MQGem Software

MQEV

IBM MQ Event Processor
User Guide

Version 9.4.0

25th June 2024

MQGem Software Limited

www.mqgem.com

support@mqgem.com

Notices

Eighteenth Edition, June 2024
This edition applies to Version 9.4.0 of IBM MQ Event Processor and to all subsequent releases and modifications until otherwise indicated in new editions.

Table of Contents

Main changes from previous version

Unless otherwise stated the behaviour of the previous version should be maintained and, if all goes well, enhanced. While every effort is made to try and ensure that there are as few bugs as possible it would be surprising if some problems didn't leak out.

When installing a new version of **MQEV** we would always recommend that users...

- ✔ take a backup of their data queue(s) using something like `DMPMQMSG` or `QLOAD`. See 3.2 Upgrade on page 14.
- ✔ keep their previous version of **MQEV** handy so that they can revert to it if a bug is found.

Needless to say, please report any incompatibilities to us if they are found and we will do our best to provide a fix.

The following changes have been made in this version:

1. **IBM MQ 9.4 Command Level Support**
   In keeping with the notion that the first two numbers of MQGem products reflect the version of IBM MQ product they support this **MQSCX** version is primarily to reflect the new IBM MQ release.

1 Introduction

MQEV is a program which will receive, store and process three types of IBM MQ messages.

1. **MQ Event Messages**
   Event messages are IBM MQ's way of telling the installation that something 'of note' has happened. This could range from something fairly innocuous such as channel starting to something fairly serious such as a queue filling up.
2. **MQ Statistics Messages**
   Statistics messages are messages which the MQ Queue Manager will generate on a regular interval to notify the user of the levels of activity. For example, how many messages have been put or got to a queue.
3. **MQ Accounting Messages**
   Accounting messages are fairly similar to statistics messages however they are from the point of view of the application. So, it gives information such has how many MQI calls, and which type, have been issued by each application in the system. Again these messages output at regular intervals.

These three types of messages contain very different message content however they are all, to some extent, reporting on 'events' within the MQ Queue Manager. This manual may refer to them all collectively therefore as just 'event' messages coming from 'event' queues. In cases where we are particularly only talking about actual MQ Event messages then this will be made clear. In general though the mechanisms and features provided by **MQEV** apply to all three types of messages.

Notifying the user of activities within the MQ system via an MQ Message is clearly a natural fit for the MQ environment. The Queue Manager puts these messages to a selection of well known queues such as:

- **SYSTEM.ADMIN.COMMAND.EVENT**
- **SYSTEM.ADMIN.STATISTICS.QUEUE**
- **SYSTEM.ADMIN.ACCOUNTING.QUEUE**
- **SYSTEM.ADMIN.CONFIG.EVENT**
- **etc...**

By changing these queue definitions these messages can, if required, be routed round the MQ network or even even published to multiple recipients. However, these messages are not without their disadvantages; these disadvantages include:

- **No built-in tools**
  Perhaps the major disadvantage of these messages is that IBM MQ does not provide the user with any way to process or respond to these messages. This is means that in many MQ installations these messages are either ignored or their generation is switched off completely. This is a huge waste of potentially valuable information. All three types of messages give important information which can give clues as to the health of your MQ environment as well providing critical information should things go wrong.
- **Readability**
  All of these messages are in a message format known as Programmable Command Format (PCF). PCF messages are messages which are formatted in a way that makes them fairly easy and efficient to process in a program but are not very human readable. Of course there are a number of programs out there which will format these message such as our **MO71** program. However, just looking at the messages on a queue and trying to find a particular field within that message is very time consuming, laborious and error prone.
- **Completeness**
  A common event which an Administrator might be interested in is when an object changed and what was changed. We are all familiar with the feeling that when something goes wrong you want to know “what changed, and who did it ?”. Well MQ tells you when something changes via a Configuration Event. Unfortunately it gives you two of them; a before image of the object configuration and an after image of the object configuration. So, to see what happened you have to compare the fields in two different messages and there may be dozens of fields. To make matters worse these two messages may not even be consecutive in the queue!

- **Searchability**
  Of course what you are often interested in is messages that belong to a particular object or group of objects. Even if you can read the messages it can be very time consuming to sift through the thousands of messages on your event queue looking for the ones that pertain to a particular object.
- **Message Size**
  The PCF message format is not very efficient when it comes to message size. String parameters are often padded with blanks to their maximum size. Booleans which can, by their very nature contain only two possible values, are stored in structures that take 16 bytes.
- **Expiration**
  These messages will not expire. This is good from the point of view that your queues contains a useful history of what happened. However, sooner or later an event becomes uninteresting. Is it really useful to know that a channel started or stopped six months ago? Worse still is that when your queue fills up with these old, uninteresting messages it stops any new event messages being written. What is needed is a way to age out event messages.
- **Event Storms**
  It is possible for a queue manager to get a storm of events. Consider, for example, a rogue application in an infinite loop trying (unsuccessfully) to connect to a queue manager. This can generate thousand of 'not authorized' events in a very short space of time. This means that your event queue can fill, preventing any other events from being raised, in just a few seconds.
- **Actionability**
  Of course in many instances what you actually want is 'an action' to automatically happen when an event arrives. The action performed could be anything such as sending an email, issuing an MQ command, raising an alert, issuing an OS command or writing to a log file. Equally whether you want the action performed will depend on many factors such as event type, object name, frequency, time of day. This is not possible to do with just standard MQ facilities.

What **MQEV** does is greatly alleviate these disadvantages. **MQEV** provides:

- An easy way to store a history of events, statistics and accounting information[1]
- Simple configuration of how long the data should be stored for
- Fast search and display of all stored data
- Consolidation and summarising of change events
- Fast totalling of statistics and accounting data
- Ability to easily write scripts to process and action events as they are issued
- Ability to easily write scripts to mine the events and statistics data
- Administration and display of MQEV data either by command line or a GUI
  - Command line provided by **MQSCX**[2]
    The version of **MQSCX** should be V9.1.0 or later.
  - GUI interface integrated into **MO71**[3]
    The version of **MO71** should be V9.1.4 or later.

1 Without the complication and expense of a database
2 No separate MQSCX licence is required. An MQEV licence is sufficient to issue MQEV MQSCX commands.
3 No separate MO71 licence is required. An MQEV licence is sufficient to issue MQEV commands from MO71.

1.1 Uses

Most people seem to instantly recognise the advantages of storing and processing events but the statistics and accounting messages are a little less clear. Many people think that these messages would only really be useful for companies where internal departments charge depending on MQ resource usage. Well, clearly this is one use case but what else can we do with these messages? What questions might we be able to answer if we had a history of such messages ?

Well, have you ever asked yourself any of the following questions:

- Did anything unusual happen with my Queue Manager over the weekend ?
- Is queue or channel XYZ still being used ? ...and, if so, by who/what ?
- What type of MQ clients are using my Queue Manager ? JMS ? C ? Java ? C++ ?
- What version of MQ client are connecting to my Queue Manager ? Are any of the clients old and need updating ?
- What channels are being used and what is the relative split of persistent message traffic ?
- How efficient are my channels ? What is the average batch size in use over time ?
- Are my channel exits efficient ? What time is being spent in them ?
- How is my network performing ? How does my network round-trip time vary over time ?
- How does the activity/depth of queue X vary over the day ? What are my peak traffic times ?
- How does the activity/depth of queue X vary over the last 3 months ? Am I seeing an increase of traffic ? Will my MQ infrastructure be able to cope with future demands ?
- Are my channels having to ever retry their puts to deliver messages ?
- Are any of my applications misbehaving ? Do we see excessive amounts of failed MQI calls such as MQGETs, short connections, or are they opening/closing the queue for each message ?
- How is my processing speed varying over time ? How long are messages languishing on queues ? Am I delivering on my Service Level Agreements (SLA) ?
- Are my applications publishing to topics with no subscribers ?
- What is the maximum depth my queue has reached during a particular period ?

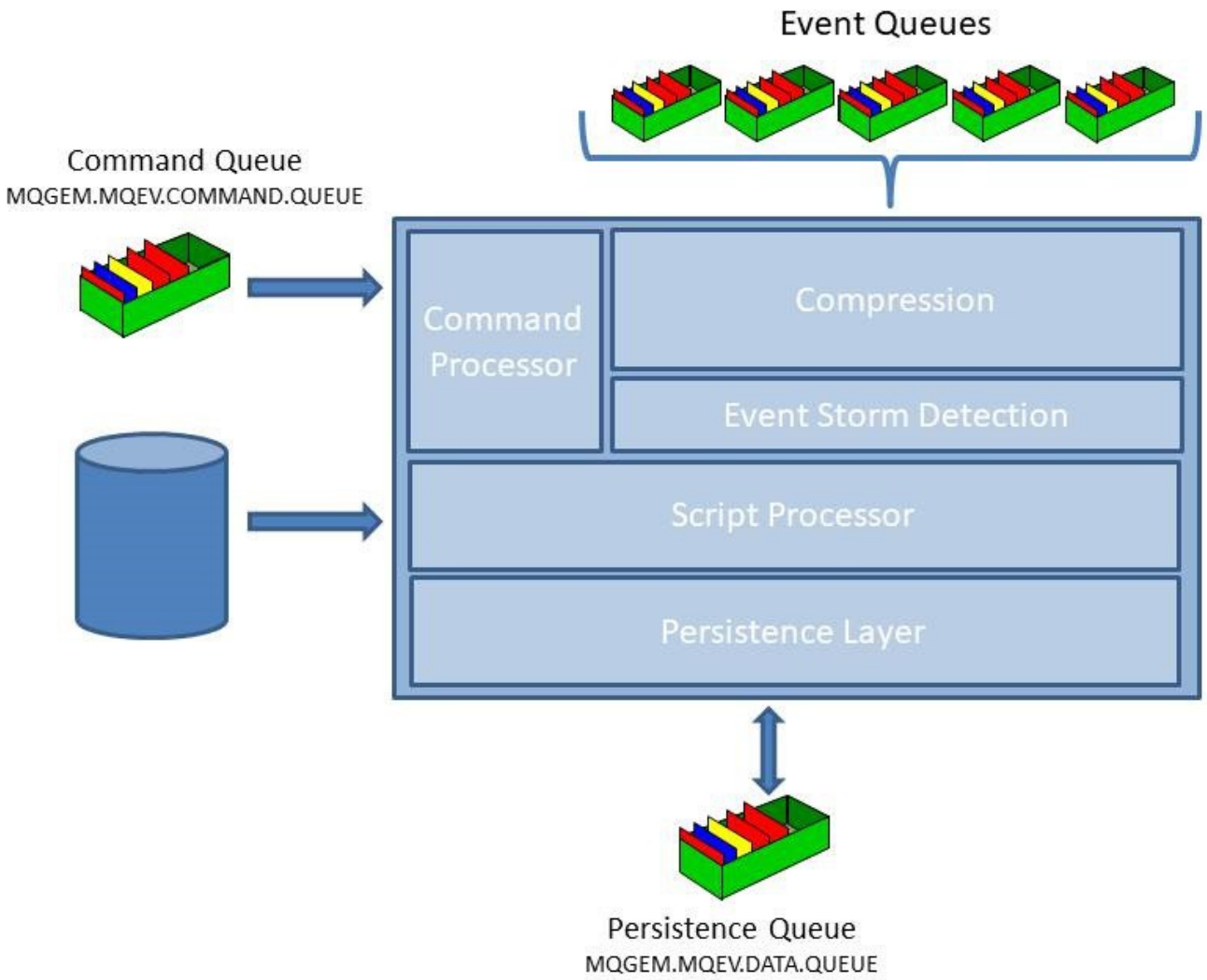
Of course there are many more questions which can be answered. For example, many installations who do monitor the accounting and statistics messages use it for capacity planning. Being able to see the current utilisation of MQ resources and see trends over time can prevent nasty surprises in the future.

In addition the information can be extremely useful to diagnose system problems. eg. Channels get backed up and you want to understand where this storm of messages came from.

Not only can you issue queries into MQEV to discover what happened during a particular period but you can actually monitor for situations real-time. Suppose you want to be told if the depth of a queue exceeds a particular value or the network response time drops or an application issues more than 100 failed MQI calls in any one interval. You can easily write a script to check for these conditions and raise some form of alert if they are detected. This alert could be in the form of an object which can be queried or you could actually send an email. In fact since you can invoke a program there is no limit to type of alert you can configure.

1.2 Concepts

Later on in this document we describe in a little more detail how **MQEV** works but for now we shall just consider the basic concepts. In essence **MQEV** is just an MQ application which consumes MQ event, statistics and accounting messages and writes them to a storage queue. However, naturally there is more to it than that. We could view **MQEV** pictorially like this:



It is recommended that wherever possible the Persistence Queue is on the same machine that **MQEV** itself is running. This is because, over time, the persistence queue could contain a fair amount of data and you do not wish this data to have to be fetched across a client link each time **MQEV** starts. However, that being said you can run **MQEV** in three modes.

1. All queues are local to the running instance of **MQEV**
2. The persistence queue is local to **MQEV** but the event queues are access across a client link
3. All queues are accessed across a client link

As we said before though, we recommend you run in one of the first two configurations.

If you are running in the second mode then **MQEV** will, necessarily, connect to two different Queue Managers. The Queue Manger you are monitoring and the Queue Manager where you choose to put the persistence queue. In such a case, the persistence Queue Manager is referred to as the State Queue Manager since it maintains the **MQEV** state. Ideally the state Queue Manager is dedicated to the purpose and does nothing other than store **MQEV** persistence data.

We'll now introduce the various aspects of the concept diagram.

1.2.1 Event Queues

MQEV can read from any number of event queues. By default it will read from the well known set of event queues, and accounting and statistics queues. These are:

- `SYSTEM.ADMIN.CHANNEL.EVENT`
- `SYSTEM.ADMIN.COMMAND.EVENT`
- `SYSTEM.ADMIN.CONFIG.EVENT`
- `SYSTEM.ADMIN.LOGGER.EVENT`[4]
- `SYSTEM.ADMIN.PERFM.EVENT`
- `SYSTEM.ADMIN.PUBSUB.EVENT`[4]
- `SYSTEM.ADMIN.QMGR.EVENT`
- `SYSTEM.ADMIN.ACCOUNTING.QUEUE`[4]
- `SYSTEM.ADMIN.STATISTICS.QUEUE`[4]

These are the various queues to which IBM MQ will write event, statistics and accounting messages. Note however that just because a queue exists does not necessarily mean that any event messages will be written to it. IBM MQ allows a fair amount of choice about which objects will generate events and what types of events. You need to ensure that the queue manager (and possibly the object definition) has enabled the event appropriately. In addition not all events are supported by all IBM MQ platforms. Most notably, sadly, IBM MQ for z/OS does not support Statistics and Accounting messages but instead uses SMF records.

You can change the list of event queues that are read by **MQEV**. For example you might prefer to merge all of your event queues into a single queue or route messages from other queue managers to another local queue. In these cases you would use commands such as **`ADD EVQ`** and **`REMOVE EVQ`** to alter the list of queues that **MQEV** will monitor. Full descriptions of all of the **MQEV** commands are in Chapter 11: Command Reference on page 50.

1.2.2 Command Queue

MQEV responds to various user commands. These commands are sent to **MQEV** via a command queue called `MQGEM.MQEV.COMMAND.QUEUE`. Commands can be in the following formats:

- MQSC
- Escape PCF (MQSC command wrapped in a PCF message)
- PCF

Please refer to Chapter 11: Command Reference on page 50 for a description of these commands.

An **MQEV** licence allows you to use **MQSCX** (Command Line) or **MO71** (GUI) to administer **MQEV**. You can also, of course, write your own tools to issue **MQEV** commands.

1.2.3 Persistence Layer

Clearly **MQEV** needs to store data to some persistent media. A database might seem the obvious choice but databases have significant disadvantages. They add complication, expertise requirements and often expense. For these reasons **MQEV** stores the data in a single queue as described below. This means that the data is kept within the MQ world, and can be consumed transactionally using just a local transaction with no need to employ a global transaction coordinator.

Other advantages include things such as HA and Disaster recovery. Having all the state on an MQ Queue in the same 'domain' as the event queues themselves means that all the event data can fail over together without the need to synchronise with a database.

Of course there are many factors related to persistence such as “How much data are we talking about?” and “What prevents this data store just getting bigger and bigger and bigger?”. These are discussed in Chapter 17 Data Management on page 194. However, for the moment, let's just content ourselves with the knowledge that data is persisted and it is all stored somehow in a single MQ queue.

4 These queues are not available on z/OS, and therefore are not added to MQEV by default on z/OS.

1.2.3.1 The Persistence Queue

Each instance of **MQEV** will store it's persistent data in a single persistence queue. It is not possible for two or more instances of **MQEV** to share the same persistence queue. Nor is it possible for a single instance of **MQEV** to use more than one queue. However, the name of the persistence queue used depends on how you configure **MQEV**. From a persistence point of view **MQEV** can run in two modes.

1. **You wish to have the persistence queue on the same Queue Manager as the event Queues**

   You could start such an **MQEV** instance with the command:
   `mqev -m QM1`

   This is the simplest way to run **MQEV**. Everything is on the same Queue Manager, `QM1`.

   In this case **MQEV** will use a queue called `MQGEM.MQEV.DATA.QUEUE`.

2. **You wish to have the persistence queue in a different 'State' Queue Manager**

   You could start such an **MQEV** instance with the command:
   `mqev -m QM1 -l -s MQEVSTATE`

   Note that we have added the **`-l`** parameter since it only really makes sense to use a different State Queue Manager if you are accessing your monitored Queue Manager over a client connection. Remember that it is not recommended that the persistence queue is accessed over a client link which is why we need the State Queue Manager in the first place.

   In this case **MQEV** will use a persistence queue which has the same name as the Queue Manager it is reading from the event queues, in this case `QM1`.

   This is why it is recommended that the `MQEVSTATE` Queue Manager is not used for anything else since you could not, for example, define a transmission queue for `QM1` because **MQEV** expects to be able to use this name.

1.2.4 Compression

MQ Event, Statistics and Accounting messages can be fairly large in relation to the data that they actually convey. This layer is responsible for a number of basic tasks.

1. Reducing the message size wherever possible. In IBM MQ PCF messages, strings are often blank padded to their maximum length. Numbers are stored as 32-bit integers even when the maximum set of values is less than 10. As a result, compression rates of 20-30% are quite normal.
2. Discarding certain messages. For example, you may decide that it is not worth saving general **`DISPLAY`** command events.
3. Merging some events. For example, IBM MQ sends a change object event as two event messages – a BEFORE and an AFTER image. **MQEV** will merge these two events into a single event message. This merge results in a storage saving, as well as making the event easier to process.
4. Discarding elements of no value. For example Statistics and Accounting messages contain lots of data saying what didn't happen. For example, there were no puts, no browses, no commits etc.
5. Discarding information about temporary queues. You can configure whether a stream should store information about temporary queues, usually just used as reply queues, or not.
6. Using a string dictionary
   The same strings occur time and time again but **MQEV** will only store the value once in a dictionary.

Clearly the combination of these tasks means that the exact content of the original event messages is not maintained. If you need to maintain all the event data 'as-is' for some reason then you can daisy chain other processes either before or after **MQEV**. You can configure **MQEV** to forward all event messages to another queue before they have been processed. See 18.4.1: Daisy chaining on page 197 for more information on this.

1.2.5 Event Storm Detection

Events Storms only apply to actual MQ Event messages. There can be times when IBM MQ will issue the same event many times in a short space of time. Usually this is because of some misbehaved application that is doing the 'wrong' thing over and over again. However, it can be something quite innocuous. Consider, the example a Queue Manager which has 10,000 clients connected to it that then loses the network. MQ will generate thousands of events within the space of a few seconds. One must ask oneself whether it really is worth storing each notification when they are all almost identical.

MQEV gives you the choice and allows you to decide what constitutes a 'storm'. By default a storm is receiving more than 20 identical events in a single minute. However, you could decide that it ought to be more or less. For more about event storms, see Chapter 13: Event Storms on page 158.

1.2.6 Script Processor

One of the powerful features of **MQEV** is that it allows you to write simple script functions which are invoked at certain key moments of **MQEV**. For example, each time **MQEV** receives an event message it will call a function called `MQEVEvent()`. One could choose to do nothing in this function or you can check certain values of the event and the issue some action. The list of possibilities is literally endless. **MQEV** uses the same language as used by **MQSCX** so any users of that product will find everything very familiar. For more about this scripting, and the various functions that are called by **MQEV**, see Chapter 14: MQEV Scripting on page 159.

1.3 Feedback

We are always interested in hearing user views, whether good or bad, and would love to hear your opinions, comments and suggestions. So, if you would like to make a comment either about **MQEV** itself or this manual then please do contact us at support@mqgem.com.

So, without further delay let’s see how to get up and running with **MQEV**.

2 Licensing

To be able to run **MQEV** you will need a valid licence. This can either be a purchased licence or a free trial licence. If you would like to try out **MQEV** for free then a 1-month trial licence can be obtained by sending an email to support@mqgem.com.

Each licence is for a certain period of time, usually one year rather than for a particular version of **MQEV**. There are number of advantages of this scheme, the two main ones being:

- **Purchasing decision is simpler**
  The **MQEV** licence covers a period of time not a release. It is therefore not necessary to concern oneself about whether a bigger, better version is about to come out soon since whatever licence you buy now will also work for that version. You can always run the latest version with the latest set of features.
- **Features are available sooner**
  Using this model it is not necessary for us to collect a large group of features together to 'justify' a new release of **MQEV**. Instead a new release can be made available whenever a new feature is added which is regarded as sufficiently useful since all current users will be able to migrate to the new version at no cost to themselves.

