum length of the string is MQ_AUTH_INFO_DESC_LENGTH.

AuthInfoType **(MQCFIN)**
Authentication information type (parameter identifier: MQIA_AUTH_INFO_TYPE).

The value is MQAIT_CRL_LDAP.

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

The maximum length of the string is MQ_LDAP_PASSWORD_LENGTH.

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

The maximum length of the string is 256.

CF structure 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.

CFLevel **(MQCFIN)**
CF level (parameter identifier: MQIA_CF_LEVEL).

CFStrucDesc **(MQCFST)**
CF Structure description (parameter identifier: MQCA_CF_STRUC_DESC).

The maximum length of the string is MQCA_CF_STRUC_DESC_LENGTH.

Recovery **(MQCFIN)**
Recovery (parameter identifier: MQIA_CF_RECOVER).

Channel attributes

Only those attributes that apply to the type of channel in question are included in the event data.

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.

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.

BatchInterval **(MQCFIN)**
Batch interval (parameter identifier: MQIACH_BATCH_INTERVAL).

BatchSize **(MQCFIN)**
Batch size (parameter identifier: MQIACH_BATCH_SIZE).

ChannelDesc **(MQCFST)**
Channel description (parameter identifier: MQCACH_DESC).

The maximum length of the string is MQ_CHANNEL_DESC_LENGTH.

ChannelMonitoring **(MQCFIN)**
Level of monitoring data collection for the channel (parameter identifier: MQIA_MONITORING_CHANNEL).

The value can be:

MQMON_OFF
Monitoring data collection is turned off.

MQMON_LOW
Monitoring data collection is turned on with a low ratio of data collection.

MQMON_MEDIUM
Monitoring data collection is turned on with a medium ratio of data collection.

MQMON_HIGH
Monitoring data collection is turned on with a high ratio of data collection.

MQMON_Q_MGR
The level of monitoring data collected is based on the queue manager attribute *ChannelMonitoring*.

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.

CipherSpec **(MQCFST)**
SSL cipher specification (parameter identifier: MQCACH_SSL_CIPHER_SPEC).

The maximum length of the string is MQ_SSL_CIPHER_SPEC_LENGTH.

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

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

CLWLChannelPriority **(MQCFIN)**
Cluster workload channel priority (parameter identifier: MQIACH_CLWL_CHANNEL_PRIORITY).

CLWLChannelRank **(MQCFIN)**
Cluster workload channel rank (parameter identifier: MQIACH_CLWL_CHANNEL_RANK).

CLWLChannelWeight **(MQCFIN)**
Cluster workload channel weight (parameter identifier: MQIACH_CLWL_CHANNEL_WEIGHT).

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

The maximum length of the string is MQ_CONN_NAME_LENGTH.

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.

DiscInterval **(MQCFIN)**
Disconnection interval (parameter identifier: MQIACH_DISC_INTERVAL).

HeaderCompression **(MQCFIL)**
Header data compression techniques supported by the channel (parameter identifier: MQIACH_HDR_COMPRESSION).

For sender, server, cluster-sender, cluster-receiver, and client-connection channels, the values specified are in order of preference.

The value can be one, or more, of the following:

MQCOMPRESS_NONE
No header data compression is performed.

MQCOMPRESS_SYSTEM
Header data compression is performed.

HeartbeatInterval **(MQCFIN)**
Heartbeat interval (parameter identifier: MQIACH_HB_INTERVAL).

KeepAliveInterval **(MQCFIN)**
Keep alive interval (parameter identifier: MQIACH_KEEP_ALIVE_INTERVAL).

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

The maximum length of the string is MQ_LOCAL_ADDRESS_LENGTH.

LongRetryCount **(MQCFIN)**
Long retry count (parameter identifier: MQIACH_LONG_RETRY).

LongRetryInterval **(MQCFIN)**
Long timer (parameter identifier: MQIACH_LONG_TIMER).

MaxMsgLength **(MQCFIN)**
Maximum message length (parameter identifier: MQIACH_MAX_MSG_LENGTH).

MCAName **(MQCFST)**
Message channel agent name (parameter identifier: MQCACH_MCA_NAME).

The maximum length of the string is MQ_MCA_NAME_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.