As has been mentioned before you can issue commands to **MQEV** from either **MQSCX** or **MO71**. These products will therefore also recognise your **MQEV** licence as valid and allow you to use them for that purpose. Note that although **MQSCX** would allow you to issue **MQEV** commands you could not issue normal MQ commands without also having an **MQSCX** licence. Similarly with **MO71** it will only allow **MQEV** based operations without a valid **MO71** licence. Other MQGem products allow for individual, single user, licences. However, this is not really applicable to **MQEV** since the program is used on a queue manager wide basis. Therefore for **MQEV** there are just two types of licences on distributed platforms, and three types on z/OS.

Type	Fields Set	Description
Ruby	machine	**MQEV** is supported by any number of users on a single z/OS machine. This licence type is not applicable for **MQEV** on distributed platforms.
Diamond	location	**MQEV** is supported by any number of users at the same site on any set of machines. The location field gives the location, for example “London, England” of where the licence is based.
Enterprise	None	**MQEV** is supported across your whole enterprise. This means any number of users at any number of locations. An Enterprise licence can be bought by purchasing just 3 Diamond licences.

2.1 Licence File Location

If a licence file is bought you will be sent an ***mqgem.lic*** file. All you need to do is place this licence file in the appropriate place for the **MQEV** program to find it as detailed in the table.

Platform	Location
Windows and Linux	Same directory as the MQEV program.
AIX and z/OS UNIX	Current directory
z/OS	DD:MQGEML

Alternatively you can set environment variable **`MQGEML`** to point to the directory path where the licence file can be found (in which case the name will be assumed to be ***mqgem.lic***), or MVS file or DD name of the licence file. For example, if you use the program in all of TSO, z/OS UNIX and from JCL, you can have one copy of the licence file saved either as a z/OS UNIX file or in an MVS dataset, and refer to it from any environment.

2.1.1 When running MQEV in z/OS UNIX

To refer to the licence file which is stored in an MVS dataset, set the **MQGEML** environment variable using commands like the following:

```
export MQGEML=//'GEMUSER.USER.LIC(MQGEM)'
mqev -m MQG1
```

2.1.2 When running MQEV interactively in TSO

You can allocate a DD name for use in TSO, using the TSO ALLOCATE command. For more information about this command, refer to z/OS TSO/E Command Reference > ALLOCATE command and Opening files > DDnames.

To refer to the licence file which is stored in an MVS dataset, using a DD name, use commands like the following.

```
ALLOCATE DDNAME(MQGEML) DSNAME('GEMUSER.USER.LIC(MQGEM)') SHR
mqev -m MQG1
```

To refer to the licence file which is stored in a z/OS UNIX file, using a DD name, use commands like the following.

```
ALLOCATE DDNAME(MQGEML) PATH('/u/gemuser/licences') PATHOPTS(ORDONLY)
mqev -m MQG1
```

2.1.3 When running MQEV from JCL

To refer to the licence file which is stored in an MVS dataset, configure your JCL as follows.

```
//MQEV     EXEC PGM=MQEV,PARM=('-m MQG1')
//MQGEML   DD   DSN=GEMUSER.USER.LIC(MQGEM),DISP=SHR
```

To refer to the licence file which is stored in a z/OS UNIX file, configure your JCL as follows.

```
//MQEV     EXEC PGM=MQEV,PARM=('-m MQG1')
//MQGEML   DD   PATH='/u/gemuser/licences',PATHOPTS=(ORDONLY)
```

2.2 Multiple licences

If you have multiple licences then they can be concatenated into a single ***mqgem.lic*** file. This can be done using simple OS commands such as **copy** or by using your favourite editor. Ensure that all lines are copied in their entirety, including the carriage return character at the end. Blank lines can be added to the file as required.

2.3 Licence Renewal

Licence renewal is never automatic. **MQEV** will start issuing warnings as you get close to the licence expiry date. These warnings will be in the form of alerts (see Chapter 12: Alerts on page 153 for more about alerts). The priority of the alert will increase as the expiry date gets closer.

By default **MQEV** will start to remind the user about an expiring licence some 60 days before it expires. However, you can adjust this interval, if you like, by altering the EV object.

A new licence can be purchased at any time and a new licence file will be sent which extends the current licence by whatever period has been purchased. There is therefore no concern about losing time by renewing early.

When you get a new licence just replace your current ***mqgem.lic*** file and within the next 24 hour period **MQEV** will notice and any outstanding alerts will be removed.

2.4 Changing your licence file

The licence file is a simple text file. Generally speaking if you change the contents of the licence file you will invalidate it and it will cease to work. However, there are some minor changes you can make if you wish. Naturally it is always recommended that you keep a copy of the original unchanged file.

- You can change the case of any of the values.
- You can add or remove white space such as blanks
- You can add or change any lines which start with '*' since these are comment lines.

Remember that more than one licence can be contained within the single ***mqgem.lic*** file if required.

3 Getting Started

This chapter will introduce you to the basics of **MQEV** and allow you to get up and running quickly. However, it is strongly recommended that you read the entire manual before deciding how to use the product in a production environment.

3.1 Installation

If **MQEV** has already been installed you can skip this section.

The examples in this manual assume that the program executable is in the path. The examples and screen-shots are from a Windows system but very similar screens will be seen in Unix. The code should run correctly on most versions of Windows and Unix however it is always strongly recommended that you 'try before you buy'. In other words, request a trial licence, play with the program and ensure that it meets you needs. If all is well then you can go ahead with the purchase. If you encounter problems then please contact support@mqgem.com and we'll see if we can help.

MQEV is provided as a zip, tar or gzip file depending on the platform. In addition to the executable file, whose name will vary by platform, the following addition text files are also provided on all platforms:

mqev.mqx	Sample minimum script file (see 14 MQEV Scripting on page 159)
mqconfig.mqx	Script file to define the two queues needed by **MQEV** (see 3.3.1 Queues on page 15)
TestEventMsgs.qld	a **QLOAD** file with a set of example event messages (see 3.7 Testing with other events on page 29 for more information about this)
mqev.h	the **MQEV** header file for writing your own program to send commands to **MQEV** if you choose not to use either **MO71** or **MQSCX** (see 11.1 Programmable command format commands and responses on page 51 for more on this)

3.1.1 Windows

MQEV is provided as a simple zip file. Once you have downloaded this file you should unzip it into a location which is either in your path or can be explicitly referenced on the command line.

3.1.2 Unix

MQEV is provided as a simple tar or gzip file. Once you have downloaded this file you should untar the file using one of the following commands depending on the file type (note that the file name depends on the platform).

```
tar -xvzf mqev.tgz

OR

gzip -d mqev.tar.gz             (If file is a gz file)
tar -xvf mqev.tar
```

Move the file to a directory which is either in your path or can be explicitly referenced on the command line.

3.1.3 z/OS

MQEV is provided as a zip file. Once you have downloaded this file you should unzip before following these instructions.

In addition to the files detailed above that are on every platform, the zip file also contains the following:-

MQEV.SEQ	sequential file containing the **MQEV** program
MQEV.JCL	example JCL for running the program in batch (see 9.2 Running MQEV in batch (z/OS only) on page 46 for more about this)
MQEVSTC.JCL	example JCL for running the program as a started task (see 9.3 Running MQEV as a Started Task (z/OS only) on page 47 for more information about this)

Once unzipped, transfer the ***MQEV.SEQ*** file to a z/OS system using the following commands.

```
ftp> binary
ftp> quote site recfm=FB lrecl=80 blksize=3120 blocks primary=1000
ftp> put MQEV.SEQ
```

Once the ***MQEV.SEQ*** file is successfully FTPed to your z/OS system, from TSO use the following command:

```
receive inds(MQEV.SEQ)
```

When prompted for a filename, reply

```
DSN(USER.LOAD)
```

The other files in the zip are text files and can be transferred using **`ascii`** mode in ftp if you plan to use them.

MQEV can be run on z/OS in BATCH (including as a Started Task). Example pieces of JCL are provided in the zip file as noted above. **MQEV** can also be run interactively, e.g. from the TSO/E READY prompt, or the ISPF Command Shell (=6). It can also be run in z/OS UNIX (see below).

3.1.3.1 z/OS Unix Installation

If you wish to run **MQEV** in z/OS UNIX, you can copy the MVS executable module that you have installed in the previous section, to a directory in z/OS UNIX with the following command.

```
TSO OPUT 'GEMUSER.USER.LOAD(MQEV)' '/u/gemuser/bin/mqev' BIN
```

3.1.4 MQEV Administration

The **MQEV** installation contains the **MQEV** program which will collect and store MQ events however it doesn't include the means to administer the program. For that you need to also install either our command line tool **MQSCX**[5] or our GUI Administration tool **MO71**[6] (or indeed both if you wish).

Download the respective install images for these products and use the **MQEV** licence file in place of the normal product licence file.

If you are already an **MQSCX** or **MO71** user, and have licences for these products, we recommend concatenating the licences into a single licence file (as described in 2.2: Multiple licences on page 10) so that these tools will find both licences and allow you to administer IBM MQ and **MQEV** from the same instance of the tool.

5 At least version 9.1.0
6 At least version 9.1.4

3.2 Upgrade

To upgrade **MQEV** to a new version, we always recommend that you take a backup of the previous version of the executable and of the data behind it before running the new version.

To backup the data that **MQEV** uses, stop the **MQEV** program from running, and then use one of the following commands, depending on the tools you have available on your system.

Ensure the file name provided is unique.

```
dmpmqmsg -m MQG1 -i MQGEM.MQEV.DATA.QUEUE -f C:\MQGem\MQEV\Backup210820.qld
```

You can get **qload** to generate a file name for you with, say the current date in it, by using the **%c** file name insert. This will help when ensuring that the file name is unique.

```
qload -m MQG1 -i MQGEM.MQEV.DATA.QUEUE -f C:\MQGem\MQEV\Backup%c.qld
```

If you have to reinstate the backup of the data, this of course implies that you will take a step back in time with the events that you have captured. If you wish to save off the event messages so that they can be processed again after a backup is reinstated, you can make use of the **FWDQ** attribute to save a copy of each event to a side queue. Read more about this attribute and also **FWDPSIST** to ensure you save the messages appropriately, in **ALTER EVQ** on page 65.

3.3 Configuration

MQEV is essentially infinitely configurable since it is capable of running scripts in response to various activities in the queue manager. However, in the interests of learning to walk before we run let's just consider the minimum configuration one needs to get the program running. It is recommended that for your first experience of **MQEV** you create a test local queue manager that you can 'play' with until you are comfortable with how **MQEV** operates and how it can be configured. For the examples below we'll assume that your test queue manager is called `MQG1` but if you name yours differently then just replace instances of `MQG1` with the name you have chosen.

3.3.1 Queues

We have seen that in order to operate **MQEV** requires two queues; a command queue and a data queue. These queues are just normal local queues and have fixed names. A file ***config.mqx*** is included in the install zip file which contains the commands for defining these queues. Feel free to change these definitions to suit your local standards however there are certain minimums which should be observed for correct program operation.

`MQGEM.MQEV.COMMAND.QUEUE`	
`MAXMSGL`	**MQEV** commands are not very long. The value must be at least 10,000 but it is recommended you keep the normal default of at least 4,194,304.
`MAXDEPTH`	The depth of the command queue should be sufficient to handle as many concurrent requests at you might ever issue. In general this will probably not be very many however it is recommended that you keep the default of at least 5,000.
Triggering	It is not recommended that triggering is enabled. **MQEV** is designed to be started at the dawn of time and retry connections at certain intervals.

`MQGEM.MQEV.DATA.QUEUE` or `<QM Name>` **if you are using a State Queue Manager**	
`MAXMSGL`	**MQEV** currently stores event data in chunks of 1 MB so a value of 2MB will work. However, it recommended that keep the normal default of at least 4,194,304
`MAXDEPTH`	This value depends to some extent on how much event data you want **MQEV** to store. A rough guide would say that you need 1 message per 3000 events. So, suppose your Queue Manager generates 5,000 events per day and you wish to keep those events for six months. This would mean **MQEV** should store 900,000 events. This would require a queue `MAXDEPTH` of at least 300. Clearly this is just a rough calculation so it is recommended that you greatly over specify your queue definition to avoid storage problems, It is recommended that you keep the normal default of at least 5,000.
`SHARE`	Ensure that the queue is shareable.
Triggering	It is not recommended that triggering is enabled.

3.3.2 Script functions

Script functions allows the user to configure behaviours within **MQEV** when certain events occur either in IBM MQ or in the life of **MQEV**. It is not necessary to actually have any code at each function point but it is necessary to have the function. **MQEV** will refuse to run if it either can't find your script file or one or more functions are missing.

To make life easier we have provided a skeleton script file called ***mqev.mqx*** in the zip file. This file must be placed where **MQEV** can read it. The simplest approach to this is just have the ***mqev.mqx*** file in the same directory as the **MQEV** program. However, if for some reason you prefer to have it elsewhere then when you run the program you must tell **MQEV** where it is. This is done with the **`-f`** parameter, or alternatively on z/OS you can use the **`MQEVMQX`** DD name (see 9.2.2 DD name MQEVMQX on page 46).

So, for example you could run **MQEV** like this:

```
mqev -f c:\MQEV\script\mqev.mqx
```

In fact **MQEV** will assume the name of the file is ***mqev.mqx*** so the following will also work:

```
mqev -f c:\MQEV\script
```

If you choose to change the name of the script file then you must include it in the parameter value.

3.3.3 Events

Of course for **MQEV** to show us anything we also need IBM MQ to generate some events. Perhaps the simplest events to get the queue manager to generate are the command and configuration events. So, let's change our queue manager to generate these types of events. Issue a command such as the following to your queue manager:

```
ALTER QMGR CONFIGEV(ENABLED) CMDEV(NODISPLAY)
```

This will cause your queue manager to send event messages to the queues `SYSTEM.ADMIN.COMMAND.EVENT` and `SYSTEM.ADMIN.CONFIG.EVENT` whenever you issue an IBM MQ command (which is not a DISPLAY) to work with an MQ object, for example to create a queue. If you really wish to see all commands then you can set the value to `CMDEV(ENABLED)`. However, this is not normally recommended since display commands are both frequent and generally uninteresting.

3.3.4 Statistics

If you are interested instead in Statistics messages i.e. information which tells you which resources are being used in your Queue Manager and by how much then you would issue a command such as the following command:

```
ALTER QMGR STATQ(ON)
```

This will cause your queue manager to send statistics messages to the queue `SYSTEM.ADMIN.STATISTICS.QUEUE.`

IBM MQ is capable of generating statistics on Queue Usage, Channel Usage and MQI Usage and each one can be switched on independently. You can also change the interval that the messages are generated using the `STATINT` attribute. Note that setting a really short interval can dramatically increase the amount of data that is needed to be stored as well as increasing the burden of message traffic and processing. It is therefore recommended not to set too small a value for the interval in production systems.

3.3.5 Accounting

If you are interested instead in Accounting messages i.e. information which tells you who and what are doing things to your Queue Manager then you would issue a command such as the following command:

```
ALTER QMGR ACCTMQI(ON)
```

This will cause your queue manager to send accounting messages to the queue `SYSTEM.ADMIN.ACCOUTING.QUEUE.`

IBM MQ is capable of generating accounting information on Queues and MQI Usage and they can be switched on independently. You can also change the interval that the messages are generated using the `ACCTINT` attribute. Note that setting a really short interval can dramatically increase the amount of data that is needed to be stored as well as increasing the burden of message traffic and processing. It is therefore recommended not to set too small a value for the interval in production systems.

3.4 Running the program

Of course normally you would run **MQEV** as a background task, perhaps started by an automatic scheduler, or some of the facilities discussed in Chapter 9 Running MQEV with your Queue Manager on page 45. However, in our first attempts it is far easier to see what is going on if we just run it in a command window.

So, bring up a command window (that has the correct MQ environment) and issue the following command:

```
mqev -m MQG1
```

If all goes well you should see a screen that looks something like this:

```
MQEV Version:9.1.0 (64 Bit) Build Date:Nov  21 2019

Log Root Directory:c:\MQGem
MQEV Initialising...
Loading MQSCX Script...
MQSCX Script active
Connecting to 'MQG1'
Connect 'MQG1' OK
Connected to 'MQG1'
09:04:10 MQEV Qm(CONNECTED) Events(0) Stats(0) Cmds(0) Alerts(0)
```

If you don't see a screen like this then there should be an error message telling what the problem is. Go back to the previous sections of this manual and double check your configuration. Ensure, for example, that you have created the command and persistence queues.

As we said before **MQEV** is designed to be a background process on your system so the output display is very rudimentary but in familiarising yourself with the product it can be useful. As you can see **MQEV** goes through a number of steps to initialise itself. The most important of these is to connect to the queue manager and to load and parse the event scripts. It will then go to the data queue and load its configuration. If it doesn't find any messages on the data queue (first time in) it will use a set of default configuration options which will read the standard IBM MQ event queues. This is perfect for our purposes.

When **MQEV** is running it will write various error and informational messages to the screen. It will also write out a very simple status summary. This can be useful as a quick glance of what the current state of the program is.

The last line will always contain the following:

- Current Time
- Name of this instance of **MQEV**
- The state of the MQ connection, whether it is connected, retrying etc
- How many Events the program has processed
- How many Accounting and Statistics messages the program has processed
- How many Commands the program has processed
- How many Alerts are currently outstanding

As the program runs and it processes messages and commands you should see these status numbers change according.

From this window you can also tell **MQEV** to end by entering 'Q' and then entering 'y' to the confirmation question.

The only other command you can enter at this point is '!' which will put **MQEV** into Debug mode. For more information about this please see Chapter 16 Debugging on page 187.

Those of you will a sharp mind may already be wondering about what happens if the **MQEV** program is running in the background – clearly you won't be able to see these status messages. How do you know what is going on. Well, there are commands you can issue to **MQEV** to interrogate it's status. **MQEV** will also write out status messages to a log file. Please see Chapter 7 Logging on page 41 for more about this.

Now that the **MQEV** program is running, you will want to interact with it. The way you do that depends on whether you want to use a command line program or to view things in a GUI. We'll cover both below. Even if you are planning to use just the GUI to control **MQEV** we suggest you read the **MQSCX** section below since it introduces a number of **MQEV** concepts.

3.5 Displaying MQEV in a command line (MQSCX)

Hopefully, as discussed before, you have installed the latest version of **MQSCX** in your path and given it access to your **MQEV** licence. Assuming that is the case then we can just start an instance of **MQSCX** with the following command:

```
mqscx -m MQG1
```

Now, when **MQSCX** starts it will be expecting to issue IBM MQ commands. However, we want **MQSCX** to send it's commands to **MQEV**. The way we tell **MQSCX** to do that is to issue the following command:

```
=mqev
```

What this does internally of course is to tell **MQSCX** to open and direct all subsequent commands to `MQGEM.MQEV.COMMAND.QUEUE`. Of course in order to do this you must be authorised to perform these actions. However, since you created the queue manager and queue we'll assume you are.

This command is not necessary if you only have an **MQEV** licence and not an **MQSCX** licence. In such cases **MQSCX** will start initially in **MQEV** mode. By default the **MQSCX** prompt will tell you whether you are in **MQEV** mode by displaying something such as:

```
MQEV:MQG1>
```

Now let's look at what queues **MQEV** is set to process by issuing the following command:

```
DISPLAY EVQ(*)
```

We will get a response something like the following:

```
EVQ(SYSTEM.ADMIN.ACCOUNTING.QUEUE)      MSGS(0)          STATUS(SUSPENDED)
EVQ(SYSTEM.ADMIN.CHANNEL.EVENT)         MSGS(0)          STATUS(SUSPENDED)
EVQ(SYSTEM.ADMIN.COMMAND.EVENT)         MSGS(0)          STATUS(SUSPENDED)
EVQ(SYSTEM.ADMIN.CONFIG.EVENT)          MSGS(0)          STATUS(SUSPENDED)
EVQ(SYSTEM.ADMIN.LOGGER.EVENT)          MSGS(0)          STATUS(SUSPENDED)
EVQ(SYSTEM.ADMIN.PERFM.EVENT)           MSGS(0)          STATUS(SUSPENDED)
EVQ(SYSTEM.ADMIN.PUBSUB.EVENT)          MSGS(0)          STATUS(SUSPENDED)
EVQ(SYSTEM.ADMIN.QMGR.EVENT)            MSGS(0)          STATUS(SUSPENDED)
EVQ(SYSTEM.ADMIN.STATISTICS.QUEUE)      MSGS(0)          STATUS(SUSPENDED)
Total display responses - Received:9
```

If this is not the first time you have run **MQEV** then you may find that some of the queues are already `ACTIVE` however initially, when they are first defined, they come up in `SUSPENDED` state to allow you to ensure that the configuration is as you want it before **MQEV** starts processing messages. In this instance we are happy to accept the default configuration so let's go ahead and tell **MQEV** to process some of the queues.

Let's issue the following commands:

```
RESUME EVQ(SYSTEM.ADMIN.COMMAND.EVENT)
RESUME EVQ(SYSTEM.ADMIN.CONFIG.EVENT)
```

If we issue the display again we should see something like this. As you can see **MQEV** has now processed both a command event and two configuration events. We'll see in a moment that these are from turning events on.

```
EVQ(SYSTEM.ADMIN.ACCOUNTING.QUEUE)      MSGS(0)             STATUS(SUSPENDED)
EVQ(SYSTEM.ADMIN.CHANNEL.EVENT)         MSGS(0)             STATUS(SUSPENDED)
EVQ(SYSTEM.ADMIN.COMMAND.EVENT)         MSGS(1)             STATUS(ACTIVE)
EVQ(SYSTEM.ADMIN.CONFIG.EVENT)          MSGS(2)             STATUS(ACTIVE)
EVQ(SYSTEM.ADMIN.LOGGER.EVENT)          MSGS(0)             STATUS(SUSPENDED)
EVQ(SYSTEM.ADMIN.PERFM.EVENT)           MSGS(0)             STATUS(SUSPENDED)
EVQ(SYSTEM.ADMIN.PUBSUB.EVENT)          MSGS(0)             STATUS(SUSPENDED)
EVQ(SYSTEM.ADMIN.QMGR.EVENT)            MSGS(0)             STATUS(SUSPENDED)
EVQ(SYSTEM.ADMIN.STATISTICS.QUEUE)      MSGS(0)             STATUS(SUSPENDED)
Total display responses - Received:9
```

MQEV will remember that you want to process these queues and next time you start **MQEV** or if it needs to reconnect for some reason it will immediately start processing messages from the active queues.

Now, let's issue a another command of some sort that will result in an event. You can choose any command you like but we'll assume we're defining a queue. Issue the following:

```
DEFINE QLOCAL(MQEV.TEST.QUEUE)
```

It depends on your queue manager settings but after issuing this command you may see the Events`()` value increment by two. So, why two ? Well, assuming that you have both command and configuration events enabled, IBM MQ will send you one of each event. One to tell you that a `DEFINE QLOCAL` command was issued and one to tell you that the configuration has changed and that a new object has been created.

What we want to do now is to be able to look at the events that were generated by IBM MQ.

So, how do we ask **MQEV** to display us these events. Well it is very simple, issue the following command:

```
DISPLAY EVENTS(*)
```

This will return something that looks a little like this:

```
EVQMGR(MQG1)        EVTIME(2019-10-09 22:15:26 (Local))
SUMMARY(Config - Create Object - Queue:MQEV.TEST.QUEUE)
____________________________________________________________________________
EVQMGR(MQG1)        EVTIME(2019-10-09 22:15:26 (Local))
SUMMARY(Command - Create Queue - Queue:MQEV.TEST.QUEUE)
____________________________________________________________________________
EVQMGR(MQG1)        EVTIME(2019-10-09 22:14:51 (Local))
SUMMARY(Config - Change Object - Qmgr:MQG1 - CONFIGEV[DISABLED -> ENABLED] CMDEV[
DISABLED -> ENABLED])
____________________________________________________________________________
EVQMGR(MQG1)        EVTIME(2019-10-09 22:14:51 (Local))
SUMMARY(Command - Change Qmgr - Qmgr:MQG1)
____________________________________________________________________________
Total display responses - Received:4
```

The first question that may spring to mind is “where are all the other fields?”. We all know that a queue definition has a lot of fields so why is the output so short ? Well, the answer is that **MQEV** allows you to choose how much data you get back. By default you will only get back a summary of information such as the Queue Manager it happened on, the time it happened and summary text of what happened in a field called `SUMMARY`. This simple field can be very useful in getting a quick feel for the event rather than trying to assimilate lots of separate values.

However, let's assume that we are curious type and we want to see the whole event, how would we do that ? Well change your command to the following (don't worry I'll explain **MAXRESP** in a moment!):

```
DISPLAY EVENTS(*) DISTYPE(DETAIL) MAXRESP(2)
```

You will see that now we get some additional fields. We get told the object name, object type and userid that was most associated with the event and the event reason code. We also get told the `EVENTID` which we'll talk about later.

```
EVQMGR(MQG1)          EVENTS($EVENTS)      EVTIME(2019-10-09 22:15:26 (Local))
EVREASON(CFGCRTOBJ)   EVUSERID(mqgemusr)   EVOBJNAME(MQEV.TEST.QUEUE)
EVOBJTYPE(QUEUE)      EVENTID(00000004)
SUMMARY(Config - Create Object - Queue:MQEV.TEST.QUEUE)
____________________________________________________________________________
EVQMGR(MQG1  )        EVENTS($EVENTS)      EVTIME(2019-10-09 22:15:26 (Local))
EVREASON(CMDCRTQ)     EVUSERID(mqgemusr)   EVOBJNAME(MQEV.TEST.QUEUE)
EVOBJTYPE(QUEUE)      EVENTID(00000003)
SUMMARY(Command - Create Queue - Queue:MQEV.TEST.QUEUE)
____________________________________________________________________________
Total display responses - Received:2
```

As I am sure you realise we are still not seeing the whole event. If we really needed to see everything then we could issue the following:

```
DISPLAY EVENTS(*) ALL MAXRESP(2)
```

...and we would see output such as this:

```
EVQMGR(MQG1)          EVENTS($EVENTS)      EVTIME(2019-10-09 22:15:26 (Local))
EVREASON(CFGCRTOBJ)   EVTYPE(CONFIG)       EVUSERID(mqgemusr)
EVOBJNAME(MQEV.TEST.QUEUE)                 EVOBJTYPE(QUEUE)     EVENTID(00000004)
CFHCMD(43)            CFHREASON(2367)
SUMMARY(Config - Create Object - Queue:MQEV.TEST.QUEUE)      EVENTUSER(mqgemusr)
EVSID(1D01010500000000000515000000E9B1B2A2AA735C01F86AE6C9EA03000000000000000000
000000)
EVORIGIN(MSG)
EVACCTTK(16010515000000E9B1B2A2AA735C01F86AE6C9EA03000000000000000000000B)
EVAPPLID( )           EVAPPLTYPE(WINDOWSNT)
EVAPPLNAME(d:\nttools\mqscx.exe)           EVAPPLORIG( )        OBJTYPE(QUEUE)
QUEUE(MQEV.TEST.QUEUE)                     DESCR( )             PROCESS( )
BOQNAME( )            INITQ( )             TRIGDATA( )          CLUSCHL( )
CUSTOM( )             CLUSTER( )           CLUSNL( )            CRDATE(2019-10-04)
CRTIME(22.52.43)      ALTDATE(2019-10-04)  ALTTIME(22.52.43)    GET(ENABLED)
PUT(ENABLED)          DEFPRTY(0)           DEFPSIST(NO)         MAXDEPTH(5000)
MAXMSGL(4194304)      BOTHRESH(0)          SHARE(1)             DEFSOPT(SHARED)
HARDENBO(1)           MSGDLVSQ(PRIORITY)   RETINTVL(999999999) USAGE(NORMAL)
TRIGCTL(NOTRIGGER)    TRIGTYPE(FIRST)      TRIGDEPTH(1)         TRIGMPRI(0)
QDEPTHHI(80)          QDEPTHLO(20)         QDPMAXEV(ENABLED)    QDPHIEV(DISABLED)
QDPLOEV(DISABLED)     QSVCINT(999999999)   QSVCIEV(DISABLED)    DISTL(NO)
NPMCLASS(NORMAL)      STATQ(QMGR)          ACCTQ(QMGR)          MONQ(QMGR)
SCOPE(QMGR)           DEFBIND(OPEN)        CLWLRANK(0)          CLWLPRTY(0)
CLWLUSEQ(QMGR)        DEFPRESP(SYNC)       DEFREADA(NO)         PROPCTL(COMPAT)
IMGRCOVQ(QMGR)        DEFTYPE(PREDEFINED)  QTYPE(QLOCAL)
____________________________________________________________________________
```

```
EVQMGR(MQG1)          EVENTS($EVENTS)      EVTIME(2019-10-09 22:15:26 (Local))
EVREASON(CMDCRTQ)     EVTYPE(COMMAND)      EVUSERID(mqgemusr)
EVOBJNAME(MQEV.TEST.QUEUE)                 EVOBJTYPE(QUEUE)     COMMAND(Create Q)
EVENTID(00000003)     CFHCMD(99)           CFHREASON(2413)
SUMMARY(Command - Create Queue - Queue:MQEV.TEST.QUEUE)
CMDCTX:
EVENTUSER(mqgemusr)
EVSID(1D01010500000000000515000000E9B1B2A2AA735C01F86AE6C9EA03000000000000000000
000000)
EVORIGIN(MSG)
EVACCTTK(16010515000000E9B1B2A2AA735C01F86AE6C9EA0300000000000000000000000B)
EVAPPLID( )           EVAPPLTYPE(WINDOWSNT)
EVAPPLNAME(d:\nttools\mqscx.exe)           EVAPPLORIG( )
CMDDATA:
QUEUE(MQEV.TEST.QUEUE)                     QTYPE(QLOCAL)
_____________________________________________________________________________
Total display responses - Received:2
```

Wow, that is a lot more stuff! We can see the full command that was issued and the full queue definition. We can even see some of the structure of the event message. For example a command event actually has three parts to it. A general set of fields which tend to be on most events. Then a group of fields known as the 'Command Context'. This is followed by a group of fields which is the 'Command Data' itself. This is shown in the **MQSCX** response as **`CMDCTX:`** and **`CMDDATA:`** respectively.

The **`DISTYPE()`** parameter has four possible values:

- **`SUMMARY`**
  This is just the default that we saw earlier.
- **`DETAIL`**
  This is the value to use if you want to see just the major identifying fields.
- **`MINIMUM`**
  Yes, believe it or not you can be even more terse than the **`SUMMARY`** display. **`MINIMUM`** tells **MQSCX** to return just the minimum of fields which are the Queue Manager and Event Reason. This is helpful if you want to add a specific set of fields to your output and not have some of the fields that **MQEV** normally returns by default.
- **`CONDENSE`**
  This is almost the same as `DETAIL`, all fields are returned, except for empty ones. **MQEV** will not return a field with no value.

Regardless of the display type you can always specify additional fields if you want those specifically returned in typical MQSC fashion. By specifying 'ALL' you are, of course, asking to see every field regardless of detail level.

So what else is this display telling us? Well the eagle eyed amongst you may have noticed the **`EVENTS($EVENTS)`** attribute. What does this mean ? Well, if you wish to know more about this them please refer to Chapter 5 Streams on page 33. For now all you really need to know is that **MQEV** does not necessarily store all of the events in the same bucket. You can choose to spread the events out according to any criteria you wish. By default though all events will be put in the `$EVENTS` bucket.

The next question that might be burning through your mind is what happens when **MQEV** has been running for, let's say, six months? We have already discussed that by then we might have 900,000 events! The last thing we want is to sit there waiting for 900,000 events to be sent to us. Well don't worry, that won't happen. At least not unless you want it to. **MQEV** has two ways of limiting the responses. Firstly there is just a standard count, called `MAXRESP`, which set the maximum number of responses you want. If not specified it will assume the default value of 100.

Go ahead, give it a try, Issue the command:

```
DISPLAY EVENTS(*) DISTYPE(DETAIL) MAXRESP(1)
```

You will see that **MQEV** responds with just a single response. So, the question then is which one does it choose ? Well, the only natural choice would be the latest event of course, **MQEV** always will return the latest events which match the display criteria in reverse chronological order. In other words the latest event will be the first one returned[7].

The second generic way that we can reduce the amount of data returned is to specify the time range of events you are interested in. By default **MQEV** will return events received in the last 24 hours. So, if you want to see events that were received older than that we need to add something to our command.

We can issue a command such as this :

```
DISPLAY EVENTS(*) FROM(8) TO(9)
```

Here we are saying that we want to see events created between 8am and 9am. We could say this another way:

```
DISPLAY EVENTS(*) FROM(8) TO(+1hour)
```

Both of these commands say the same thing. In one case we are using absolute times and in the second case we are using a relative time. We can even say this:

```
DISPLAY EVENTS(*) FROM(-1hour) TO(9)
```

We can also use the term now to refer to the current time.

```
DISPLAY EVENTS(*) FROM(-4hour) TO(now)
```

However, '**now**' is the default so this is the same as saying

```
DISPLAY EVENTS(*) FROM(-4hour)
```

which tells MQEV to return events generated in the last 4 hours.

Of course the absolute and relative time formats can be more specific. You can specify the date and minutes and seconds if you so wish. For more information please refer to **DISPLAY EVENTS** on page 105. Of course limiting the number of events in this way only partially solves the problem. If we have thousands of events we may still be looking for a needle in a haystack. So, how else can we limit the information returned?

7 Events are sorted internally in timestamp order. However, it is entirely possible that two events have the same timestamp in which case the order returned is the order that **MQEV** processed the message. If the events arrived on different queues then essentially it is random as to which event is 'seen' first by **MQEV.**

A simple trick is to issue a command such as:

```
DISPLAY EVENTS(*) ?
```

This command fails of course but it now tells us the command syntax.

```
AMQ8427: Valid syntax for the MQEV command:
   DISPLAY EVENTS( wildcarded_stream_name )
     [ EVQMGR( wildcarded_qmgr_name ) ]
     [ DISPCMDS( HIDE | SHOW ) ]
     [ DISTYPE( DETAIL | CONDENSE | SUMMARY | MINIMUM ) ]
     [ EVENTID( event_id ) ]
     [ EVOBJNAME( wildcarded_object_name ) ]
     [ EVOBJTYPE( object_type ) ]
     [ EVREASON( CONFIG | CFGCHGOBJ |.. ]
     [ EVTYPE( AUTHOR | CHANNEL | COMMAND | CONFIG | INHIBIT ... ]
     [ EVUSERID( wildcarded_userid ) ]
     [ FROM( absolute_time | relative_time ) ]
     [ TO( absolute_time | relative_time ) ]
     [ TZ( timezone_bias ) ]
     [ MAXRESP( number_responses ) ]
     [ WHERE( filter_expression ) ]
     [ { [ attribute ] ... } ] [ ALL ]
```

We can see that we can also limit the events return by Queue Manager, by Event Reason, Event Type, Userid, Object Name and Object Type. In the case of string fields then you can use wildcards where '*' represents 0 to any characters and '?' represents a single character. For example:

```
DISPLAY EVENTS(*) EVOBJNAME(*PROD*)
```

This would display any events which were related to any object which had 'PROD' somewhere in the object name. If you want to get even more specific then you can use the power of the **WHERE()** clause. Those of you familiar with IBM MQ will already know about the **WHERE()** clause however the **MQEV WHERE()** clause is a little different. The concept is much the same, it allows you to filter responses based on fields within the event. However, the standard MQ **WHERE()** clause is very limiting. For example it will only allow you to use a single attribute. The **MQEV WHERE()** clause is far less restrictive.

For example, you can issue a command such as the following:

```
DISPLAY EVENTS(*) WHERE(MAXDEPTH = 5000 & MAXMSGL >= 4194304)
```

There is no limit to the complexity of your **WHERE()** clause. For a fuller description of the **WHERE()** clause please refer to Chapter 8 Where Clause() on page 42.

Now, let's generate a different event message just to see what it looks like. Let's change the queue definition we created. Issue the following command to your Queue Manager:

```
ALTER QL(MQEV.TEST.QUEUE) MAXMSGL(4000000) DEFPSIST(YES)
```

This will, of course, generate a new command event and a new configuration event.

Let's display them using the following command:

```
DISPLAY EVENTS(*) DISTYPE(DETAIL) MAXRESP(2)
```

...and we would see output such as this:

```
EVQMGR(MQG1)         EVENTS($EVENTS)     EVTIME(2019-10-09 22:26:38 (Local))
EVREASON(CFGCHGOBJ) EVUSERID(mqgemusr)  EVOBJNAME(MQEV.TEST.QUEUE)
EVOBJTYPE(QUEUE)     EVENTID(00000006)
SUMMARY(Config - Change Object - Queue:MQEV.TEST.QUEUE - DEFPSIST[NO -> YES]
MAXMSGL[4194304 -> 4000000])
____________________________________________________________________________
EVQMGR(MQG1)         EVENTS($EVENTS)     EVTIME(2019-10-09 22:26:38 (Local))
EVREASON(CMDCHGQ)    EVUSERID(mqgemusr)  EVOBJNAME(MQEV.TEST.QUEUE)
EVOBJTYPE(QUEUE)     EVENTID(00000005)
SUMMARY(Command - Change Queue – Queue:MQEV.TEST.QUEUE)
____________________________________________________________________________
Total display responses - Received:2
```

We can see that **MQEV** has done a pretty good job at summarising the change for us. Both changes are clearly shown in the **SUMMARY** field of the change object event. Of course there is a limit to how much can be shown in this field since it has a maximum length but it certainly gives a pretty clear indication of what went on. Of course if we want to see the entire message then we can do so by changing the detail. Try issuing the following command:

```
DISPLAY EVENTS(*) ALL MAXRESP(1)
```

The addition of the MAXRESP(1) field tell **MQEV** to only return one event – the latest one. In this case it is the change object we are interested in. Of course we can not guarantee that – MQ may have sent the last two events to us in any order and since they are put to two different queues they could have been processed in a different order. And, naturally, IBM MQ could have generated further messages since we issued the **ALTER QL** command. So, how are we supposed to view a particular event and not have to care whether the event data is changing underneath us?

Well, as you may have imagined it is the **EVENTID** field which is what we need. The combination of Event Queue Manager, Stream Name and Event Id is always guaranteed to be unique. So, if we need to see the first event in more detail we can issue the command:

```
DISPLAY EVENTS($EVENTS) EVQMGR(MQG1) EVENTID(00000006) ALL
```

This will show us the entire event such as the following

```
EVQMGR(MQG1)          EVENTS($EVENTS)      EVTIME( 2019-10-09 22:26:38 (Local))
EVREASON(CFGCHGOBJ)   EVTYPE(CONFIG)       EVUSERID(mqgemusr)
EVOBJNAME(MQEV.TEST.QUEUE)                 EVOBJTYPE(QUEUE)     EVENTID(00000006)
CFHCMD(43)            CFHREASON(2368)
SUMMARY(Config - Change Object - Queue:MQEV.TEST.QUEUE - DEFPSIST[NO -> YES]
MAXMSGL[4194304 -> 4000000])
BEFORE:
EVENTUSER(mqgemusr)
EVSID(1D0101050000000000005150000001AFA5FFE70975006C3A1C633E903000000000000000000000
00000)
EVORIGIN(MSG)
EVACCTTK(160105150000001AFA5FFE70975006C3A1C633E903000000000000000000000B)
EVAPPLID( )           EVAPPLTYPE(WINDOWSNT)
EVAPPLNAME(d:\nttools\mqscx.exe)           EVAPPLORIG( )        OBJTYPE(QUEUE)
QUEUE(MQEV.TEST.QUEUE)                     DESCR( )             PROCESS( )
BOQNAME( )            INITQ( )             TRIGDATA( )          CLUSCHL( )
CUSTOM( )             CLUSTER( )           CLUSNL( )            CRDATE(2019-10-08)
CRTIME(09.50.37)      ALTDATE(2019-10-08)  ALTTIME(10.06.11)    GET(ENABLED)
PUT(ENABLED)          DEFPRTY(0)           DEFPSIST(NO)         MAXDEPTH(5000)
MAXMSGL(4194304)      BOTHRESH(0)          SHARE(1)             DEFSOPT(SHARED)
HARDENBO(1)           MSGDLVSQ(PRIORITY)   RETINTVL(999999999) USAGE(NORMAL)
TRIGCTL(NOTRIGGER)    TRIGTYPE(FIRST)      TRIGDEPTH(1)         TRIGMPRI(0)
QDEPTHHI(80)          QDEPTHLO(20)         QDPMAXEV(ENABLED)    QDPHIEV(DISABLED)
QDPLOEV(DISABLED)     QSVCINT(999999999)   QSVCIEV(DISABLED)    DISTL(NO)
NPMCLASS(NORMAL)      STATQ(QMGR)          ACCTQ(QMGR)          MONQ(HIGH)
SCOPE(QMGR)           DEFBIND(OPEN)        CLWLRANK(0)          CLWLPRTY(0)
CLWLUSEQ(QMGR)        DEFPRESP(SYNC)       DEFREADA(NO)         PROPCTL(COMPAT)
IMGRCOVQ(QMGR)        DEFTYPE(PREDEFINED)  QTYPE(QLOCAL)
AFTER:
ALTDATE(2019-10-08)   ALTTIME(10.32.38)    DEFPSIST(YES)        MAXMSGL(4000000)
Total display responses - Received:1
```

This is a lot of information to deal with but essentially it is split into three parts. The first part is largely event elements we are already familiar with. This is followed by a 'BEFORE' section. This lists the values of the object fields before the change was made. This in turn is followed by an 'AFTER' section and it is here that the event tells us what was changed and their new values.

As we mentioned before, in order to do this **MQEV** has received two different event messages, compared them and constructed this single event to show the differences.