MessageCompression **(MQCFIL)**
Message data compression techniques supported by the channel (parameter identifier: MQIACH_MSG_COMPRESSION).

For sender, server, cluster-sender, cluster-receiver, and client-connection channels, the values specified are in order of preference.

The value can be one, or more, of:

MQCOMPRESS_NONE
No message data compression is performed. This is the default value.

MQCOMPRESS_RLE
Message data compression is performed using run-length encoding.

MQCOMPRESS_ZLIBFAST
Message data compression is performed using ZLIB encoding with speed prioritized.

MQCOMPRESS_ZLIBHIGH
Message data compression is performed using ZLIB encoding with compression prioritized.

MQCOMPRESS_ANY
Any compression technique supported by the queue manager can be used. This is only valid for receiver, requester, and server-connection channels.

ModeName **(MQCFST)**
Mode name (parameter identifier: MQCACH_MODE_NAME).

The maximum length of the string is MQ_MODE_NAME_LENGTH.

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.

MsgRetryCount **(MQCFIN)**
Message retry count (parameter identifier: MQIACH_MR_COUNT).

Specifies the number of times that a failing message should be retried.

This parameter is only valid for receiver, cluster-receiver, and requester channels.

MsgRetryExit **(MQCFST)**
Message retry exit name (parameter identifier: MQCACH_MR_EXIT_NAME).

This parameter is only valid for receiver, cluster-receiver, and requester channels.

The maximum length of the string is MQ_MAX_EXIT_NAME_LENGTH.

MsgRetryInterval **(MQCFIN)**
Message retry interval (parameter identifier: MQIACH_MR_INTERVAL).

Specifies the minimum time interval in milliseconds between retries of failing messages.

This parameter is only valid for receiver, cluster-receiver, and requester channels.

MsgRetryUserData **(MQCFST)**
Message retry exit user data (parameter identifier: MQCACH_MR_EXIT_USER_DATA).

Specifies user data that is passed to the message retry exit.

This parameter is only valid for receiver, cluster-receiver, and requester channels.

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.

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

NonPersistentMsgSpeed **(MQCFIN)**
Speed at which nonpersistent messages are to be sent (parameter identifier: MQIACH_NPM_SPEED).

The value can be:

MQNPMS_NORMAL
Normal speed.

MQNPMS_FAST
Fast speed.

Password **(MQCFST)**
Password (parameter identifier: MQCACH_PASSWORD).

The maximum length of the string is MQ_PASSWORD_LENGTH.

PeerName **(MQCFST)**
SSL peer name (parameter identifier: MQCACH_SSL_PEER_NAME).

The maximum length of the string is 256.

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.

QMgrName **(MQCFST)**
Queue manager name (parameter identifier: MQCA_Q_MGR_NAME).

The maximum length of the string is MQ_Q_MGR_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.

For a client-connection channel the maximum length of the exit name is MQ_MAX_EXIT_NAME_LENGTH. For all other channels, the maximum length of the exit name is MQ_EXIT_NAME_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.

SecurityExit **(MQCFST)**
Security exit name (parameter identifier: MQCACH_SEC_EXIT_NAME).

For a client-connection channel the maximum length of the exit name is MQ_MAX_EXIT_NAME_LENGTH. For all other channels, the maximum length of the exit name is MQ_EXIT_NAME_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.

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.

For a client-connection channel the maximum length of the exit name is MQ_MAX_EXIT_NAME_LENGTH. For all other channels, the maximum length of the exit name is MQ_EXIT_NAME_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.

SeqNumberWrap **(MQCFIN)**
Sequence wrap number (parameter identifier: MQIACH_SEQUENCE_NUMBER_WRAP).

ShortRetryCount **(MQCFIN)**
Short retry count (parameter identifier: MQIACH_SHORT_RETRY).

ShortRetryInterval **(MQCFIN)**
Short timer (parameter identifier: MQIACH_SHORT_TIMER).

SSLClientAuthentication **(MQCFIN)**
SSL client authentication (parameter identifier: MQIACH_SSL_CLIENT_AUTH).

The value can be:

MQSCA_REQUIRED
Certificate required.

MQSCA_OPTIONAL
Certificate optional.

TpName **(MQCFST)**
Transaction program name (parameter identifier: MQCACH_TP_NAME).

The maximum length of the string is MQ_TP_NAME_LENGTH.