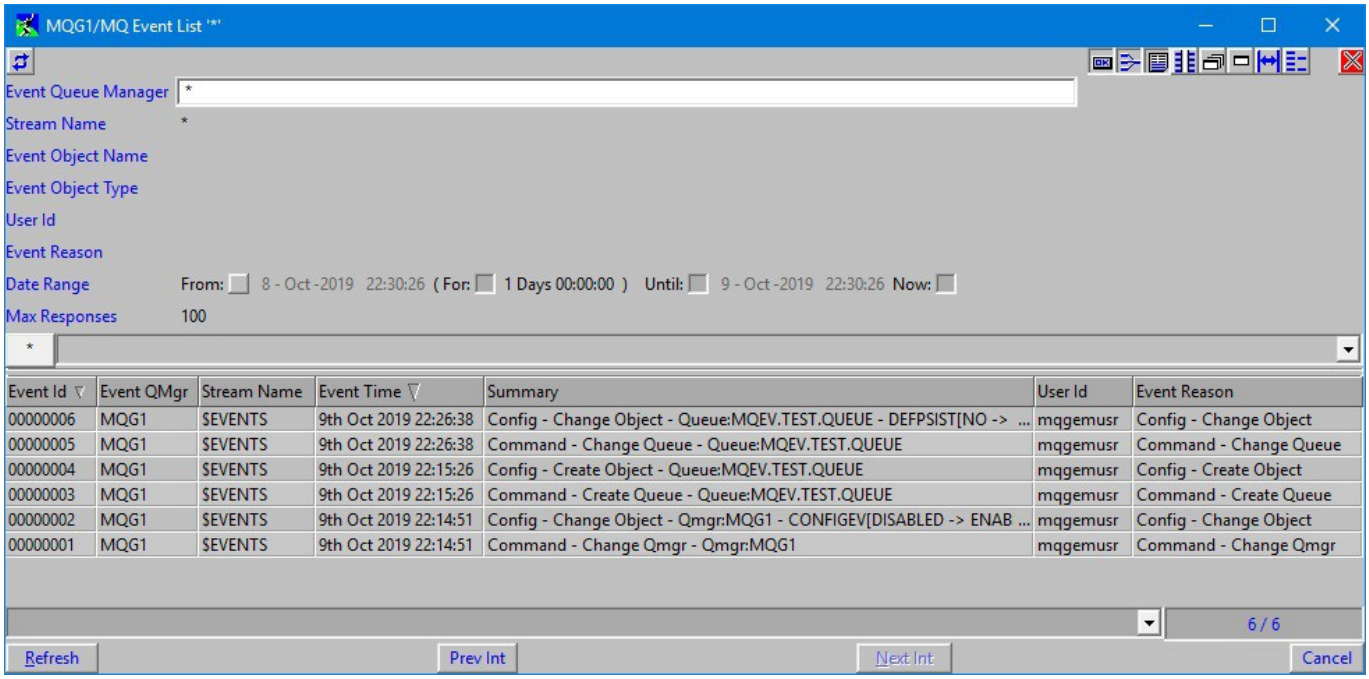
3.6 Displaying MQEV in a GUI (MO71)

Even if you have not installed **MQSCX** it is recommended that you read the section “Displaying MQEV in a command line (MQSCX)“ since it introduces some of the **MQEV** concepts. Command utilities like **MQSCX** are great for automated tasks or for running sets of pre-canned commands. However, for day-to-day monitoring of **MQEV** data it is far easier to use a GUI.

MO71 is MQGem Software's GUI Administrator which can issue commands to IBM MQ queue managers as well as a variety of other tasks. It is no surprise then that you can also issue **MQEV** commands. By selecting a Queue Manager and bringing up the command context menu you should new menus associated with **MQEV**. If you use menu categories then these will be under two new Categories see two new menus Categories

- **MQEV Events**
  These contain the operations associated with MQ Events and general operation of MQEV.
- **MQEV Acct & Stat**
  These contain operations associated with just Accounting and Statistics data.

So, if we select 'MQ Event List...' from the MQEV Events category we will be presented with a list dialog. Pressing “Refresh” on this menu will show us the same data as we saw in **MQSCX** for our two events but this time in a GUI.

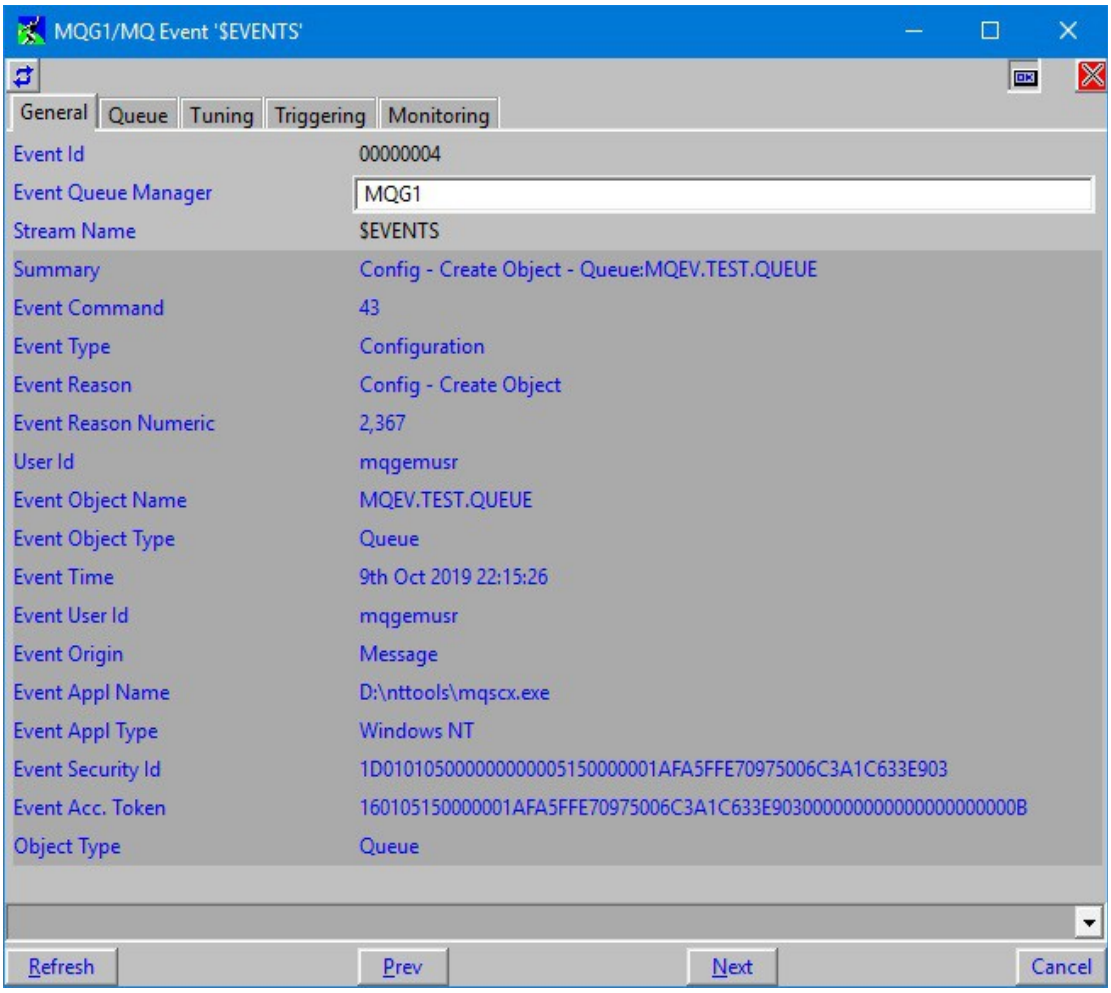


As you might expect the operation of this dialog is largely the same as any other **MO71** dialogs. If you are not familiar with **MO71** then we strongly suggest you read the **MO71** manual from cover to cover. It contains lots of information about powerful features in **MO71**.

Perhaps one thing that users of **MO71** won't immediately recognise is the format of the 'Date Range' field. As you know, when querying **MQEV** Events one of the key ways to limit the information you are shown is to specify a range of dates/times you are interested in. You can do this in different ways. By specifying both ends of the date range or specifying just one end and an elapsed length of time. For example, if you look at the dialog you will see that by default it will show you events in the last 24 hours. This is indicated by the 'Now' button being selected – which means that events up until 'now' will be selected. The 'For' field is also selected with a value of 1 day. So, we have a selection of “a days worth of events up until now”. Each time the 'Refresh' button is pressed the greyed out values will be changed to indicate the effective value used on the command.

Of course all of these fields can be changed. By selecting or deselecting the buttons you can enable the data fields themselves and set whatever value you wish. To change a value just select the required field, you will see a bar above and below the field indicating it has focus. It is then merely a question of using the mouse wheel, clicking on the bars or using the up and down arrow keys to select the value you want. Pressing right and left arrow keys will move through the date elements.

Of course we can look at an individual event in more detail by double-clicking on it. So let's do that now on the configuration create object event. You would get something like this:



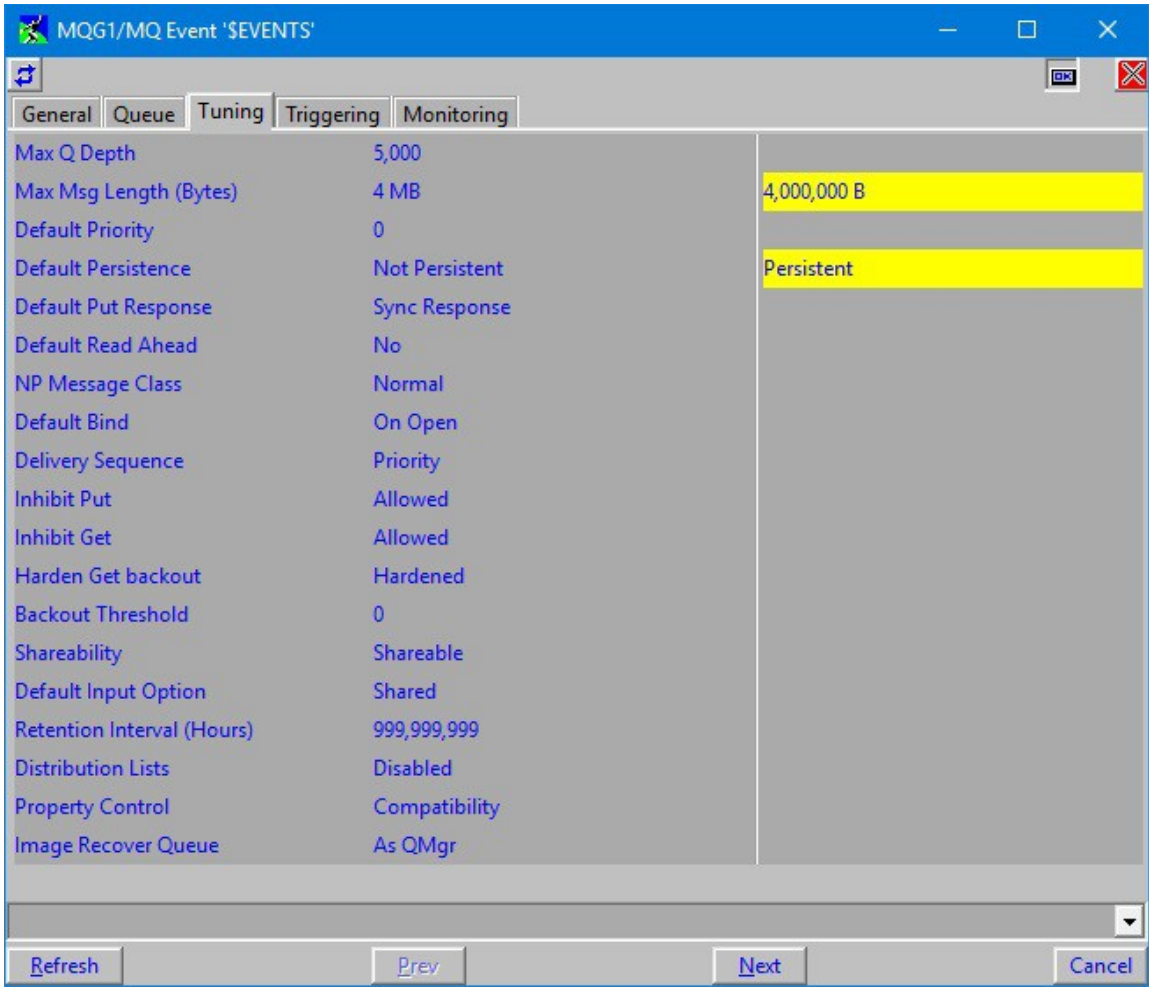
Here we can see this event in more detail. It looks quite similar to a normal Queue dialog and that is no accident. A configuration dialog is essentially a representation of the object that was created and what better way to show it than a dialog that looks very similar to the dialog you would have used in **MO71**.

Of course the exact look of each event will be different. MQ issues a large variety of event messages, each containing different fields, so naturally the dialog to view them is necessarily different. However, all events will have a common set of values shown on the first tab. Whether subsequent tabs are included depends on the event in question.

Let's take a look at the other configuration we created, the change object event.

Double click on that list entry and cycle through the tabs on the resultant dialog.

You can see that there is a striking difference between this dialog and the previous one we looked at.



Here we see the dialog fields essentially split into two sections. I am sure you don't need me to tell you that what we are seeing here is the 'before' and 'after' values. In other words the values shown on the right, highlighted in yellow[8], were the changes that were made that the event is reporting about.

8 You can of course change the highlight colour in the colour choose dialog

3.7 Testing with other events

We have used command and configuration events for this walk-through to introduce you to viewing event messages in **MQEV** because they are very simple to generate; turn them on, issue any command, and there you have an event message. Other event messages are not always so simple to generate.

Therefore, if you would like to have some event messages to use to get to know **MQEV**, we have provided a set of event messages in a **QLOAD** file that you can simply load onto one of your IBM MQ event queues that **MQEV** is processing, and then display them through **MQEV**, either with **MQSCX** or **MO71** as shown in the previous sections.

There is a file called ***TestEventMsgs.qld*** which can be found in the same zip/tar file as the executable. To load it onto an event queue, say `SYSTEM.ADMIN.COMMAND.EVENT` (since you are already set up with **MQEV** processing that queue), run the following command:-

```
qload -m MQG1 -o SYSTEM.ADMIN.COMMAND.EVENT -f TestEventMsgs.qld -CI
```

If you're not a **QLOAD** licensee, you can, in this case, use **dmpmqmsg** with the same parameters.

Of course this assumes that you have the authority to put messages to the `SYSTEM.ADMIN.COMMAND.EVENT` queue on your system. If you don't have this authority, you could instead use the following steps.

```
DEFINE QLOCAL(LOADED.EVENTS)
```

Now load the events messages onto your newly created queue

```
qload -m MQG1 -o LOADED.EVENTS -f TestEventMsgs.qld -CI
```

Now tell **MQEV** to process this queue.

```
=mqev
ADD EVQ(LOADED.EVENTS)
RESUME EVQ(LOADED.EVENTS)
```

Now you will have a large number of events to work with and try out the various commands, for example:-

```
DISPLAY EVENTS(*)
```

4 Parameters

MQEV supports the following parameters:

Parameter	Effect
-b	Background Mode **MQEV** will not output status lines to stdout nor attempt to read characters from stdin. When initially getting going with **MQEV**, running in foreground mode can be useful, and then later, once your log files are where you expect them to go, using background mode would be appropriate.
-c <Maximum Command Level>	By default **MQEV** will not run against Queue Managers which are beyond a command level it understands. This is to ensure that any generated events are understood. However, the user can override this behaviour by explicitly specifying the supported command level. This parameter can be used to both increase or reduce the supported command level. You could, for example, reduce the supported command level if you knew that your script functions do not yet support later command levels.
-f <Script File>	Location of script file if not the default ***mqev.mqx*** in the default location. File can be specified as just a path, just a file name, or both depending on what you wish to override. If not specified on z/OS, **MQEV** will attempt to open DD card **MQEVMQX**. Whether specified using the **-f** flag or the DD card, this can reference a PDS member, a sequential data set, a UNIX file, or just a UNIX file system directory (in which case the default file name ***mqev.mqx*** will be used).
-k	An indication that MQEV is running as an IBM MQ Service. This will have the following effect: • MQEV will run in background mode • MQEV will not retry the connection if it detects that the 'service' Queue Manager is ending. The 'service' Queue Manager will be the Queue Manager that MQEV is defined as a service of.
-l	Make a client connection to the monitored Queue Manager Note that a State Queue Manager, if used, will always use a bindings connection. This parameter does not apply when the program is running on z/OS. You can of course connect to a z/OS queue manager using this parameter when the program is running on a platform other than z/OS.
-L <Log Path>	Directory to store the **MQEV** logs if you wish to override the default location. On z/OS this can specify a Partitioned Data Set (PSDE) or a UNIX file system directory. For full details of how to over-ride the default location of the **MQEV** log files, see Chapter 7 Logging on page 41.
-m <Queue Manager>	Specify the name of the queue manger to connect to if not the default.

<table>
<tr><td><code>-r <Retry Scheme></code></td><td>This parameter controls how frequently MQEV will attempt to connect to a Queue Manager. This retry will occur when the initial connection attempt is made and also if that connection is subsequently lost. Note that not all failures are retryable. MQEV will not retry, for example, if it finds it's command queue missing.

The retry settings are in two stages. A short retry interval and count and a long retry interval and count. Very similar to IBM MQ Channels. You can specify any values you wish but the following values are predefined:
<table>
<tr><th>Name</th><th>Short Interval</th><th>Short Count</th><th>Long Interval</th><th>Long Count</th></tr>
<tr><td>low</td><td>120</td><td>5</td><td>600</td><td>999999999</td></tr>
<tr><td>default</td><td>60</td><td>10</td><td>300</td><td>999999999</td></tr>
<tr><td>high</td><td>10</td><td>60</td><td>60</td><td>999999999</td></tr>
<tr><td>none</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
</table>
So, a parameter of -r high will ask MQEV to do a high rate of retry which is 60 attempts 10 seconds apart and then falling back to retrying every 60 seconds forever.

If you don't want to use the predefined values then you can specify your own values using a command such as: -r 10(20),120(999999999)
Each part of the specification is optional and the missing value will be taken from the default values above. So, the parameter -r 10(20),120 will have the same effect.</td></tr>
<tr><td><code>-s <State Queue Manager></code></td><td>The Queue Manager where the persistent state queue is held.

It is recommended that you use this option if you are connecting to your monitored Queue Manager over a client connection so that all the MQEV state is kept local to the MQEV program.</td></tr>
<tr><td><code>-u <User Id></code></td><td>The user identifier to be used for the MQ Connection.

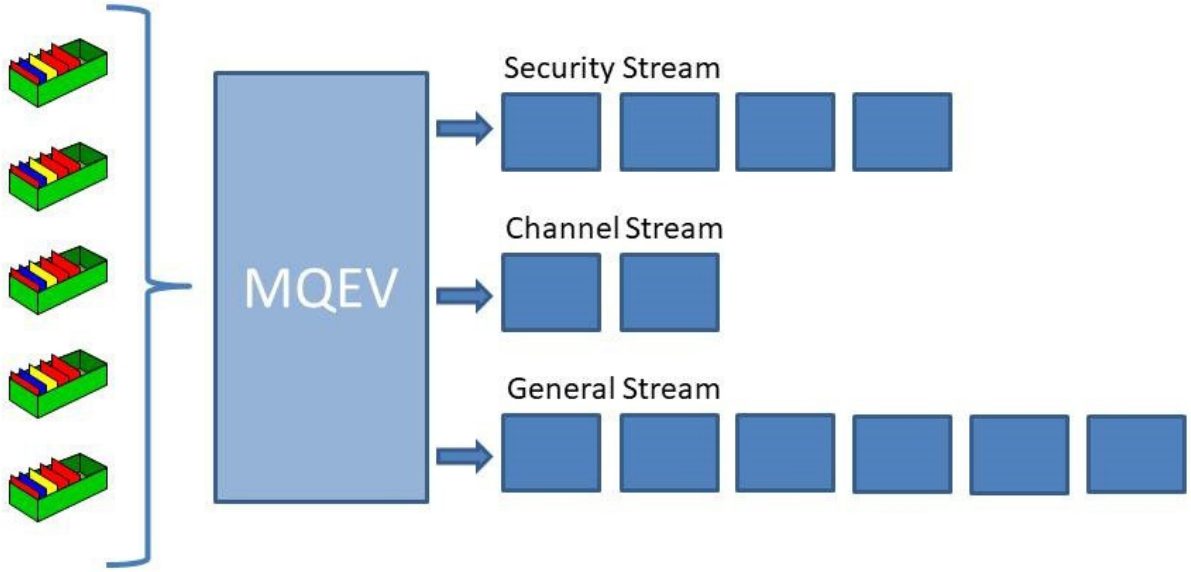
The parameter will apply to the preceding <code>-m</code> or <code>-s</code> parameter</td></tr>
<tr><td><code>-U <Password></code></td><td>The password to be used for the MQ Connection.

The parameter will apply to the preceding <code>-m</code> or <code>-s</code> parameter</td></tr>
</table>

-v <Verbose options>	Verbose options tell **MQEV** to output extra information to it's main window. Normally they are not used but they can be useful to diagnose certain situations. **'c'** – Print compression statistics **'d'** – Print PCF details **'h'** – Print hardening activity **'l'** – Print log file access **'m'** – Print output machine information (useful for licences) **'P'** – Print compressed PCF **'p'** – Print source PCF **'r'** – Print retry values **'s'** – Print storage statistics **'S'** – Print stream statistics **'t'** - Print display time values **'T'** – Print event storm diagnostics
-!	Run **MQEV** in debug mode. For more information about debugging please refer to Chapter 16 Debugging on page 187.

5 Streams

MQEV allows the user to split the incoming events into streams. Streams are just an administrative aid which allows the events to be grouped. This could be shown in the following diagram:



Depending on the type of message they will go to different streams by default.

Type of Message	Default Stream	Default Retention Interval
MQ Event	`$EVENTS`	90 days
MQ Statistics Queue	`$STATQ`	45 days
MQ Statistics Channel	`$STATCHL`	45 days
MQ Statistics MQI	`$STATMQI`	45 days
MQ Accounting Queue	`$ACCTQ`	45 days
MQ Accounting MQI	`$ACCTMQI`	45 days

These are the defaults. However, you could choose, for example that you would like to group all of your authorisation failures into a security stream. Or you might decide that you would like to split the streams by application groups. The choice is entirely up to you; whatever makes sense for your installation.