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.

UserIdentifier **(MQCFST)**
Task user identifier (parameter identifier: MQCACH_USER_ID).

The maximum length of the string is MQ_USER_ID_LENGTH.

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

The maximum length of the string is MQ_Q_NAME_LENGTH.

Namelist 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.

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

The number of names contained in the namelist.

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

The maximum length of the string is MQ_NAMELIST_DESC_LENGTH.

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

The maximum length of the string is MQ_NAMELIST_NAME_LENGTH.

NamelistType **(MQCFIN)**
Namelist type (parameter identifier: MQIA_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.

Process 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.

ApplId **(MQCFST)**
Application identifier (parameter identifier: MQCA_APPL_ID).

The maximum length of the string is MQ_PROCESS_APPL_ID_LENGTH.

ApplType **(MQCFIN)**
Application type (parameter identifier: MQIA_APPL_TYPE).

EnvData **(MQCFST)**
Environment data (parameter identifier: MQCA_ENV_DATA).

The maximum length of the string is MQ_PROCESS_ENV_DATA_LENGTH.

ProcessDesc **(MQCFST)**
Description of process definition (parameter identifier: MQCA_PROCESS_DESC).

The maximum length of the string is MQ_PROCESS_DESC_LENGTH.

ProcessName **(MQCFST)**
The name of the process definition (parameter identifier: MQCA_PROCESS_NAME).

The maximum length of the string is MQ_PROCESS_NAME_LENGTH.

UserData **(MQCFST)**
User data (parameter identifier: MQCA_USER_DATA).

The maximum length of the string is MQ_PROCESS_USER_DATA_LENGTH.

Queue attributes

Only those attributes that apply to the type of queue in question are included in the event data.

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.

BackoutRequeueName **(MQCFST)**
Excessive backout requeue name (parameter identifier: MQCA_BACKOUT_REQ_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

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

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.

CFstructure **(MQCFST)**
CF structure name (parameter identifier: MQCA_CF_STRUC_NAME).

The maximum length of the string is MQ_CF_STRUC_NAME_LENGTH.

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

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

CLWLQueuePriority **(MQCFIN)**
Queue priority (parameter identifier: MQIA_CLWL_Q_PRIORITY).

CLWLQueueRank **(MQCFIN)**
Queue rank (parameter identifier: MQIA_CLWL_Q_RANK).

CLWLUseQ **(MQCFIN)**
This defines the behavior of an MQPUT when the target queue has both a local instance and at least one remote cluster instance (parameter identifier: MQIA_CLWL_USEQ).

The value can be:

MQCLWL_USEQ_ANY
Use remote and local queues.

MQCLWL_USEQ_LOCAL
Do not use remote queues.

MQCLWL_USEQ_AS_Q_MGR
Inherit definition from the queue manager attribute *CLWLUseQ*.

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.

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.

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.

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.

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.

DefPriority **(MQCFIN)**
Default priority (parameter identifier: MQIA_DEF_PRIORITY).

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.

IndexType **(MQCFIN)**
Index type (parameter identifier: MQIA_INDEX_TYPE).

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.

InitiationQName **(MQCFST)**
Initiation queue name (parameter identifier: MQCA_INITIATION_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

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

MaxQDepth **(MQCFIN)**
Maximum queue depth (parameter identifier: MQIA_MAX_Q_DEPTH).

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).

ProcessName **(MQCFST)**
Name of process definition for queue (parameter identifier: MQCA_PROCESS_NAME).

The maximum length of the string is MQ_PROCESS_NAME_LENGTH.

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.

QDesc **(MQCFST)**
Queue description (parameter identifier: MQCA_Q_DESC).

The maximum length of the string is MQ_Q_DESC_LENGTH.

QName **(MQCFST)**
Queue name (parameter identifier: MQCA_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

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.

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.

QueueAccounting **(MQCFIN)**
Specifies whether accounting information is collected (parameter identifier: MQIA_ACCOUNTING_Q).

The value can be:

MQMON_ON
Accounting information is collected for the queue.

MQMON_OFF
Accounting information is not collected for the queue.

MQMON_Q_MGR
The collection of accounting information for this queue is based is based on the queue manager attribute *QueueAccounting*.

`QueueMonitoring` **(MQCFIN)**
Level of monitoring data collection for the queue (parameter identifier: MQIA_MONITORING_Q).

The value can be:

MQMON_OFF
Monitoring data collection is turned off.

MQMON_LOW
Monitoring data collection is turned on with a low ratio of data collection.

MQMON_MEDIUM
Monitoring data collection is turned on with a moderate ratio of data collection.

MQMON_HIGH
Monitoring data collection is turned on with a high ratio of data collection.

MQMON_Q_MGR
The level of monitoring data collected is based on the queue manager attribute *QueueMonitoring*.

`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.

`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.

`RetentionInterval` **(MQCFIN)**
Retention interval (parameter identifier: MQIA_RETENTION_INTERVAL).

`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.

`StorageClass` **(MQCFST)**
Storage class name (parameter identifier: MQCA_STORAGE_CLASS).

The maximum length of the string is MQ_STORAGE_CLASS_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.

TriggerData **(MQCFST)**
Trigger data (parameter identifier: MQCA_TRIGGER_DATA).

The maximum length of the string is MQ_TRIGGER_DATA_LENGTH.

TriggerDepth **(MQCFIN)**
Trigger depth (parameter identifier: MQIA_TRIGGER_DEPTH).

TriggerMsgPriority **(MQCFIN)**
Threshold message priority for triggers (parameter identifier: MQIA_TRIGGER_MSG_PRIORITY).

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.

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

The value can be:

MQUS_NORMAL
Normal usage.

MQUS_TRANSMISSION
Transmission queue.

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

The maximum length of the string is MQ_Q_NAME_LENGTH.

Queue manager attributes

ActivityRecording **(MQCFIN)**
Specifies whether activity recording is enabled or disabled (parameter identifier: MQIA_ACTIVITY_RECORDING).

The value can be:

MQRECORDING_MSG
Activity recording is enabled. Activity reports are delivered to the reply-to queue specified in the message descriptor of the message.

MQRECORDING_Q
: Activity recording is enabled. Activity reports are delivered to a fixed name queue.

MQRECORDING_DISABLED.
: Activity recording is disabled.

AdoptNewMCACheck **(MQCFIN)**
: Procedure to determine if an existing receiver MCA is to be adopted when an inbound channel is detected of the same name (parameter identifier: MQIA_ADOPTNEWMCA_CHECK).

The value can be:

MQADOPT_CHECK_Q_MGR_NAME
: Compare the receiver MCA and the inbound channel. If the queue manager names match, the existing receiver MCA is adopted providing it is active. If they don't match, the existing receiver MCA is cancelled, and a new MCA is created.

MQADOPT_CHECK_NET_ADDR
: Compare the receiver MCA and the inbound channel. If the network addresses match, the existing receiver MCA is adopted providing it is active. If they don't match, the existing receiver MCA is cancelled, and a new MCA is created.

MQADOPT_CHECK_ALL
: Compare the receiver MCA and the inbound channel. If both the queue manager names, and the network addresses match, the existing receiver MCA is adopted providing it is active. If they don't match, the existing receiver MCA is cancelled, and a new MCA is created.

MQADOPT_CHECK_NONE
: If the existing receiver MCA is active it is adopted with no checks.

AdoptNewMCAType **(MQCFIN)**
: Specifies whether orphaned receiver MCAs are to be restarted when an inbound channel matching the *AdoptNewMCACheck* procedure is detected (parameter identifier: MQIA_ADOPTNEWMCA_TYPE).

The value can be:

MQADOPT_TYPE_NO
: Do not restart and adopt orphaned receiver MCAs.

MQADOPT_TYPE_ALL
: Restart and adopt orphaned receiver MCAs.

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.

AuthorityEvent **(MQCFIN)**
: Controls whether authorization (Not Authorized) events are generated (parameter identifier: MQIA_AUTHORITY_EVENT).

The value can be:

MQEVR_DISABLED
Event reporting disabled.

BridgeEvent **(MQCFIN)**
Determines whether IMS bridge events are generated (parameter identifier: MQIA_BRIDGE_EVENT).

The value can be:

MQEVR_ENABLED
All IMS bridge events are enabled.

MQEVR_DISABLED
All IMS bridge events are disabled.

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.