So, why would you want to split the streams? Well there are two reasons. The first is the retention level. Each stream can have a separately configured retention interval. As you can see by default the `$EVENTS` stream has a retention interval of 90 days. The assumption is that after 3 months the event information is of little use. However, you might decide that there are certain types of events you want to keep for longer, or shorter.

The second reason is that it can be a useful administrative grouping. You can **`DISPLAY`** or **`PURGE`** streams by stream name. For example the command:

```
DISPLAY EVENTS(SECURITY)
```

would only show you events stored in your security stream. Always assuming, of course, that you had named it `SECURITY.` Any stream can be identified as the 'default' stream i.e. the stream that should be used by default for that type of data. You can, of course, have only one stream identified as the default for each type of message.

5.1 Directing events to a stream

So, how do you configure certain events to go to certain streams? Well, here we need to put our programmers hat on but don't worry it is very simple. When **MQEV** receives an event it will always call the `MQEVEvent()` function in your script file. During the processing of that function all you need to do is set the system variable `_stream` to the name of the stream you wish to use. So, in it's simplest form we could code this:

```
func MQEVEvent()
  _stream = "SECURITY"
endfunc
```

Of course this would mean that all event messages would be sent to the security stream. Probably not what we want. So, we would modify the code to check what the event being processed is, perhaps something like this:

```
func MQEVEvent()
  if (event.evtype = AUTHOR)
    _stream = "SECURITY"
  endif
endfunc
```

This tells **MQEV** that any authorisation events should be sent to the `SECURITY` stream. Of course the beauty of having this type of processing in a script rather than in a configuration file is that it is totally flexible. You can choose how events are distributed amongst your streams. It could be by event type, by application group, by 'seriousness', by object name[9] or even day of the week if you so wish!

5.2 Directing accounting and statistics messages

It is also possible to do a similar thing with accounting and statistics messages, however there is an operating difference in that many of the accounting and statistics messages contain data for multiple different objects in the same message. It is therefore not possible to direct these messages to a stream by object name. However, it may still be useful to direct the stream used based on other criteria, such as the application name, the time of day, or certain criteria that make the data interesting in some way.

In the function example below we initially assume that we will discard this message, by setting the stream name to `"$null"`. We then look for interesting data – higher volumes of puts and gets than expected in this example, and set the stream name accordingly. The complete queue accounting message is then stored on the nominated stream.

```
func MQEVAcctQ()
  ******************************************************************
  * Initially assume we plan to discard this message (set null stream)*
  ******************************************************************
  if (_idxData = 1)
    _stream = "$null"
  endif

  ******************************************************************
  * Save this message to our high volume stream if numbers are large *
  ******************************************************************
  @allput = data.PUTNP + data.PUTP + data.PUT1NP + data.PUT1P
  if (@allput > 10000 AND (data.GETP + data.GETNP) < @allput/2)
    _stream = "HIGH.VOLUME"
  endif
endfunc
```

9 It is not recommended that you have too many streams. A stream is the unit of expiry and consolidation. Having many streams means that the events do not fill up the stream messages quite so quickly. This means that there can be more 'dead' information which can not yet be discarded.

6 Emitters

As we know by now **MQEV** will collect your **IBM MQ** event, accounting and statistics messages and store them for later retrieval, filtering and display. However, there are a number of reasons why you might want **MQEV** to immediately output data to a file or queue.

- **Downstream processing**
  We have already mentioned that for every event queue that **MQEV** is configured to read you defined a forwarding queue which can contain a copy of the messages. This allows you to easily daisy-chain downstream programs. However, suppose the downstream program is not well suited to receiving **IBM MQ** event messages. After all `PCF` messages are not overly intuitive or easy to process. One might easily prefer JSON format for example. This is what Emitters bring to the table. You can configure an Emitter which will output any received event messages in JSON format which might be suitable for a downstream service such as Elastic or Splunk.
- **Archiving and Logging**
  Some people like to have a record of all that has happened in a simple, easy to read, file. Emitters can be configured to write to a file in CSV, MQSC or JSON format.
- **Debugging**
  When you are initially configuring the system it can be very handy to get a file created in a directory to confirm that an **IBM MQ** message has been caught by **MQEV** and to show you the content.

6.1 Stream configuration

Streams are the **MQEV** construct for grouping together event data that should be treated in a similar way, for example, that have the same retention requirements. Streams are also where you configure **MQEV** to emit the event data on that stream in a particular format. You do this by first describing the way your want the data to be emitted in an **`EVEMIT`** object, and then reference that new object on one or more streams.

Here is an example:-

```
DEFINE EVEMIT(TO.ELASTIC) QUEUE(ELASTIC.JSONQ) FORMAT(JSON) +
       DESCR('Write JSON format messages out for sending to Elastic') +
       PUT(IMMEDIATE)

ALTER EVSTREAM($EVENTS) EVEMIT(TO.ELASTIC)
```

In this example, **MQEV** has been configured to create JSON format versions of all the events on the default event stream, **`$EVENTS`**, and to write them to a queue called **`ELASTIC.JSONQ`**. Now an application can read the messages from that queue and post them to an Elastic HTTP listener. The supplied **GetPost Application** (described on page 38) can be used for this purpose. Of course Elastic is just one example, one could easily write a different application to import the data into any other environment. The use of an MQ queue removes the need for **MQEV** itself to be aware of all of these different environments and therefore adding downstream applications is straight-forward. Secondly the queue adds a useful buffering mechanism. If, for some reason, the downstream process is not available or slow, the queue will be used to buffer the data until the service is available again this allows **MQEV** to continue processing the other events in the system and responding to commands.

6.2 Emitter Code page

By default **MQEV** will output the data in the Queue Manager's code page. However, if you are outputting the data to a file then you can override this value by setting environment variable **MQEV_EMIT_CCSID** to the integer code page. For example, **MQEV_EMIT_CCSID=1047.** Note that the code page should be an SBCS value and the Queue Manager must know how to convert from the Queue Manager code page to this code page. For information about installing conversion tables in IBM MQ please refer to the IBM MQ documentation.

6.3 Emitter Formats

6.3.1 CSV

A common interchange file format where fields are comma separated. This format is particularly useful for loading into programs such as a spreadsheet. Once loaded into a spreadsheet then clearly the data can be manipulated or graphed with ease.

6.3.2 JSON

One of the formats you can emit your data in is JSON. This might be useful for uploading into systems such as Elastic or Splunk. If you emit your JSON data to a queue, you might find the supplied **`GetPost Application`** (see page 38) application useful.

6.3.3 NDJSON

This format, Newline Delimited JSON, is a slight variation of standard JSON. Rather than emitted objects being contained in a JSON array the objects are merely separated by a newline character. This means that it is easier to have a downstream program reading the emitted content at the same time as MQEV is writing the data since it need not worry about array opening or closing brackets.

6.3.3.1 Unique ID

When **MQEV** emits data in JSON format, it contains a unique id, `mqevUniqueId`. This field is designed to ensure that if you run your MQEV emitters in a mode that could cause duplicate data, i.e. using the **`EVEMIT`** parameter **`PUT(IMMEDIATE)`**, then your JSON data receiver, e.g. Elastic or Splunk, can be configured to discard any duplicates by using this field.

Possible formats of this field are:

```
"mqevUniqueId": "414D51204E5450474331202020202020FFCD8D61028D1040-1"
```

or

```
"mqevUniqueId": "414D51204E5450474331202020202020FFCD8D61028D1040"
```

6.3.4 MQSC

Those familiar with RUNMQSC may prefer the data to be output in a 'MQSC-like' format. Clearly the files will not themselves be true MQSC since the data is not an actual object. However, the easily recognisable format can be useful for archiving the data. An example would be:

```
ACCTQ('MQGEM.MQEV.COMMAND.QUEUE') EVQMGR('NTPGC1') INTVLSTA('2021-11-15 08:59:49
(UTC)')INTVLEND('2021-11-15 09:13:57 (UTC)') CMDLEVEL(923) APPLNAME('MQGem Software
MQEV')CONNID(414D51434E5450474331202020202020FFCD8D61009A1040) USERID('Paul') PID(13192)
TID(1) QTYPE(QLOCAL) DEFTYPE(PREDEFINED) OPENCNT(4) OPENTI('2021-11-15 08:59:49 (UTC)')
CLOSECNT(4) CLOSETI('2021-11-15 09:13:57 (UTC)') ALLPUT(1) PUT(1) PUTNP(1) GET(7) GETNP(7)
GETFAIL(6) GETBYTE(520) GETBYTENP(520) GETMAXBYTE(96) GETMAXBYTENP(96) ONQMINNPTI(19)
ONQMAXNPTI(20977161) ONQAVGNPTI(3160297)
```

6.4 Emitter File Name

You can configure an emitter to output the data either to a queue or a file. If you choose a queue then each set of emitted data will become a message on the queue. However, if the output is a file then you have some additional flexibility. It is unlikely that you just want **MQEV** to write more and more data to a file with no limit. This would cause a number of problems not least of which that **MQEV** would continually keep the file open which would prevent you from using the file downstream. Instead what you probably want is that **MQEV** will write to the directory in a set of files. Exactly what that file distribution would be would probably depend on the usage of the data. If, for example, you were just keeping an archive of the processed data then it is likely you would want to put some time limit on each file. For example, you could request that each file contains and hours worth of data. The way to achieve this is to simply put the time (with a resolution of an hour) in the file name.

Such as this:

```
FILE(c:\emit\statq_%c_%H.mqx)
```

One could just as easily make the file have a resolution of a day or a minute. The full range of name inserts are given below. Here we have essentially used a file name with a resolution of an hour. This means that as **MQEV** comes to write each piece of data it will calculate that a different file is required each hour.

Of course we might not be archiving the data but passing it to a downstream program. One could argue that in this case it would make more sense to use a queue but suppose our downstream program reads only files. How do we get each set of data to use a different file ? Well all we need to do is add either a %i insert or %u insert.

```
FILE(c:\emit\statq_%c_%H_%i.mqx)
```

or perhaps just even

```
FILE(c:\emit\statq_%i.mqx)
```

6.4.1 Emitter Filename Inserts

Sequence	Effect
%%	%
%i	A monotonically increasing number with each set of data written. **MQEV** will start the sequence at the first file which does not already exists in the directory
%I	As %i but the index number is padded to five digits.
%u	A unique number, starting at 1, that included in the file name to make the file name unique.
%U	As %u but the unique number is padded to five digits.
%p	**am** or **pm** depending on time of day
%P	**AM** or **PM** depending on time of day
%h	Two digit hour (12 hour clock)
%hh	Hour (12 hour clock)
%H	Two digit hour (24 hour clock)
%HH	Hour (24 hour clock)

%M	Two digit minutes
%S	Two digit seconds
%d	Two digit day of month
%dd	Day of month including suffix eg.1st, 2nd, 3rd …
%j	Zero based Julian date
%J	One based Julian date
%m	Three character month name eg.Jan,Feb,Mar….
%mm	Two digit month
%mmm	Full month name eg. January, February,March…
%y	Four digit year
%yy	Two digit year
%t	Simple time format HHMMSS eg. 181403
%c	Current date in YYMMDD format
%C	Current date in YYYYMMDD format

6.5 GetPost Application

We supply a sample Python application, the **`getpost`** application, to read from an emitter queue and post the contents of the message to a supplied URL. This can useful to interface with environments such as Elastic and Splunk, or any other HTTP listening process that accepts JSON format data. The program reads from a queue, checks that the message content is valid JSON, and then posts the message data to the supplied URL. We supply the source of this application so if you need to use it for other similar environments, you can adjust as necessary. On Windows we also supply an executable version of the program for those of you that cannot, or do not want to, run a Python interpreter.

If you do make changes to the program and you feel that the changes would be useful to other members of the MQ community then please do feel free to send us a copy of your changes and we will possibly incorporate them into the product.

The supplied **`getpost`** application can either be run from the command line or be started using IBM MQ Triggering.

6.5.1 Parameters

The command line parameters for the **`getpost`** application are shown in the table below.

Parameter	Meaning
`-l <filename>`	**File to write logging information to.** The program will always write to stdout, but in some environments, there will be no stdout to view, so information can also be written to a file.
`-m <Queue Manager>`	Queue Manager name (not needed when triggered)
`-q <Queue Name>`	Input Queue Name (not needed when triggered)
`-u <URL>`	URL of HTTP(S) Listener
`-b`	**Use the Elastic Bulk format.** This will convert any JSON array objects into nd-json (newline delimited JSON) and use the Elastic bulk format URL to upload multiple objects with a single POST. Use of this format will also use the `mqevUniqueId` as the `_id` in the Elastic index. It is

	recommended to use this flag when POSTing to Elastic, but only essential for uploading `ACCTQ`, `STATQ` and `STATCHL` types of data as those contain a JSON array of objects.
`-U <userid>`	Userid for connect to queue manager
`-P <password>`	Password for connect to queue manager If not supplied the password will be prompted for if -U is used
`-v`	Verbose output. When switched on the program will output more about what it is doing to stdout and the log file (if used).
`-p`	Output the message content to stdout and log file (if used)
`-w <Wait Interval>`	Wait interval to wait for more messages (seconds). The default is 300 seconds (5 minutes).
`-c <Num Msgs>`	Commit after this many messages Default of 50 messages if neither -c or -C are specified
`-C <Num Bytes>`	Commit after this many bytes Default of 1,000,000 bytes if neither -c or -C are specified

6.5.2 Error processing

The **`getpost`** application will use the backout requeue queue, if one is defined on the input queue. Any messages found with an `MQMD.BackoutCount` greater than the threshold defined on the input queue, will be treated as a poison message and immediately put to the backout requeue queue.

Any messages which are discovered not to be valid JSON will also be put to this queue so that the application can continue to process other messages on the input queue.

6.5.3 Transactions

The getpost application will get messages from the input queue inside a transaction. The transaction will be committed once the specified number of messages, or number of bytes of data, have been read, whichever is reached first. Since an HTTP POST is not transactional, if a number a messages are read with MQGET and then POSTed to the URL and then there is a failure and the transaction of gets is rolled back, the HTTP POSTs will be repeated. Each **MQEV** JSON message has a unique ID in the data that can be used to detect duplicate POSTs to the HTTP URL by the listening service. How this is done will depend on the service in use. See the `-b` flag if the service you are using is Elastic. You can reduce the risk of repeated messages by reducing the transaction size to 1 using the **`-c`** flag.

6.5.4 Using Triggering

Here are some example commands to show how to set up the **`getpost`** application to be triggered to run when messages are put by **MQEV** on the emitter queues.

Define an initiation queue:

```
DEFINE QLOCAL(PYTHON.INITQ) DESCR('Initiation Queue for Python App')
```

Define a single process object to describe the **getpost** application. This definition can contain parameters to the **getpost** application that are the same for each emitter queue.

```
DEFINE PROCESS(PYTHON.PROC) DESCR('Triggered Python App') +
       APPLICID('C:\MQGem\getpost.py')                    +
       USERDATA('-c 25 -w 120')
```

Define each emitter queue to be triggered. This definition can contain parameters to the **getpost** application that differ depending on the queue being triggered, for example the target URL.

```
DEFINE QLOCAL(ELASTIC.JSON.EVENTS)                   +
       DESCR('Emitter Q for JSON Events to Elastic') +
       INITQ(PYTHON.INITQ) PROCESS(PYTHON.PROC)      +
       TRIGGER TRIGTYPE(FIRST)                       +
       TRIGDATA('-u https://mqgem.es.io/mqev-events/doc -b')
```

```
DEFINE QLOCAL(ELASTIC.JSON.STATQ)                   +
       DESCR('Emitter Q for JSON StatQ to Elastic') +
       INITQ(PYTHON.INITQ) PROCESS(PYTHON.PROC)     +
       TRIGGER TRIGTYPE(FIRST)                      +
       TRIGDATA('-u https://mqgem.es.io/mqev-statq/doc -b')
```

7 Logging

MQEV will write out status messages to a log file. The location of these log files is in one of the following places, listed in precedence order.

- The program parameter **`-L`**
- A DD card named **`MQEVLOG`**, which can reference a PDS or a UNIX file system directory. This only applies on the z/OS platform. This DD name must not supply a member name.
- An environment variable **`MQEV_LOG_PATH`**
- The same location as where the program resides, except on AIX and z/OS UNIX, where the location of the program cannot be determined, so the current directory is used instead.

If you have more than one queue manager on the same machine, and therefore more than one instance of **MQEV** running, ensure that the log path used by each instance is distinct, for example by including the queue manager name in the log path.

The log files themselves will be called ***MQEVLog_<Date><Time>.txt***, unless written to a Partitioned Data Set (PDSE), in which case the log files will have a member name generated as ***LYJJJxxx*** where

- **`L`** – just the letter L
- **`Y`** is a 1-digit year
- **`JJJ`** is the 3-digit, 1-based, Julian date
- **`xxx`** is an incrementing number if more than one log file is written in any one day. This number uses digits from 000-999.

On z/OS, if you choose to write log files to a Partitioned Data Set, you are advised to use the DD card method, so that you can set **`DISP=SHR`**. Otherwise you will not be able to view old log files in the Partitioned Data Set while **MQEV** is running.

As **MQEV** runs it will always have one log file open. You can control the maximum size and age a log file can get before the file is closed and a new log file is opened. If **MQEV** runs for a long time you can also tell it to delete log files once they reach a certain age.

Note that **MQEV** will not delete any log files generated in a previous instance of **MQEV**. If you want that to happen then we suggest you run **MQEV** in a simple script which erases all ***MQEVLog**** files before invoking **MQEV**.

8 Where Clause()

The standard IBM MQ **WHERE()** clause is rather restrictive. It is also complicated and has a difficult to follow syntax. However the ability to specify additional search criteria is very useful so we need something along these lines. So, for the **MQEV** commands we have implemented our own flavour of a **WHERE()** clause which is similar in many ways to the IBM MQ **WHERE()** clause but extensively enhanced.

MQEV uses the same **WHERE()** clause engine which is used in **MQSCX.** This removes many of the restrictions found in the IBM MQ **WHERE()** clause and enhances it so that one can now issue much more powerful queries.

So, perhaps listing the standard restrictions might be a good place to start. If you are new to the **WHERE()** clause it might be worth revisiting the IBM MQ manuals to read a description of the syntax and come back when you have the basics. You will have seen that you can do lots of good stuff in a **WHERE()** clause but what might not be so obvious is what you can't do……which is:

- You can't use the primary object name in a **WHERE()** clause
- You can't use any of the sub-filter attributes in a **WHERE()** clause
- A **WHERE()** clause may only contain one operation – you can't link them together with **AND** and **OR**
- You can only have a single attribute in a **WHERE()** clause. For example, you can't compare one attribute with another.
- Although it supports wildcard comparisons the wildcards may only be at the end of strings.
- You can't use indicators[10] in a **WHERE()** clause.

Phew! Quite a list of restrictions and unfortunately most of these would be very nice to have.

But, as you may suspect, with **MQEV** none of these restrictions apply.

Let's try some real world examples. Provided you know the **WHERE()** syntax you should have no trouble working out what the command is doing. Just to keep it interesting we'll stick to commands which would have been illegal to issue using the IBM MQ **WHERE()** clause rules, see if you can spot why the command would have been illegal.

```
DISPLAY EVENTS(*) WHERE(EVOBJNAME LK '*ABC*')
```

So, this command will show you all the events which were generated that contain the text 'ABC' somewhere in the name of the object that the event is about. Note that we can use multiple wildcards in the operand.

```
DISPLAY EVENTS(*) WHERE(EVUSERID = 'mqgemusr' AND EVOBJNAME LK '*ABC*')
```

This command will show you all the events where the user id 'mqgemusr' did something to objects that have 'ABC' somewhere in their name. This shows that you can link multiple expressions using **AND** and **OR** operators. If you prefer you can use the operators '&' and '|' to represent the 'and' and 'or' operators. Using '&' and '|' can make the expression easier to read and clearly involve less typing, partly because they are shorter but also because they don't require a space character to separate them from the attribute.

Ok, so, what other flavours of **WHERE** expressions are supported. Consider the following:

```
DISPLAY EVENTS(*) WHERE(MSGSIN >= 2*MSGSOUT)
```

This command compares two attributes within the same event, in this case a performance event.

It is requesting to see all the events where the number of messages enqueued is two or more times greater than the number of messages dequeued in the same period.

This command demonstrates a couple of points worth mentioning

10 An indicator is an attribute, usually returned on a status display, which contains two values in list format. Usually a short term and a long term average. Examples include NETTIME on channel status or QTIME on a queue status response.

- **Mathematical expressions**
  The **WHERE()** clause is essentially one big Boolean expression. If the expression evaluates to TRUE (non-zero) for any object then it is displayed. If it evaluates to FALSE (zero) then it is not displayed.

  Within that remit you are free to enter virtually any mathematical expression you like, subject to the supported operators. If the expression evaluates to a non-zero value then the object is displayed. So, for example a filter of **WHERE(1)** is perfectly valid although somewhat superfluous.

  For a complete list of the WHERE operators please see Appendix A. Expression Operators on page 209.

- **Operator synonyms**
  A number of operators have synonyms. In this case the operator **GE** could equally have been used. It is purely a matter of taste whether you have more of an SQL versus a mathematical background.