ChannelEvent **(MQCFIN)**
Determines whether channel events are generated (parameter identifier: MQIA_CHANNEL_EVENT).

The value can be:

MQEVR_ENABLED
All channel events are enabled.

MQEVR_EXCEPTION
Only the following channels events are enabled:

- MQRC_CHANNEL_ACTIVATED
- MQRC_CHANNEL_CONV_ERROR
- MQRC_CHANNEL_NOT_ACTIVATED
- MQRC_CHANNEL_STOPPED

MQEVR_DISABLED
All channel events are disabled.

ChannelMonitoring **(MQCFIN)**
Level of real-time monitoring data collection for channels (parameter identifier: MQIA_MONITORING_CHANNEL).

The value can be:

MQMON_NONE
Monitoring data collection is disabled, regardless of the setting for the *ChannelMonitoring* channel attribute.

MQMON_OFF
Monitoring data collection is turned off for channels specifying MQMON_Q_MGR in the *ChannelMonitoring* channel attribute.

MQMON_LOW
Monitoring data collection is turned on with a low ratio of data collection for channels specifying MQMON_Q_MGR in the *ChannelMonitoring* channel attribute.

MQMON_MEDIUM
Monitoring data collection is turned on with a moderate ratio of data

collection for channels specifying MQMON_Q_MGR in the *ChannelMonitoring* channel attribute.

MQMON_HIGH
Monitoring data collection is turned on with a high ratio of data collection for channels specifying MQMON_Q_MGR in the *ChannelMonitoring* channel attribute.

ChinitAdapters **(MQCFIN)**
Number of channel initiator adapter subtasks to use for processing WebSphere MQ calls (parameter identifier: MQIA_CHINIT_ADAPTERS).

This value must be between 0 and 9999.

ChinitDispatchers **(MQCFIN)**
Number of dispatchers to use for the channel initiator (parameter identifier: MQIA_CHINIT_DISPATCHERS).

ChinitServiceParm **(MQCFST)**
This attribute is reserved for use by IBM (parameter identifier: MQCA_CHINIT_SERVICE_PARM).

ChinitTraceAutoStart **(MQCFIN)**
Specifies whether the channel initiator trace should start automatically (parameter identifier: MQIA_CHINIT_TRACE_AUTO_START).

The value can be:

MQTRAXSTR_YES
Channel initiator trace starts automatically.

MQTRAXSTR_NO
Channel initiator trace does not starts automatically.

ChinitTraceTableSize **(MQCFIN)**
Size of the channel initiator's trace data space, in MB (parameter identifier: MQIA_CHINIT_TRACE_TABLE_SIZE).

ClusterSenderMonitoring **(MQCFIN)**
Level of real-time monitoring data collection for auto-defined cluster sender channels (parameter identifier: MQIA_MONITORING_AUTO_CLUSSDR).

The value can be:

MQMON_Q_MGR
The collection of monitoring data is inherited from the setting of the *ChannelMonitoring* attribute in the queue manager object.

MQMON_OFF
Monitoring data collection is disabled.

MQMON_LOW
Monitoring data collection is turned on with a low ratio of data collection.

MQMON_MEDIUM
Monitoring data collection is turned on with a moderate ratio of data collection.

MQMON_HIGH
Monitoring data collection is turned on with a high ratio of data collection.

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

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.

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

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

CLWLMRUChannels **(MQCFIN)**
Maximum number of most recently used channels for cluster workload balancing (parameter identifier: MQIA_CLWL_MRU_CHANNELS).

CLWLUseQ **(MQCFIN)**
This defines the behavior of an MQPUT when the target queue has both a local instance and at least one remote cluster instance (parameter identifier: MQIA_CLWL_USEQ).

The value can be:

MQCLWL_USEQ_ANY
Use remote and local queues.

MQCLWL_USEQ_LOCAL
Do not use remote queues.

CodedCharSetId **(MQCFIN)**
Coded character set identifier (parameter identifier: MQIA_CODED_CHAR_SET_ID).

CommandEvent **(MQCFIN)**
Controls whether command events are generated (parameter identifier: MQIA_COMMAND_EVENT).

The value can be:

MQEVR_DISABLED
Command event generation disabled.

MQEVR_ENABLED
Command event generation enabled.

MQEVR_NO_DISPLAY
Command events are generated for all commands other than MQSC DISPLAY commands and PCF Inquire commands.