So, staying on a mathematical theme can we guess what this might do?

```
DISPLAY EVENTS(*) WHERE(EVOBJNAME < 10)
```

To understand this filter we really just need to know how **MQEV** handles an expression containing both strings and numbers. The answer is coercion[11]. **MQEV** will automatically convert any string value to a numerical equivalent if it has an operation concerning both a string and number. The number it converts it to is its length. So, this command is saying, display any events which have an object name of less than 10 characters.

Another example:

```
DISPLAY EVENTS(*) WHERE(EVOBJNAME != upper(EVOBJNAME))
```

It should be fairly clear what this is doing. We are asking to see any events where the object name is not upper case i.e. The object name is not the same value when upper cased. Of course depending on your system configuration you may or may not see results. Commands like these can be useful to ensure compliance with installation standards.

One of the most notable things about this command is the use of a function in the **WHERE()** clause. For a list of available functions please see Appendix B. Expression Functions on 210.

The next expression format we want to discuss is enumerated types. These are many event attributes which have a fixed set of values, for example, suppose we wish to see the events which were caused by applications running on Unix. We would enter the command:

```
DISPLAY EVENTS(*) WHERE(APPLTYPE = UNIX)
```

Note that you must enter the fields this way round. It would be invalid to enter the command:

```
DISPLAY EVENTS(*) WHERE(UNIX = APPLTYPE)
```

This would generate a variable error since the program would look for a variable called 'unix'.

Variables can also have modifiers, either prefixes or suffixes, which allow you to identify a particular instance of the variable. For example some variables are what is known as 'indicators'. These contain both a long term and short term average value. You can access these by using the .LONG and .SHORT suffixed. For example `QTIME.SHORT`.

Another case where a modifier is useful is the change object event. When an object is changed IBM MQ will send two event messages, a before event and an after event. These are awkward to deal with so **MQEV** will combine the two into a single change object event message where the before and after fields are stored in before and after groups. It follows therefore that in the same event message there could be two fields of the same name.

11 Coercion, in the computing sense, is merely the process of making one data type compatible with another data type.

This allows us to issue a command such as the following:

```
DISPLAY EVENTS(*) WHERE(BEFORE.INITQ != AFTER.INITQ)
```

This says “show any event which changed the value of an initiation queue”. This can be very useful for tracking down when something changed and who did it.

8.1.1 Attribute presence

Of course it is possible to use an attribute in an expression which doesn't get returned in each event. Consider the expression:

```
DISPLAY EVENTS(*) WHERE(INITQ NE 'xxxxxxxx')
```

One might expect that this expression would show all events relating to queues since it is unlikely that you have any queues defined with an INITQ value of 'xxxxxxxx'. However, the semantics of the `WHERE` clause is such that the expression always evaluates to FALSE if a variable is not present. So, only events messages which actually contain the `INITQ` field will be returned.

This mechanism is quite useful since, given the **`WHERE`** expression, it is likely that the user is only interested certain types of objects.

However, as we have heard in the sections above, the **`WHERE`** clause has been extended to include logical OR expressions so in this case this rule has been this relaxed.

Consider the expression:

```
DISPLAY EVENTS(*) WHERE(RNAME | TARGET)
```

If the `WHERE`() clause stuck to the original rule then this expression would never return any objects since 'rname' is an attribute of a remote queue and 'target' is an attribute of an alias queue. The two attributes are mutually exclusive since they belong to different object types. However, the rule above has been relaxed and this expression will return a list containing all remote queues which have a value for 'rname' and all alias queues with a value for 'target'.

9 Running MQEV with your Queue Manager

Generally speaking you would want **MQEV** to be running whenever your Queue Manager is running. This means that **MQEV** will be ready, willing and able to process events as and when they arrive.

There are slightly different facilities available on distributed and z/OS platforms to achieve this. This chapter describes those facilities.

9.1 Running MQEV as an IBM MQ Service (Distributed platforms)

Handily, IBM MQ has a mechanism for programs to use, if they wish to start and end alongside the Queue Manger, known as Service objects. The idea is that you define a service object which describes the program you wish to configure and the commands needed for both starting and stopping it. You can then either issue commands `START` and `STOP SERVICE` manually or have them automatically start and stop as the Queue Manager comes up and down.

In the case of **MQEV**, the definition you would make would be along the following lines:

9.1.1 When running on the event Queue Manager

```
DEFINE SERVICE(MQGEM.MQEV) +
       DESCR('MQGem Software Event Monitor') +
       CONTROL(QMGR) SERVTYPE(SERVER) +
       STARTCMD('mqev') STARTARG('-m +QMNAME+ -k') +
     [ STOPCMD('mqscx') STOPARG('-m +QMNAME+ -f -C "=mqev;STOP EV"') ]*
```

You can see that starting the program is a simple case of running the **MQEV** program. If the program is not in your path then you may need to provide the fully qualified location.

* As far as stopping **MQEV** is concerned there are two modes.

1. It is actually recommended that you leave the `STOPCMD` and `STOPARG` blank since MQEV will end anyway when the Queue Manager comes down. The only benefit of `STOPCMD` and `STOPARG` is that they allow you to end **MQEV** using the `STOP SERVICE` command. However, the advantage of this is outweighed by the disadvantage of a less efficient command sequence when the Queue Manager ends since the `STOP SERVICE` command will instantiate a new **MQSCX** instance.
2. If you do set the value of `STOPCMD` and `STOPARG` as above then `STOP SERVICE` will work. However, note that making a new connection when the Queue Manager comes down is slightly less efficient.

9.1.2 When running using a State Queue Manager

When you are using a state Queue Manager the definition is slightly different. For one thing the service object is defined as a service on the state Queue Manager rather than the event Queue Manager. You would define a service object for each remote Queue Manager you wished to monitor. The definition would look something like the following:

```
DEFINE SERVICE(MQG1) +
       DESCR('MQGem Software Event Monitor') +
       CONTROL(QMGR) SERVTYPE(SERVER) +
       STARTCMD('mqev') +
       STARTARG('-m +MQ_SERVICE_NAME+ -l -s +QMNAME+ -k') +
       STOPCMD('mqscx') +
     [ STOPARG('-m +MQ_SERVICE_NAME+ -l -f -C "=mqev;STOP EV"') ] *
```

The inserts such as, **+QMNAME+**, can be entered exactly like this and IBM MQ will substitute the actual name of the Queue Manager for you. This can be very useful since if you stick to the naming convention and exact format above you can make additional definitions for additional Queue Mangers using the very simple command:

```
DEFINE SERVICE(MQG2) LIKE(MQG1)
```

9.2 Running MQEV in batch (z/OS only)

On z/OS, you are most likely to want to run the **MQEV** program in batch. What follows is an example piece of JCL to allow you to do that and an explanation of the various DD names that **MQEV** will look for.

```
//MQEV     EXEC PGM=MQEV,
//          PARM=('-m MQG1')
//STEPLIB  DD DSN=MQGEM.LOAD,DISP=SHR
//         DD DSN=IBM.MQ.SCSQANLE,DISP=SHR
//         DD DSN=IBM.MQ.SCSQAUTH,DISP=SHR
//MQGEML   DD DSN=MQGEM.LICENCES,DISP=SHR
//MQEVMQX  DD DSN=MQGEM.MQEV.SCRIPTS(MQEV),DISP=SHR
//MQEVLOG  DD DSN=MQGEM.MQEV.LOGS.MQG1,DISP=SHR
```

There are three well defined DD names that the MQEV program will look for.

9.2.1 DD name MQGEML

The **MQEV** program must be able to find your MQGem Software licence file in order to be able to run. This is a licence file specific for **MQEV** on z/OS. A distributed platform licence will not enable MQEV on z/OS to run.

See 2.1 Licence File Location on page 9 for more information about using the **MQGEML** DD name.

9.2.2 DD name MQEVMQX

The **MQEV** program must be able to find the **MQEV** script. This can be located using the **-f** program parameter (see 3.3.2 Script functions on page 15), however in JCL it is neater, and keeps the **PARM** string shorter, by using this DD name to locate it instead.

9.2.3 DD name MQEVLOG

The **MQEV** program needs to know where to write out its log files. This can be located using the **-L** program parameter, however in JCL it is neater, and keeps the **PARM** string shorter, by using this DD name to locate the library/directory instead. See 7 Logging on page 41 for more details about all the different ways to set your log path.

9.3 Running MQEV as a Started Task (z/OS only)

It may be appropriate in your z/OS environment to run the MQEV program as a started task.

A sample started task procedure is provided in the zip file, ***MQEVSTC.JCL***. Copy this to your procedure library and give it an appropriate name, for example ***xxxxMQEV*** (where xxxx is the name of your IBM MQ queue manager). The IBM-supplied procedure library is called ***SYS1.PROCLIB***, but your installation might use its own naming convention.

You will of course need to edit it to match your installation naming conventions for datasets.

```
//*****************************************************************
//* Run MQEV program                                              *
//*****************************************************************
//         PROC QMGR=MQG1
//PROCSTEP EXEC PGM=MQEV,
//         PARM=('-m &QMGR')
//*
//STEPLIB  DD DSN=MQGEM.LOAD,DISP=SHR
//         DD DSN=IBM.MQ.SCSQANLE,DISP=SHR
//         DD DSN=IBM.MQ.SCSQAUTH,DISP=SHR
//SYSPRINT DD SYSOUT=*
//SYSOUT   DD SYSOUT=*
//MQGEML   DD DSN=MQGEM.LICENCES,DISP=SHR
//MQEVMQX  DD DSN=MQGEM.MQEV.SCRIPTS(MQEV),DISP=SHR
//MQEVLOG  DD DSN=MQGEM.MQEV.LOGS.&QMGR,DISP=SHR
//
```

Having done this set-up, the **MQEV** program can be started using the MVS start command. The following example assumes the name of the member in your procedure library is ***MQG1MQEV***.

```
/S MQG1MQEV
```

or

```
/START MQG1MQEV
```

9.4 Stopping MQEV using the MVS STOP command

MQEV on z/OS can be stopped either

- by using the **`STOP EV`** command, sent to the `MQGEM.MQEV.COMMAND.QUEUE`, by using **MQSCX** or **MO71** (see Chapter 11.32 STOP EV on page 152 for a description of this command)
- or by using the MVS stop command. For example:

```
/P MQG1MQEV
```

or

```
/STOP MQG1MQEV
```

10 Returned Interval Times

When looking at accounting and statistics records using one of the following DISPLAY commands:-

- **`DISPLAY ACCTMQI`**
- **`DISPLAY ACCTQ`**
- **`DISPLAY STATCHL`**
- **`DISPLAY STATMQI`**
- **`DISPLAY STATQ`**

there are a number of different ways that the data can be totalled which will result in different times being returned as the start and end of intervals.

10.1 SUM(NONE)

If you select **`SUM(NONE)`** as a parameter, then no totalling takes place and each record returned as output on the DISPLAY command shows the start and end interval times as originally recorded by IBM MQ as it created the accounting or statistics record.

10.2 SUM(something) with no INTVL

In this example you are requesting a single record for each unique 'key' you specify in the SUM field. So, for example if you specify **`SUM(APPL,CHANNEL,CONNAME)`** then you are asking for one record to be returned for each unique combination of the given fields. It might be useful to think of **SUM(...)** as 'sum by'**.** This concept can be very useful to answer all sorts of questions. Consider the following options:

SUM() value	Meaning
`APPL`	Return a record for each Application Useful if you want to know what's doing the most messaging etc
`CHANNEL`	Return a unique record for each Channel Useful for determining which channels are used the most and perhaps which are no-longer used.
`USERID`	Return a record for each unique Userid Useful for seeing who is doing what to your Queue Manager
`RPRODUCT`	Sum by remote product identifier Useful to know what remote clients are connected to your Queue Manager. Are people using JMS, C, .Net Java etc
`APPL,RPRODUCT`	Similar to the one above but now you get a record for each Application/Product combination. This is useful for seeing which applications are using which clients.
`RVERSION`	Sum by remote version number Useful to know what version of MQ Client people are using. For example, are people still using an old out of date product
`APPL,RPRODUCT, RVERSION`	Similar to the one above but now you get a record for each Application/Product Version combination. This is useful for seeing which applications are which version of client.
`CONNAME`	Sum by remote IP address Useful to know how many machine are connecting to you and where from.

Of course there are many more combinations and each combination has the potential to show you something interesting.

The start and end interval times of each record returned will be dependent on the minimum and maximum times of the records within each record. This can be useful to get an indication of when something is happening, either a particular time of day or a range of times.

10.3 SUM(something) and INTVL(something)

This is very like the case above however you will get a record for each SUM combination **and** interval. So, for example, if you requested to get a days worth of data (using the FROM and TO parameters on the DISPLAY command) but specified an interval of 4 hours then you might receive up to 6 records for that 24 hour period, depending on the activity of that SUM combination.

Reducing the interval value increases the amount of results you receive but increases the resolution of the time when the activity happened. Needless to say you cannot increase the resolution beyond that which IBM MQ sent the messages of course. In other words, if MQ is issuing a statistics every 30 minutes then there is little point in requesting statistics with an interval lower than that. Accounting data is slightly different. If you have applications which are continually connecting and disconnecting then MQ will send records faster that the configured interval speed so having a low interval may well give you more frequent records than the configured accounting interval..

Using an interval has two main advantages:

1. You get a greater resolution of when something happened
2. You get a list of values which are graph-able

Consider the case where we have requested to see puts and gets for the last 24 hours with a 1 hour interval. **MQEV** will return us up to 24 records. Of course rather than displaying your data in a list you may wish to see the data in a graph. This can be very useful for seeing trends. There are two ways of doing this:

1. **You can bring up the MO71 graph and add the MQEV fields directly.**
   Please see the chapter on 'Graphing' in the **MO71** manual to see how this is done.
2. **Export your collected list as a CSV file**
   It is relatively straightforward in either **MQSCX** or in **MO71** to turn that set of data into a comma separated (CSV) file on disk. This CSV file can then be fed into a host of applications such as Excel which will display the data as a graph.

10.3.1 Graphing

If you have interest in graphing then there are two other options on the command which are useful.

10.3.1.1 GAPFILL

The **`INTVL`** parameters allows you to get a record for each interval over the range of dates you specify. However, by default if there is no activity during a particular interval then you will not get a record. As an example, suppose you issue a display for activity over the last week and ask for an interval of every 4 hours. Now suppose a particular queue was only used on Monday. You will not receive any records for Tuesday, Wednesday, Thursday etc. You only receive records in which there was at least *some* activity. This is great if the command is being issued to determine when the queue was being used. However, if you want to graph activity then you really want MQEV to give you records containing the '0' value so that it becomes a point on the graph. You can ask for this to happen by specifying something like **`GAPFILL(ACTIVITY)`**. **`GAPFILL`** has a number of options so use the one you need in order to get the effect you are after.

10.3.1.2 ZEROVALS

For efficiency MQEV will normally not send responses for counts that contain a zero value. For example, suppose you issue a command to determine what MQI calls have been issued. There seems little point in sending a response saying 'Number of Browses zero, Number of MQCB zero, Number of MQCMIT zero etc'. What the user generally wants to know are the non-zero values. I,e. What DID happen, not what DIDN'T happen. However, that is not so true if the results are being graphed. In those cases you probably do want an actual zero value sent to you. You can do this by adding **`ZEROVALS(SHOW)`** to your command.

11 Command Reference

This chapter describes, in alphabetic order, all the MQSC commands that can be issued to MQEV. In addition, each MQSC command description contains the details for using the same command via the PCF interface.

The MQEV event processor has the following sets of commands:

11.1 Programmable command format commands and responses

You can use **MQSCX** or **MO71** to administer **MQEV**. Alternatively, you can write your own application to send PCF format commands to the **MQEV** command server.

PCF commands and responses have a consistent structure including a PCF header (MQCFH) structure and any number of parameter structures of defined types, just in the same way as the IBM MQ command server. To learn more about the general structure and usage of the PCF format, please read Knowledge Center, starting at https://www.ibm.com/support/knowledgecenter/en/SSFKSJ_9.1.0/com.ibm.mq.ref.adm.doc/q088570_.htm

When sending PCF format messages to the MQEV command server, the following points should be noted in addition to that described in IBM Knowledge Center.

- The `MQCFH Type` field must contain the value `MQCFT_COMMAND.`
- The `MQCFH Version` field must be `MQCFH_VERSION_1`
- The constants described in this chapter to be used in PCF format messages sent to the **MQEV** command server that begin `MQG_` are defined in `mqev.h` which can be found in your zip/tar file.
- Unlike the IBM MQ command server, responses are not sent as individual messages, but instead are gathered together with many responses to the command in a single reply message. This is a more efficient way to use MQ, but is different from the way you may have processed PCF response messages in other applications. In order to determine where the end of one response ends and another response begins, look for an `MQCFIN` parameter as described below.
- The `WHERE` (identifier: `MQG_ATTR_WHERE`) and `PREWHERE` (identifier: `MQG_ATTR_PREWHERE`) parameters are very different to the IBM MQ command server's use of `WHERE`. These are simple string parameters which contain the MQSC style `WHERE` value. **MQEV** does not make use of the PCF filter types, `MQCFIF`, `MQCFBF` or `MQCFSF`.
- 64-bit integer values (`MQCFIN64`) may be provided as 32-bit integers (`MQCFIN`) if the value to be conveyed to the command server is small enough to fit in a 32-bit integer field.
- The `REPLACE` (identifier `MQIACF_REPLACE`) parameter does not have a negative equivalent attribute `NOREPLACE` in **MQEV**. If the `REPLACE` parameter is omitted, `NOREPLACE` is assumed.
- Some parameters may be represented by an `MQCFIN` or an `MQCFIL` depending on whether one, or more than one value needs to be conveyed to the command server. This is described fully on the parameters where it is applicable.

ResponseSeparator (MQCFIN)

The response separator (parameter identifier: `MQG_ATTR_RESPONSE_DELIMETER`).

This parameter is the separator between responses in a reply message from the **MQEV** command server.

The value is always set to zero, and conveys no meaning.

In the pages that follow, the PCF constants needed for each command, and its responses are described alongside the equivalent MQSC flavour of the command.

11.2 ADD EVALERT

Use the MQSC command **ADD EVALERT** (or it's equivalent PCF command **MQG_CMD_ADD_EV_ALERT**) to create an alert. Alerts can be used as reminders or notifications. Learn more about alerts in Chapter 12 Alerts on page 153.

A log file entry will be written by this command showing the details of the alert that was added.

11.2.1 Syntax diagram for ADD EVALERT

```
                                        ┌─ CATEGORY( ' ' ) ──┐
►►─ ADD EVALERT ── TEXT( string ) ──┤                    ├──────────────►
                                        └─ CATEGORY( string ) ─┘

   ┌─ EVENTID( ' ' ) ──┐   ┌─ EVOBJNAME( ' ' ) ──┐   ┌─ EVQMGR( ' ' ) ──┐
►──┤                   ├───┤                     ├───┤                  ├──►
   └─ EVENTID( integer ) ─┘   └─ EVOBJNAME( string ) ─┘   └─ EVQMGR( string ) ─┘

   ┌─ EVOBJTYPE( NONE ) ──────────┐   ┌─ SEVERITY( ERROR ) ──────────┐
►──┤                              ├───┤                              ├──────►
   └─ EVOBJTYPE( ─┬─ AUTHINFO ─┬─ )─┘   └─ SEVERITY( ─┬─ TERM ──────┬─ ) ─┘
                  ├─ AUTHREC ──┤                       ├─ SEVERE ─┤
                  ├─ CFSTRUCT ─┤                       ├─ WARN ───┤
                  ├─ CHANNEL ──┤                       └─ INFO ───┘
                  ├─ CHLAUTH ──┤
                  ├─ CLNTCONN ─┤
                  ├─ COMMINFO ─┤
                  ├─ LISTENER ─┤
                  ├─ NAMELIST ─┤
                  ├─ PROCESS ──┤
                  ├─ QUEUE ────┤
                  ├─ QMGR ─────┤
                  ├─ RQMNAME ──┤
                  ├─ SERVICE ──┤
                  ├─ SUB ──────┤
                  ├─ STGCLASS ─┤
                  ├─ TOPIC ────┤
                  └─ TOPICSTR ─┘

   ┌─ RETINTVL( ev-default ) ──┐   ┌─ EVSTREAM( ' ' ) ──┐
►──┤                           ├───┤                    ├──┬──────────┬──►◄
   └─ RETINTVL( integer ) ─────┘   └─ EVSTREAM( string ) ─┘  └─ REPLACE ─┘
```

11.2.2 Parameter descriptions for ADD EVALERT

CATEGORY

The category of the alert. A category can be any string which conveniently groups different alerts of the same type. The maximum length of this string is MQG_ALERT_CATEGORY_LENGTH (10).The default value is blank. Users should not use categories starting with a $ (dollar) sign. These are reserved for system use. Currently the following alert categories may be generated by **MQEV**.

- **$LICENCE** Alerts to inform the user of impending licence expiry
- **$VERSION** Alerts to inform the user that a new version of MQEV exists.
- **$STORM** Alerts to inform the user that an event storm has been detected.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_ALERT_CATEGORY

EVENTID

The unique ID of the event this alert is about. The default value is blank.

If **`ADD EVALERT`** is called from inside the `MQEVEvent` function, and this is left blank, then this is automatically filled-in with the EVENTID of the current event being processed (assuming the function does not choose to discard the event). Learn more about event functions in Chapter 14 MQEV Scripting on page 159.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_EVENT_ID.

EVOBJNAME

The object name that this alert refers to. The default value is blank.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_OBJECT.

EVOBJTYPE

The object type that this alert refers to. The default value is blank.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_OBJECT_TYPE.