CommandInputQName **(MQCFST)**
Command input queue name (parameter identifier: MQCA_COMMAND_INPUT_Q_NAME).

The maximum length of the string is MQ_Q_NAME_LENGTH.

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

ConfigurationEvent **(MQCFIN)**
Controls whether configuration events are generated (parameter identifier: MQIA_CONFIGURATION_EVENT).

The value can be:

MQEVR_DISABLED
Configuration event generation disabled.

MQEVR_ENABLED
Configuration event generation enabled.

CPILevel **(MQCFIN)**
CPI level (parameter identifier: MQIA_CPI_LEVEL).

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.

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.

DNSGroup **(MQCFST)**
The name of the group that the TCP listener that handles inbound transmissions for the queue sharing group must join when using Workload Manager for Dynamic Domain Name Services (parameter identifier: MQCA_DNS_GROUP).

The maximum length of this name is MQ_DNS_GROUP_NAME_LENGTH.

DNSWLM **(MQCFIN)**
Specifies whether the TCP listener that handles inbound transmissions for the queue sharing group will register with the Workload Manager for Dynamic Domain Name Services (parameter identifier: MQIA_DNS_WLM).

The value can be:

MQDNSWLM_YES
Register with the Workload Manager for Dynamic Domain Name Services.

MQDNSWLM_NO
Do not register with the Workload Manager for Dynamic Domain Name Services.

ExpiryInterval **(MQCFIN)**
Expiry interval (parameter identifier: MQIA_EXPIRY_INTERVAL).

IGQPutAuthority **(MQCFIN)**
IGQ put authority (parameter identifier: MQIA_IGQ_PUT_AUTHORITY).

IGQUserId **(MQCFST)**
IGQ user identifier (parameter identifier: MQCA_IGQ_USER_ID).

The maximum length of the **string** is MQ_USER_ID_LENGTH.

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.

IntraGroupQueueing **(MQCFIN)**
Intra group queueing (parameter identifier: MQIA_INTRA_GROUP_QUEUING).

IPAddressVersion **(MQCFIN)**
Specifies the IP version to be used (parameter identifier: MQIA_IP_ADDRESS_VERSION).

The value can be:

MQIPADDR_IPV4
The IPv4 stack is used.

MQIPADDR_IPV6
The IPv6 stack is used.

ListenerTimer **(MQCFIN)**
The time interval, in seconds, between attempts to restart a listener following an APPC or TCP/IP failure (parameter identifier: MQCA_LISTENER_TIMER).

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.

LU62ARMSuffix **(MQCFST)**
The suffix of the SYS1.PARMLIB member APPCPMxx, that nominates the LUADD for this channel initiator (parameter identifier: MQCA_LU62_ARM_SUFFIX).

The maximum length of this name is MQ_ARM_SUFFIX_LENGTH.

LU62Channels **(MQCFIN)**
Maximum number of current channels that use the LU 6.2 transmission protocol, including clients connected to server connection channels (parameter identifier: MQIA_LU62_CHANNELS).

LUGroupName **(MQCFST)**
The generic LU name that the LU 6.2 listener that handles inbound transmissions for the queue sharing group is to use. This name must be the same as *LUName* (parameter identifier: MQCA_LU_GROUP_NAME).

The maximum length of this name is MQ_LU_NAME_LENGTH.

LUName **(MQCFST)**
The LU name that the LU 6.2 listener that handles outbound transmissions is to use. This name must be the same as *LUGroupName* (parameter identifier: MQCA_LU_NAME).

The maximum length of this name is MQ_LU_NAME_LENGTH.

MaxActiveChannels **(MQCFIN)**
Maximum number of channels that can be active at the same time (parameter identifier: MQIA_ACTIVE_CHANNELS).

MaxChannels **(MQCFIN)**
Maximum number of current channels, including clients connected to server connection channels (parameter identifier: MQIA_MAX_CHANNELS).

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.

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

MaxPriority **(MQCFIN)**
Maximum priority (parameter identifier: MQIA_MAX_PRIORITY).

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.

OutboundPortMax **(MQCFIN)**
Outbound port range maximum (parameter identifier: MQIA_OUTBOUND_PORT_MAX).

The upper limit for the range of port numbers used when binding outgoing channels.