Possible Values are:-

MQSC value	Meaning	PCF constant
AUTHINFO	Authentication information object	`MQOT_AUTH_INFO`
AUTHREC	Authorization records	`MQOT_AUTH_REC`
CFSTRUCT	CF Structure	`MQOT_CF_STRUC`
CHANNEL	Channel	`MQOT_CHANNEL`
CHLAUTH	Channel Authentication records	`MQOT_CHLAUTH`
CLNTCONN	Client connection channel	`MQOT_CLNTCONN_CHANNEL`
COMMINFO	Communication information object	`MQOT_COMM_INFO`
LISTENER	Listener	`MQOT_LISTENER`
NAMELIST	Namelist	`MQOT_NAMELIST`
NONE	None. This is the default value.	`MQOT_NONE`
PROCESS	Process	`MQOT_PROCESS`
QUEUE	Queue	`MQOT_Q`
QMGR	Queue manager	`MQOT_Q_MGR`
RQMNAME	Remote queue manager	`MQOT_REMOTE_Q_MGR_NAME`
SERVICE	Service object	`MQOT_SERVICE`
STGCLASS	Storage Class	`MQOT_STORAGE_CLASS`
SUB	Subscription	`MQG_OT_SUB`
TOPIC	Topic	`MQOT_TOPIC`
TOPICSTR	Topic String	`MQG_OT_TOPICSTR`

EVQMGR

The queue manager to which this alert is associated. The maximum length of this string is MQ_Q_MGR_NAME_LENGTH (48). The default value is blank.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_ALERT_Q_MGR.

REPLACE

Whether an existing alert is to be replaced with this one. This is optional. The default is not to replace the alert. When specified the alert replaces an existing one with the following matching attributes.

- TEXT
- CATEGORY
- EVQMGR
- EVOBJECT
- EVOBJTYPE

If a matching alert does not exist, one is created.

RETINTVL

The retention interval, in seconds, for this alert. The default value is the **ALERTRET** value on the EV object converted into seconds.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_RETENTION_INTERVAL.

SEVERITY

The severity of the alert.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_ALERT_SEVERITY.

Possible values are:-

MQSC value	Meaning	PCF constant
TERM	Termination	MQG_SEVERITY_TERM
SEVERE	Severe Error	MQG_SEVERITY_SEVERE
ERROR	Error. This is the default value	MQG_SEVERITY_ERROR
WARN	Warning	MQG_SEVERITY_WARN
INFO	Information	MQG_SEVERITY_INFO

EVSTREAM

The stream to which this alert is associated. The maximum length of this string is MQG_STREAM_NAME_LENGTH (64). The default value is blank.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_STREAM_NAME.

TEXT

The text of the alert. The maximum length of this string is MQG_ALERT_TEXT_LENGTH (1024). This attribute must be specified.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_ALERT_TEXT.

11.3 ADD EVQ

Use the MQSC command **`ADD EVQ`** (or it's equivalent PCF command **`MQG_CMD_ADD_EV_Q`**) to add **MQEV** details regarding a queue that is to be processed by the **MQEV** event processor.

A log file entry will be written by this command showing the queue name that was added.

N.B. If you attempt to `ADD` a queue that is already there (with the `REPLACE` keyword), it will succeed as if you had done an `ALTER`.

11.3.1 Syntax diagram for ADD EVQ

```
                                 ┌─ DESCR( ' ' ) ──────┐  ┌─ FWDPSIST( ASMSG ) ──────┐
►►─ ADD EVQ( queue-name ) ───────┼─────────────────────┼──┼──────────────────────────┼──►
                                 └─ DESCR( string ) ───┘  └─ FWDPSIST( ┬─ YES ─┬─ ) ─┘
                                                                      └─ NO ──┘
  ┌─ FWDQ( ' ' ) ──────┐  ┌─ TEMPQ( DISCARD ) ─┐
►─┼────────────────────┼──┼────────────────────┼──┬──────────────────────────────────►◄
  └─ FWDQ( string ) ───┘  └─ TEMPQ( STORE ) ───┘  └─ REPLACE ─┘
```

11.3.2 Parameter descriptions for ADD EVQ

(queue-name)

The name of the IBM MQ event queue being added to MQEV for processing. The maximum length of this string is MQ_Q_NAME_LENGTH (48). If a `FWDQ` is specified, this queue will be opened using MQOO_SAVE_ALL_CONTEXT.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_EVENT_Q_NAME.

DESCR

A description of the event queue. The maximum length of this string is MQ_Q_DESC_LENGTH (64).

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_EVENT_Q_DESC.

FWDQ

The queue name event messages should be forwarded to for daisy-chaining purposes. The maximum length of this string is MQ_Q_NAME_LENGTH (48). If provided this queue will be opened with MQOO_PASS_ALL_CONTEXT.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_FORWARD_Q_NAME.

FWDPSIST

The persistence that should be used for event messages that are forwarded to the queue named in the `FWDQ` attribute.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_FORWARD_PERSISTENCE.

Possible Values are:-

ASMSG

Forwarded messages have the same persistence as the original event message. This is the initial value.
The PCF value for this is MQG_PERSISTENCE_AS_MESSAGE.

YES

Forwarded message are persistent.
The PCF value for this is MQG_PERSISTENCE_YES.

NO

Forwarded messages are non-persistent.
The PCF value for this is MQG_PERSISTENCE_NO.

TEMPQ

How statistics records about temporary queues are handled.

It is only currently possible to determine when a queue is temporary through the statistics records. For accounting and events, it is not possible to tell and therefore this attribute will have no effect.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_TEMPQ_DISP.

Possible Values are:-

DISCARD

Records about temporary queues are discarded. This is the initial value.
The PCF value for this is MQG_TEMPQ_DISCARD.

STORE

Records about temporary queues are stored.
The PCF value for this is MQG_TEMPQ_STORE.

REPLACE

Whether the existing event queue configuration is to be replaced with this one. This is optional. The default is not to replace the event queue configuration. When specified the event queue replaces the existing one with the same name. If a matching event queue configuration does not exist, one is added.

When using the PCF interface, this is an MQCFIN parameter with identifier MQIACF_REPLACE.

11.4 ALTER EV

Use the MQSC command **`ALTER EV`** (or it's equivalent PCF command **`MQG_CMD_ALTER_EV`**) to change the main overall configuration of the **MQEV** event processor.

A log file entry will be written by this command showing the alteration that was made.

11.4.1 Syntax diagram for ALTER EV

```
►►─ ALTER EV ──┬──────────────────────────┬──┬──────────────────────────┬──►
               └─ ALERTRET( integer ) ─┘  └─ DEADQ( string ) ─┘

►──┬──────────────────────────┬──────────────────────────────────────────►
   └─ DEFMAXRECS( integer ) ─┘

►──┬──────────────────────────┬──┬──────────────────────────────────────►
   └─ DESCR( string ) ─┘  └─ DISPCMDS( ─┬─ DISCARD ─┬─ ) ─┘
                                        ├─ HIDE ────┤
                                        └─ SHOW ────┘
►──┬──────────────────────────┬──┬──────────────────────────┬──────────►
   └─ LICREMIND( integer ) ─┘  └─ LOGRET( integer ) ─┘

►──┬──────────────────────────┬──┬──────────────────────────┬──────────►
   └─ MAXLOGAGE( integer ) ─┘  └─ MAXLOGSZ( integer ) ─┘

►──┬──────────────────────┬──┬──────────────────────────┬──┬──────────────────────────┬──►
   └─ NAME( string ) ─┘  └─ STORMINT( integer ) ─┘  └─ STORMTHR( integer ) ─┘

►──┬──────────────────────────────────────┬──┬──────────────────────────────────┬──►◄
   └─ USEDLQ( ─┬─ YES ───────┬─ ) ─┘  └─ VERCHECK( ─┬─ BUILD ───┬─ ) ─┘
               ├─ NO ────────┤                      ├─ VERSION ─┤
               └─ IFAVAIL ───┘                      └─ NO ──────┘
```

11.4.2 Parameter descriptions for ALTER EV

ALERTRET

The default retention interval, in days, for alerts. The initial value for this is 14 days.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_DEF_ALERT_RETENTION_INTERVAL.

DEADQ

The name of the dead-letter queue to use. If this is blank, the queue manager defined DEADQ is used. The initial value for this field is blank. The maximum length of this string is MQ_Q_NAME_LENGTH (48).

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_DEAD_LETTER_QUEUE.

DEFMAXRECS

The default number of source records which should be used to satisfy any `DISPLAY` command for Accounting or Statistics data. This can prevent inadvertent consumption of CPU when issuing queries against large amounts of data. The default value for this field is 1,000,000 (1 million).

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_DEFAULT_MAX_RECORDS.

DESCR

A text description of this MQEV instance. The maximum length of this string is MQG_EV_DESC_LENGTH (64).

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_EV_DESC.

DISPCMDS

How to handle command events received that record DISPLAY commands. IBM MQ will only produce events for **`DISPLAY`** commands if the Queue Manager `CMDEV` attribute is set to `ENABLED` rather than `NODISPLAY`.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_DISPLAY_COMMANDS.

Possible Values are:-

DISCARD

Don't store any DISPLAY command events. This is the initial value.
The PCF value for this is MQG_DISPCMDS_DISCARD.

HIDE

Store but don't show any DISPLAY command events by default. This can be over-ridden on the DISPLAY EVENTS command.
The PCF value for this is MQG_DISPCMDS_HIDE.

SHOW

Store and show DISPLAY command events.
The PCF value for this is MQG_DISPCMDS_SHOW.

LICREMIND

The time left on your MQEV licence, in days, after which you will begin to get reminders. Initially you will receive weekly reminders, but once you have only 7 days left, then you will receive daily reminders. These reminders will be in the form of alerts. The initial value for this is 60 days.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_LICENCE_REMIND_TIME.

LOGRET

The retention interval, in days, for MQEV log files[12]. The initial value is 7.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_LOG_RETENTION_INTERVAL.

MAXLOGAGE

The maximum age, in minutes, of an MQEV log file. A new log file will be created if the current log file reaches this age. The initial value for this is 240 minutes.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_MAX_LOG_AGE.

12 Note that MQEV will not delete log files created by a previous instance of MQEV.

MAXLOGSZ

The maximum size, in kilobytes, of an MQEV log file. A new log file will be created if the current log file reaches this size. This initial value for this is 10000.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_MAX_LOG_SIZE.

NAME

The name of this MQEV instance. The maximum length of this string is MQG_EV_NAME_LENGTH (8).

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_EV_NAME.

STORMINT

The time period, in seconds, within which a number of identical events are received (configured by STORMTHR) before it is considered to be an event storm. The initial value for this is 60 seconds.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_STORM_INTERVAL.

STORMTHR

The number of identical events received in a time period (configured by STORMINT) before it is considered to be an event storm. The initial value for this is 20.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_STORM_THRESHOLD.

USEDLQ

Whether the Dead-letter queue should be used for unprocessed messages.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_USE_DEAD_LETTER_Q.

Possible values are:-

YES

The dead-letter queue, if named on either the EV object or on the IBM MQ QMgr object, is used and any failure to place a dead-lettered message on the queue will cause the MQEV program to end.
The PCF value for this is MQG_USEDLQ_YES.

NO

The dead-letter queue is not used. Messages found on the queues MQEV is processing, which are not events or command messages, are forwarded (if used) and discarded.
The PCF value for this is MQG_USEDLQ_NO.

IFAVAIL

The dead-letter queue is used but any failure to place a dead-lettered message on the queue will not cause the MQEV program to end and instead the message will be forwarded (if used) and discarded. This is the initial value.
The PCF value for this is MQG_USEDLQ_IF_AVAIL.

VERCHECK

MQEV can check for newer versions of the product, and provide an alert when a newer version is available.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_VERSION_CHECK.

Possible values are:-

NO

No check for newer versions of the MQEV product is made.
The PCF value for this is MQG_VERSION_CHECK_NO.

VERSION

A check is made for newer versions of the MQEV product, but not for newer build dates of the current version.
The PCF value for this is MQG_VERSION_CHECK_VERSION.

BUILD

A check is made for newer versions and newer build dates of the MQEV product. This is the initial value.
The PCF value for this is MQG_VERSION_CHECK_BUILD.

11.5 ALTER EVEMIT

Use the MQSC command **`ALTER EVEMIT`** (or it's equivalent PCF command **`MQG_CMD_ALTER_EV_EMIT`**) to change an emitter object.

A log file entry will be written by this command showing the alteration that was made.

N.B. If you attempt to ALTER an emitter object that doesn't exist, it will fail.

11.5.1 Syntax diagram for ALTER EVEMIT

```
►►─ ALTER EVEMIT( emit-object-name ) ──┬──────────────────────────────────────────────────►
                                       ├─ QUEUE( queue-name ) ─ FILE( ' ' ) ──┤
                                       └─ FILE( file-name ) ── QUEUE( ' ' ) ──┘

►──┬──────────────────────────┬──┬──────────────────────────────────────────────────────►
   └─ DESCR( string ) ────────┘  └─ FORMAT( ──┬─ JSON ─┬─ ) ─┘
                                              ├─ MQSC ─┤
                                              └─ CSV ──┘

►──┬──────────────────────────────────────────┬──┬──────────────────────────────────────►
   └─ GROUPING( ──┬─ ORIGINAL ─┬─ ) ─┘        └─ MAXSIZE( kilobytes ) ─┘
                  └─ SINGLE ────┘

►──┬──────────────────────────────────────────┬──┬──────────────────────────────────────►
   └─ MSGPERS( ──┬─── ASMSG ───┬─ ) ─┘        └─ ONERROR( ──┬─ BACKOUT ─┬─ ) ─┘
                 ├─── NO ──────┤                             └─ SUSPEND ─┘
                 └─── YES ─────┘

►──┬──────────────────────────────────────────┬──┬──────────────────────────────────────►
   └─ PUT( ──┬─ NORMAL ────┬─ ) ─┘           └─ READER( ──┬─ HUMAN ────┬─ ) ─┘
             └─ IMMEDIATE ─┘                             └─ PROGRAM ───┘

►──┬──────────────────────────────────────────┬──────────────────────────────────────────►◄
   └─ ZEROVALS( ──┬─ HIDE ─┬─ ) ─┘
                  └─ SHOW ─┘
```

11.5.2 Parameter descriptions for ALTER EVEMIT

(emit-object-name)

The name of the emitter object whose configuration is to be altered. The maximum length of this string is MQG_EMIT_NAME_LENGTH (48).

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_EMIT_NAME.

QUEUE

The queue to emit data to. Only one of QUEUE or FILE can be specified. If you are altering an emitter object that currently has a file name defined, you must blank out that attribute at the same time as setting the queue name. The maximum length of this string is MQ_Q_NAME_LENGTH (48).

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_EMIT_Q_NAME.

FILE

The file name to emit data to. Only one of QUEUE of FILE can be specified. If you are altering an emitter object that currently has a queue name defined, you must blank out that attribute at the same time as setting the file name. The maximum length of this string is MQG_FILE_NAME_LENGTH (300).

The file name can contain inserts to ensure each new file name is unique. These inserts are described in 6.4.1 Emitter Filename Inserts on page 37.

On z/OS, file names must be provided directly, and not use DD names, and must be z/OS UNIX files.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_EMIT_FILE_NAME.

DESCR

A text description of the emit object. The maximum length of this string is MQ_APPL_DESC_LENGTH (64).

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_EMIT_DESC.

FORMAT

The format that the data will be emitted in.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_EMIT_FORMAT.

Possible values are:-

JSON

Data will be emitted in JSON format.
The PCF value for this is MQG_FORMAT_JSON.

NDJSON

Data will be emitted in Newline Delimited JSON format.
The PCF value for this is MQG_FORMAT_NDJSON.

MQSC

Data will be emitted in MQSC format.
The PCF value for this is MQG_FORMAT_MQSC.

CSV

Data will be emitted in comma separated values (CSV) format.
The PCF value for this is MQG_FORMAT_CSV.

GROUPING

Accounting Queue, Statistic Queue and Statistics Channel messages from MQ may contain multiple object information in a single message. This is done for efficiency reasons since a separate message for each object would clearly require a lot more processing. You can configure the emitter to continue this grouping or to split each object into a separate emission. It is more efficient to maintain the grouping but it may be that the downstream application can only deal with one object at a time.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_EMIT_GROUPING.

Possible values are:-

ORIGINAL

The grouping provided by MQ is maintained.
The PCF value for this is MQG_EMIT_GROUPING_ORIGINAL.

SINGLE

Output will be split into separate outputs if using a queue as output or a file using the %i insert.
The PCF value for this is MQG_EMIT_GROUPING_SINGLE.

MAXSIZE

For a queue emitter, this is the maximum size of emitted message written to the queue.

For a file emitter, this is the maximum size of the file, after which point a new file will be created with a unique name.

The size is measured in kilobytes.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_MAX_SIZE.

MSGPERS

The persistence of messages written to the queue. This attribute only applies when the QUEUE attribute is used.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_MSG_PERSISTENCE.

Possible values are:

ASMSG

Messages have the same persistence as the original event message.
The PCF value for this is MQG_PERSISTENCE_AS_MESSAGE.

YES

Messages are persistent.
The PCF value for this is MQG_PERSISTENCE_YES.

NO

Messages are non-persistent
The PCF value for this is MQG_PERSISTENCE_NO.

ONERROR

How to behave if an attempt to write to the queue or file specified fails. If MQEV is configured to use the Dead Letter Queue, then this is considered a success and the behaviour described here does not apply. If the DLQ is not being used, or if writing to the DLQ fails, then the behaviour described below will apply.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_ON_ERROR.

Possible values are:-

SUSPEND

The message read from the EVQ is backed out and the EVQ is suspended. An alert is raised to indicate the problem. The ERRORCT value will be incremented each time this happens. Once the issue has been rectified, the administrator should use the RESUME EVQ command to indicate to MQEV that the queue can be processed again.
The PCF value for this is MQG_ONERROR_SUSPEND.

DISCARD

The emitted message, which could not be written to the queue or file, is discarded. An alert is raised to indicate the problem. This will result in gaps in the emitted stream of data but will not cause the suspension of event processing. The ERRORCT value will be incremented for each message that is discarded in this way.
The PCF value for this is MQG_ONERROR_DISCARD.

PUT

The transactionality used when putting messages to the emitter queue. This attribute is ignored if this emitter is writing to a file.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_PUT_TRANSACT.

Possible values are:-

NORMAL

The messages put to the emitter queue are part of the same transaction as writing the data to the MQEV data queue. If any failure occurs and this transaction is rolled back, so are the messages put to the emitter queue. This means that no duplicate messages can occur on the emitter queue, but also means that the availability of messages on the emitter queue is no sooner than the completion of the transaction processing the event message data stored to the MQEV data queue.
The PCF value for this is MQG_PUT_NORMAL.

IMMEDIATE

The messages put to the emitter queue are put outside of a transaction and are immediately available to be consumed and posted elsewhere. While this improves the availability of these messages, there is the possibility of duplicate messages on this queue should the transaction writing the event message data to the MQEV data queue be rolled back and re-processed. See 6.3.3.1 Unique ID on page 36 for more information about handling these possible duplicates.
The PCF value for this is MQG_PUT_IMMEDIATE.

READER

Whether the data is compressed or retains all the spaces and newlines to be more “human readable”.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_READER.

Possible values are:

PROGRAM

The data is designed to be read by a computer program. It is compressed and all extraneous newlines and spaces are removed. The data is all on one line.
The PCF value for this is MQG_READER_PROGRAM.

HUMAN

The data is designed to be human readable and contains newlines and spaces that are not essential but make the layout easier to read for human beings.
The PCF value for this is MQG_READER_HUMAN.

ZEROVALS

How to handle zero or empty values in the messages emitted.
This parameter does not apply for emitters of type CSV.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_ZERO_VALUES.

Possible values are:-

HIDE

Do not include any zero values or empty strings in the messages.
The PCF value for this is MQG_ZEROVALS_HIDE.

SHOW

Include zero values and empty strings in the messages.
The PCF value for this is MQG_ZEROVALS_SHOW

11.6 ALTER EVQ

Use the MQSC command **`ALTER EVQ`** (or it's equivalent PCF command **`MQG_CMD_ALTER_EV_Q`**) to change the MQEV details regarding a queue that is being processed by the MQEV event processor.

A log file entry will be written by this command showing the alteration that was made.

N.B. If you attempt to ALTER a queue that doesn't exist, it will fail.

11.6.1 Syntax diagram for ALTER EVQ

```
►►─ ALTER EVQ( queue-name ) ─┬──────────────────────┘ └──────────────────────┘──►
                             └─ DESCR( string ) ─┘    └─ FWDQ( string ) ─┘

►──┬──────────────────────────────────────┘ └──────────────────────────────────┘──►◄
   └─ FWDPSIST( ─┬─ ASMSG ─┬─ ) ─┘          └─ TEMPQ( ─┬─ DISCARD ─┬─ ) ─┘
                 ├─ YES ───┤                           └─ STORE ───┘
                 └─ NO ────┘
```

11.6.2 Parameter descriptions for ALTER EVQ

(queue-name)

The name of the event queue whose configuration is to be altered. The maximum length of this string is MQ_Q_NAME_LENGTH (48).

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_EVENT_Q_NAME.

DESCR

A description of the queue being processed by MQEV. The maximum length of this string is MQ_Q_DESC_LENGTH (64).

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_EVENT_Q_DESC.

FWDPSIST

The persistence that should be used for messages that are forwarded to the queue named in the `FWDQ` attribute.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_FORWARD_PERSISTENCE.