OutboundPortMin **(MQCFIN)**
Outbound port range minimum (parameter identifier: MQIA_OUTBOUND_PORT_MIN).

The lower limit for the range of port numbers used when binding outgoing channels.

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.

`Platform` **(MQCFIN)**
Platform on which the queue manager resides (parameter identifier: MQIA_PLATFORM).

`QMgrDesc` **(MQCFST)**
Queue manager description (parameter identifier: MQCA_Q_MGR_DESC).

The maximum length of the string is MQ_Q_MGR_DESC_LENGTH.

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

The unique identifier of the queue manager.

`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.

`QSGName` **(MQCFST)**
Queue sharing group name (parameter identifier: MQCA_QSG_NAME).

The maximum length of the string is MQ_QSG_NAME_LENGTH.

`QueueAccounting` **(MQCFIN)**
Specifies whether accounting information is collected for queues (parameter identifier: MQIA_ACCOUNTING_Q).

The value can be:

MQMON_ON
For all queues that have the queue parameter *QueueAccounting* specified as MQMON_Q_MGR, accounting information is collected.

MQMON_OFF
For all queues that have the queue parameter *QueueAccounting* specified as MQMON_Q_MGR, accounting information is not collected.

MQMON_NONE
Accounting information is not collected for queues.

`QueueMonitoring` **(MQCFIN)**
Level of real-time monitoring data collection for queues (parameter identifier: MQIA_MONITORING_Q).

The value can be:

MQMON_NONE
Monitoring data collection is disabled, regardless of the setting for the `QueueMonitoring` queue attribute.

MQMON_OFF
Monitoring data collection is turned off for queues specifying MQMON_Q_MGR in the `QueueMonitoring` queue attribute.

MQMON_LOW
Monitoring data collection is turned on with a low ratio of data collection for queues specifying MQMON_Q_MGR in the `QueueMonitoring` queue attribute.

MQMON_MEDIUM
: Monitoring data collection is turned on with a moderate ratio of data collection for queues specifying MQMON_Q_MGR in the *QueueMonitoring* queue attribute.

MQMON_HIGH
: Monitoring data collection is turned on with a high ratio of data collection for queues specifying MQMON_Q_MGR in the *QueueMonitoring* queue attribute.

ReceiveTimeout **(MQCFIN)**
: In conjunction with *ReceiveTimeoutType* specifies how long a TCP/IP channel will wait to receive data, including heartbeats, from its partner before returning to the inactive state (parameter identifier: MQIA_RECEIVE_TIMEOUT).

ReceiveTimeoutMin **(MQCFIN)**
: The minimum time, in seconds, that a TCP/IP channel will wait to receive data, including heartbeats, from its partner before returning to the inactive state (parameter identifier: MQIA_RECEIVE_TIMEOUT_MIN).

ReceiveTimeoutType **(MQCFIN)**
: In conjunction with *ReceiveTimeout* specifies how long a TCP/IP channel will wait to receive data, including heartbeats, from its partner before returning to the inactive state (parameter identifier: MQIA_RECEIVE_TIMEOUT_TYPE).

The value can be:

MQRCVTIME_MULTIPLY
: The *ReceiveTimeout* value is a multiplier to be applied to the negotiated value of *HeartbeatInterval* to determine how long a channel will wait. This is the queue manager's initial default value.

MQRCVTIME_ADD
: *ReceiveTimeout* is a value, in seconds, to be added to the negotiated value of *HeartbeatInterval* to determine how long a channel will wait.

MQRCVTIME_EQUAL
: *ReceiveTimeout* is a value, in seconds, representing how long a channel will wait.

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.

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.

SharedQueueQueueManagerName **(MQCFIN)**
Specifies how messages are put on a shared queue that specifies another queue manager from a queue sharing group as the object queue manager (parameter identifier: MQIA_SHARED_Q_Q_MGR_NAME).

The value can be:

MQSQQM_USE
Messages are delivered to the object queue manager before being put on the shared queue.

MQSQQM_IGNORE
Messages are put directly on the shared queue.

SSLCRLNameList **(MQCFST)**
SSL CRL name list (parameter identifier: MQCA_SSL_CRL_NAMELIST).

The maximum length of the string is MQ_NAMELIST_NAME_LENGTH.

SSLEvent **(MQCFIN)**
Determines whether IMS bridge events are generated (parameter identifier: MQIA_SSL_EVENT).

The value can be:

MQEVR_ENABLED
All SSL events are enabled.

MQEVR_DISABLED
All SSL events are disabled.

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

The maximum length of the string is MQ_SSL_KEY_REPOSITORY_LENGTH.

SSLKeyResetCount **(MQCFIN)**
SSL key reset count (parameter identifier: MQIA_SSL_RESET_COUNT).

The maximum length of the string is MQ_SSL_KEY_REPOSITORY_LENGTH.

SSLTasks **(MQCFIN)**
SSL tasks (parameter identifier: MQIA_SSL_TASKS).

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.

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

TCPChannels **(MQCFIN)**
Maximum number of current channels that use the TCP/IP transmission protocol, including clients connected to server connection channels (parameter identifier: MQIA_TCP_CHANNELS).

TCPKeepAlive **(MQCFIN)**
: Specifies whether to use the TCP KEEPALIVE facility to check whether the MCA at the opposite end of a channel is available (parameter identifier: MQIA_TCP_KEEP_ALIVE).

  The value can be:

  MQTCPKEEP_YES
  : Use the TCP KEEPALIVE facility as specified in the TCP profile configuration data set.

  MQTCPKEEP_NO
  : Do not use the TCP KEEPALIVE facility.

TCPName **(MQCFST)**
: TCP name (parameter identifier: MQIA_TCP_NAME).

  The name of the current TCP/IP system in use.

  The maximum length of this value is MQ_TCP_NAME_LENGTH.

TCPStackType **(MQCFIN)**
: TCP stack type (parameter identifier: MQIA_TCP_STACK_TYPE).

  Specifies whether the channel initiator uses the TCP/IP address space specified in TCPNAME only, or whether it can bind to any selected TCP/IP address.

  The value can be:

  MQTCPSTACK_SINGLE
  : The channel initiator uses the TCP/IP address space specified in TCPNAME only.

  MQTCPSTACK_MULTIPLE
  : The initiator can use any TCP/IP address space available to it. If no other address spaces are available, the address space specified in TCPNAME is used.

TraceRouteRecording **(MQCFIN)**
: Specifies whether trace-route messaging is enabled or disabled (parameter identifier: MQIA_TRACE_ROUTE_RECORDING).

  The value can be:

  MQRECORDING_MSG
  : Trace-route messaging is enabled. Trace-route reply messages are delivered to the reply-to queue specified in the message descriptor of the message.

  MQRECORDING_Q
  : Trace-route messaging is enabled. Trace-route reply messages are delivered to a fixed name queue.

  MQRECORDING_DISABLED.
  : Trace-route messaging is disabled.

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.

Storage class 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.

PageSetId **(MQCFIN)**
Page set identifier (parameter identifier: MQIA_PAGESET_ID).

PassTicketApplication **(MQCFST)**
Name of the application used to authenticate IMS bridge passtickets (parameter identifier: MQCA_PASS_TICKET_APPL).

The maximum length of the string is MQ_PASS_TICKET_APPL_LENGTH.

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.

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-k,u
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.

AIX	i5/OS	IBM
IBMLink™	IMS	MQSeries
NetView	S/390	System/390
WebSphere	z/OS	

Index

Sending your comments to IBM

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book.

Please limit your comments to the information in this book and the way in which the information is presented.

To make comments about the functions of IBM products or systems, talk to your IBM representative or to your IBM authorized remarketer.

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

You can send your comments to IBM in any of the following ways:
- By mail, to this address:

  User Technologies Department (MP095)
  IBM United Kingdom Laboratories
  Hursley Park
  WINCHESTER,
  Hampshire
  SO21 2JN
  United Kingdom
- By fax:
  - From outside the U.K., after your international access code use 44-1962-816151
  - From within the U.K., use 01962-816151
- Electronically, use the appropriate network ID:
  - IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
  - IBMLink: HURSLEY(IDRCF)
  - Internet: idrcf@hursley.ibm.com

Whichever method you use, ensure that you include:
- The publication title and order number
- The topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.

SC34-6937-00

Spine information:

IBM WebSphere MQ Monitoring WebSphere MQ Version 7.0