Possible Values are:-

ASMSG

Forwarded messages have the same persistence as the original message.
The PCF value for this is MQG_PERSISTENCE_AS_MESSAGE.

YES

Forwarded message are persistent.
The PCF value for this is MQG_PERSISTENCE_YES.

NO

Forwarded messages are non-persistent.
The PCF value for this is MQG_PERSISTENCE_NO.

FWDQ

The queue name messages should be forwarded to for daisy-chaining purposes. The maximum length of this string is MQ_Q_NAME_LENGTH (48). If provided this queue will be opened with MQOO_PASS_ALL_CONTEXT.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_FORWARD_Q_NAME.

TEMPQ

How statistics records about temporary queues are handled.

It is only currently possible to determine when a queue is temporary through the statistics records. For accounting and events, it is not possible to tell and therefore this attribute will have no effect.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_TEMPQ_DISP.

Possible Values are:-

DISCARD

Records about temporary queues are discarded.
The PCF value for this is MQG_TEMPQ_DISCARD.

STORE

Records about temporary queues are stored.
The PCF value for this is MQG_TEMPQ_STORE.

11.7 ALTER EVSTREAM

Use the MQSC command **`ALTER EVSTREAM`** (or it's equivalent PCF command **`MQG_CMD_ALTER_EV_STREAM`**) to change a stream.

Streams are used to store the records (events, accounting and statistics) processed by **MQEV**.

A log file entry will be written by this command showing the stream that was changed.

11.7.1 Syntax diagram for ALTER EVSTREAM

```
►►─ ALTER EVSTREAM( stream-name ) ──┬──────────────────────────────────────────────────────►
                                    └── TYPE( ─┬─ EVENTS ─┬─ ) ─┘
                                               ├─ STATQ ──┤
                                               ├─ STATCHL ─┤
                                               ├─ STATMQI ─┤
                                               ├─ ACCTQ ──┤
                                               └─ ACCTMQI ─┘

►──┬──────────────────────────────────────┬──┬──────────────────────────────────────┬──────►
   └── DEFAULT( ─┬─ YES ─┬─ ) ─┘           └── DESCR( string ) ─┘
                 └─ NO ──┘

►──┬──────────────────────────────────────┬──┬──────────────────────────────────────┬──►◄
   └── EVEMIT( emit-object-name ) ─┘           └── RETINTVL( integer ) ─┘
```

11.7.2 Parameter descriptions for ALTER EVSTREAM

(stream-name)

The name of the stream to be altered. The maximum length of this string is MQG_STREAM_NAME_LENGTH (64).

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_STREAM_NAME.

TYPE

The type of data that is stored on this stream. This attribute is optional, and only required if the stream name is not a unique reference to the stream object being altered.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_STREAM_TYPE.

Possible values are:-

EVENTS

This stream contains event data.
The PCF value for this is MQG_STREAM_TYPE_EVENTS.

STATQ

This stream contains queue statistics data.
The PCF value for this is MQG_STREAM_TYPE_STAT_Q.

STATCHL

This stream contains channel statistics data.
The PCF value for this is MQG_STREAM_TYPE_STAT_CHL.

STATMQI

This stream contains MQI statistics data.
The PCF value for this is MQG_STREAM_TYPE_STAT_MQI.

ACCTQ

This stream contains queue accounting data.
The PCF value for this is MQG_STREAM_TYPE_ACCT_Q.

ACCTMQI

This stream contains MQI accounting data.
The PCF value for this is MQG_STREAM_TYPE_ACCT_MQI.

DEFAULT

Whether this stream is to be the default stream. Only one stream of each type has the value DEFAULT(YES). If you alter another stream to have DEFAULT(YES), the current default stream for that type is changed to DEFAULT(NO).

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_DEFAULT_STREAM.

Possible values are:-

YES

This stream is the default stream.
The PCF value for this is MQG_DEFAULT_YES.

NO

This stream is not the default stream.
The PCF value for this is MQG_DEFAULT_NO.

DESCR

A text description of the stream. The maximum length of this string is MQ_APPL_DESC_LENGTH (64).

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_STREAM_DESC.

EVEMIT

The name of an EVEMIT object which controls how data on this stream is additionally emitted. If this field is blank it means data is not emitted in any format. The maximum length of this string is MQG_EMIT_NAME_LENGTH.

The named emit object must exist or the ALTER command will fail.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_EMIT_NAME.

RETINTVL

The retention interval for data on this stream (in days).

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_RETENTION_INTERVAL.

11.8 DEFINE EVEMIT

Use the MQSC command **DEFINE EVEMIT** (or it's equivalent PCF command **MQG_CMD_DEFINE_EV_EMIT**) to create an emitter object.

A log file entry will be written by this command showing the emitter that was created.

11.8.1 Syntax diagram for DEFINE EVEMIT

```
►►─ DEFINE EVEMIT( emit-object-name ) ─┬─ QUEUE( queue-name ) ─┬──────────────────►
                                       └─ FILE( file-name ) ───┘   └─ REPLACE ─┘

   ┌─ DESCR( ' ' ) ──────┐                          ┌─ GROUPING( ORIGINAL ) ─┐
►──┤                     ├──FORMAT( ┬─ JSON ─┬ )──┤                        ├──────────►
   └─ DESCR( string ) ───┘          ├─ MQSC ─┤      └─ GROUPING( SINGLE ) ──┘
                                    └─ CSV ──┘

   ┌─ MAXSIZE( 100 ) ──────┐   ┌─ MSGPERS( ASMSG ) ─┐   ┌─ ONERROR( SUSPEND ) ─┐
►──┤                       ├───┤                    ├───┤                      ├──────►
   └─ MAXSIZE( kilobytes ) ─┘   ├─ MSGPERS( YES ) ───┤   └─ ONERROR( DISCARD ) ─┘
                                └─ MSGPERS( NO ) ────┘

   ┌─ PUT( NORMAL ) ──────┐   ┌─ READER ( PROGRAM ) ─┐   ┌─ ZEROVALS( HIDE ) ─┐
►──┤                      ├───┤                      ├───┤                    ├──────►◄
   └─ PUT( IMMEDIATE ) ───┘   └─ READER ( HUMAN ) ───┘   └─ ZEROVALS( SHOW ) ─┘
```

11.8.2 Parameter descriptions for DEFINE EVEMIT

(emit-object-name)

The name of the emit object to be created. The maximum length of this string is MQG_EMIT_NAME_LENGTH

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_EMIT_NAME (48).

QUEUE

The queue to emit data to. Only one of QUEUE or FILE can be specified.

The maximum length of this string is MQ_Q_NAME_LENGTH (48).

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_EMIT_Q_NAME.

FILE

The file name to emit data to. Only one of QUEUE of FILE can be specified.

The maximum length of this string is MQG_FILE_NAME_LENGTH (300).

The file name can contain inserts to ensure each new file name is unique. These inserts are described in 6.4.1 Emitter Filename Inserts on page 37.

On z/OS, file names must be provided directly, and not use DD names, and must be z/OS UNIX files.

When using the PCF interface, this is an MQCFST parameter with identified MQG_ATTR_EMIT_FILE_NAME.

DESCR

A text description of the emit object. The maximum length of this string is MQ_APPL_DESC_LENGTH (64).

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_EMIT_DESC.

FORMAT

The format that the data will be emitted in.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_EMIT_FORMAT.

There is no default value for this parameter, it must be specified at creation time. Possible values are:-

JSON

Data will be emitted in JSON format.
The PCF value for this is MQG_FORMAT_JSON.

NDJSON

Data will be emitted in Newline Delimited JSON format.
The PCF value for this is MQG_FORMAT_NDJSON.

MQSC

Data will be emitted in MQSC format.
The PCF value for this is MQG_FORMAT_MQSC.

CSV

Data will be emitted in comma separated values (CSV) format.
The PCF value for this is MQG_FORMAT_CSV.

GROUPING

Accounting Queue, Statistic Queue and Statistics Channel messages from MQ may contain multiple object information in a single message. This is done for efficiency reasons since a separate message for each object would clearly require a lot more processing. You can configure the emitter to continue this grouping or to split each object into a separate emission. It is more efficient to maintain the grouping but it may be that the downstream application can only deal with one object at a time.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_EMIT_GROUPING.

Possible values are:-

ORIGINAL

The grouping provided by MQ is maintained.
The PCF value for this is MQG_EMIT_GROUPING_ORIGINAL.

SINGLE

Output will be split into separate outputs if using a queue as output or a file using the %i insert.
The PCF value for this is MQG_EMIT_GROUPING_SINGLE.

MAXSIZE

The maximum size of an emitted message written to the queue or file, in kilobytes.

The default value is 100 kilobytes

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_MAX_SIZE

MSGPERS

The persistence of messages written to the queue. This attribute only applies when the QUEUE attribute is used.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_MSG_PERSISTENCE.

Possible values are:

ASMSG

Messages have the same persistence as the original event message. This is the default value.
The PCF value for this is MQG_PERSISTENCE_AS_MESSAGE.

YES

Messages are persistent.
The PCF value for this is MQG_PERSISTENCE_YES.

NO

Messages are non-persistent
The PCF value for this is MQG_PERSISTENCE_NO.

ONERROR

How to behave if an attempt to write to the queue or file specified fails. If MQEV is configured to use the Dead Letter Queue, then this is considered a success and the behaviour described here does not apply. If the DLQ is not being used, or if writing to the DLQ fails, then the behaviour described below will apply.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_ON_ERROR.

Possible values are:-

SUSPEND

The message read from the EVQ is backed out and the EVQ is suspended. An alert is raised to indicate the problem. The ERRORCT value will be incremented each time this happens. Once the issue has been rectified, the administrator should use the RESUME EVQ command to indicate to MQEV that the queue can be processed again.
The PCF value for this is MQG_ONERROR_SUSPEND.

DISCARD

The emitted message, which could not be written to the queue or file, is discarded. An alert is raised to indicate the problem. This will result in gaps in the emitted stream of data but will not cause the suspension of event processing. The ERRORCT value will be incremented for each message that is discarded in this way.
The PCF value for this is MQG_ONERROR_DISCARD.

PUT

The transactionality used when putting messages to the emitter queue. This attribute is ignored if this emitter is writing to a file.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_PUT_TRANSACT.

Possible values are:-

NORMAL

The messages put to the emitter queue are part of the same transaction as writing the data to the MQEV data queue. If any failure occurs and this transaction is rolled back, so are the messages put to the emitter queue. This means that no duplicate messages can occur on the emitter queue, but also means that the availability of messages on the emitter queue is no sooner than the completion of the transaction processing the event message data stored to the MQEV data queue. This is the default value.
The PCF value for this is MQG_PUT_NORMAL.

IMMEDIATE

The messages put to the emitter queue are put outside of a transaction and are immediately available to be consumed and posted elsewhere. While this improves the availability of these messages, there is the possibility of duplicate messages on this queue should the transaction writing the event message data to the MQEV data queue be rolled back and re-processed. See 6.3.3.1 Unique ID on page 36 for more information about handling these possible duplicates.
The PCF value for this is MQG_PUT_IMMEDIATE.

READER

Whether the data is compressed or retains all the spaces and newlines to be more “human readable”.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_READER.

Possible values are:

PROGRAM

The data is designed to be read by a computer program. It is compressed and all extraneous newlines and spaces are removed. The data is all on one line. This is the default value.
The PCF value for this is MQG_READER_PROGRAM.

HUMAN

The data is designed to be human readable and contains newlines and spaces that are not essential but make the layout easier to read for human beings.
The PCF value for this is MQG_READER_HUMAN.

ZEROVALS

How to handle zero values in the messages emitted.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_ZERO_VALUES.

Possible values are:-

HIDE

Do not include any zero values in the messages. This is the default value
The PCF value for this is MQG_ZEROVALS_SHOW.

SHOW

Include zero values in the messages.
The PCF value for this is MQG_ZEROVALS_SHOW.

REPLACE

Whether the existing emitter configuration is to be replaced with this one. This is optional. The default is not to replace the emitter configuration. When specified the emitter replaces the existing one with the same name. If a matching emitter configuration does not exist, one is added.

When using the PCF interface, this is an MQCFIN parameter with identifier MQIACF_REPLACE.

11.9 DEFINE EVSTREAM

Use the MQSC command **`DEFINE EVSTREAM`** (or it's equivalent PCF command **`MQG_CMD_DEFINE_EV_STREAM`**) to create a stream.

Streams are used to store the records (events, accounting and statistics) processed by **MQEV**.

A log file entry will be written by this command showing the stream that was created.

11.9.1 Syntax diagram for DEFINE EVSTREAM

```
►►─ DEFINE EVSTREAM( stream-name ) ──────── TYPE( ┬─ EVENTS ─┬─ ) ──────►
                                                  ├─ STATQ ──┤
                                                  ├─ STATCHL ┤
                                                  ├─ STATMQI ┤
                                                  ├─ ACCTQ ──┤
                                                  └─ ACCTMQI ┘
    ┌─ DEFAULT( NO ) ──┐    ┌─ DESCR( ' ' ) ──┐
►───┤                  ├────┤                 ├──────────────────────────►
    └─ DEFAULT( YES ) ─┘    └─ DESCR( string ) ┘       └─ REPLACE ─┘

    ┌─ EVEMIT( ' ' ) ──────────────┐    ┌─ RETINTVL( default-retintvl ) ─┐
►───┤                              ├────┤                                ├──────►◄
    └─ EVEMIT( emit-object-name ) ─┘    └─ RETINTVL( integer ) ──────────┘
```

11.9.2 Parameter descriptions for DEFINE EVSTREAM

(stream-name)

The name of the stream to be created. The maximum length of this string is MQG_STREAM_NAME_LENGTH (64).

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_STREAM_NAME.

TYPE

The type of data that will be stored on this stream. This attribute cannot be altered.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_STREAM_TYPE.

Possible values are:-

EVENTS

This stream will contain event data. Displaying the data on this stream is done using the DISPLAY EVENTS command.
The PCF value for this is MQG_STREAM_TYPE_EVENTS.

STATQ

This stream will contain queue statistics data. Displaying the data on this stream is done using the DISPLAY STATQ command.
The PCF value for this is MQG_STREAM_TYPE_STAT_Q.

STATCHL

This stream will contain channel statistics data. Displaying the data on this stream is done using the DISPLAY STATCHL command.
The PCF value for this is MQG_STREAM_TYPE_STAT_CHL.

STATMQI

This stream will contain MQI statistics data. Displaying the data on this stream is done using the DISPLAY STATMQI command.
The PCF value for this is MQG_STREAM_TYPE_STAT_MQI.

ACCTQ

This stream will contain queue accounting data. Displaying the data on this stream is done using the DISPLAY ACCTQ command.
The PCF value for this is MQG_STREAM_TYPE_ACCT_Q.

ACCTMQI

This stream will contain MQI accounting data. Displaying the data on this stream is done using the DISPLAY ACCTMQI command.
The PCF value for this is MQG_STREAM_TYPE_ACCT_MQI.

DEFAULT

Whether this stream is to be the default stream. Only one stream of each type has the value DEFAULT(YES). If you create another stream to have DEFAULT(YES) the current default stream for that type is changed to DEFAULT(NO).

If no streams are nominated as default, a stream is automatically created and set to be the default when one is needed. See Chapter 5: Streams on page 33 for the default names.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_DEFAULT_STREAM.

Possible values are:-

YES

This stream is the default stream.
The PCF value for this is MQG_DEFAULT_YES.

NO

This stream is not the default stream.
The PCF value for this is MQG_DEFAULT_NO.

DESCR

A text description of the stream. The maximum length of this string is MQ_APPL_DESC_LENGTH (64). Automatically generated streams have a description of “Auto-defined on date”.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_STREAM_DESC.

EVEMIT

The name of an EVEMIT object which controls how data on this stream is additionally emitted. The default value for this field is blank which means data is not emitted in any format. The maximum length of this string is MQG_EMIT_NAME_LENGTH.

The named emit object must exist or the DEFINE command will fail.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_EMIT_NAME.

RETINTVL

The retention interval for data on this stream (in days). If not specified the default value is taken from the `RETINTVL` on the EV object.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_RETENTION_INTERVAL.

REPLACE

Whether the existing stream configuration is to be replaced with this one. This is optional. The default is not to replace the stream configuration. When specified the stream replaces the existing one with the same name. If a matching stream configuration does not exist, one is added.

When using the PCF interface, this is an MQCFIN parameter with identifier MQIACF_REPLACE.

11.10 DELETE EVEMIT

Use the MQSC command **`DELETE EVEMIT`** (or it's equivalent PCF command **`MQG_CMD_DELETE_EV_EMIT`**) to remove an emit object.

If the emit object to be deleted is currently referenced in an **`EVSTREAM`** object, the deletion will fail with an 'in use' error.

A log file entry will be written by this command showing the emit object that was removed.

11.10.1 Syntax diagram for DELETE EVEMIT

```
►►─ DELETE EVEMIT( emit-object-name ) ──────────────────────────────────►◄
```

11.10.2 Parameter descriptions for DELETE EVEMIT

(emit-object-name)

The name of the emit object to be deleted. The maximum length of this string is MQG_EMIT_NAME_LENGTH (48).

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_EMIT_NAME.

11.11 DELETE EVSTREAM

Use the MQSC command **`DELETE EVSTREAM`** (or it's equivalent PCF command **`MQG_CMD_DELETE_EV_STREAM`**) to remove a stream.

Streams are used to store the records (events, accounting and statistics) processed by **MQEV**. Streams can only be deleted if they are empty i.e. do not contain any records.

A log file entry will be written by this command showing the stream that was removed.

11.11.1 Syntax diagram for DELETE EVSTREAM

```
►►─ DELETE EVSTREAM( stream-name ) ──┬──────────────────────────────────────────┬──►◄
                                     └─── TYPE( ─┬─ EVENTS ──┬─ ) ─┘
                                                 ├─ STATQ ───┤
                                                 ├─ STATCHL ─┤
                                                 ├─ STATMQI ─┤
                                                 ├─ ACCTQ ───┤
                                                 └─ ACCTMQI ─┘
```

11.11.2 Parameter descriptions for DELETE EVSTREAM

(stream-name)

The name of the stream to be deleted. The maximum length of this string is MQG_STREAM_NAME_LENGTH (64). When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_STREAM_NAME.

TYPE

The type of data that is stored on this stream. This attribute is optional, and only required if the stream name is not a unique reference to the stream object being deleted.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_STREAM_TYPE.

Possible values are:-

EVENTS

This stream contains event data. The PCF value for this is MQG_STREAM_TYPE_EVENTS.

STATQ

This stream contains queue statistics data.
The PCF value for this is MQG_STREAM_TYPE_STAT_Q.

STATCHL

This stream contains channel statistics data.
The PCF value for this is MQG_STREAM_TYPE_STAT_CHL.

STATMQI

This stream contains MQI statistics data.
The PCF value for this is MQG_STREAM_TYPE_STAT_MQI.

ACCTQ

This stream contains queue accounting data.
The PCF value for this is MQG_STREAM_TYPE_ACCT_Q.

ACCTMQI

This stream contains MQI accounting data.
The PCF value for this is MQG_STREAM_TYPE_ACCT_MQI.

11.12 DISPLAY ACCTMQI

Use the MQSC command **DISPLAY ACCTMQI** (or it's equivalent PCF command **MQG_CMD_DISPLAY_ACCT_MQI**) to display the MQI accounting records processed and stored by **MQEV**. For a description of the accounting fields provided by MQ please read the MQ documentation

11.12.1 Syntax diagram for DISPLAY ACCTMQI

```
►►─ DISPLAY ACCTMQI( wildcarded-appl-name ) ─┬──────────────────────────────►
                                             └─ EVQMGR( qmgr-name ) ─┘

    ┌─ GAPFILL( NO ) ──────────────┐     ┌─ SUM( APPL ) ──────────────┐
►───┤                              ├─────┤                            ├──────►
    └─ GAPFILL( ─┬─ INNER ────┬─ ) ─┘     ├─ SUM( ─┬─ NONE ───┬─ ) ─┤
                 ├─ ACTIVITY ─┤            │       └─ TOTAL ──┘      │
                 └─ FULL ─────┘            │       ┌─,──────────┐    │
                                           │       ↓            │    │
                                           └─ SUM( ─┬─ APPL ────┬─ ) ─┘
                                                    ├─ CHANNEL ─┤
                                                    ├─ CONNAME ─┤
                                                    ├─ USERID ──┤
                                                    ├─ RPRODUCT ─┤
                                                    ├─ RVERSION ─┤
                                                    ├─ PID ─────┤
                                                    ├─ TID ─────┤
                                                    ├─ STREAM ──┤
                                                    └─ CONNID ──┘
    ┌─ ZEROVALS( HIDE ) ─┐
►───┤                    ├──────────────────────────────────────────────────►
    └─ ZEROVALS( SHOW ) ─┘

    ┌─ FROM( '-24h' ) ─────┐   ┌─ TO( 'now' ) ─────────┐   ┌─ TZ( 0 ) ─────┐
►───┤                      ├───┤                       ├───┤               ├──►
    └─ FROM( date-and-time ) ─┘   └─ TO( date-and-time ) ─┘   └─ TZ( integer ) ─┘

    ┌─ INTVL( ' ' ) ─────┐
►───┤                    ├──────────────────────────────────────────────────►
    └─ INTVL( time-period ) ─┘

►───┬──────────────────────────────────────┬──┬──────────────────────────────►
    └─ INTVLSTA( date-and-time-seconds ) ─┘  └─ INTVLEND( date-and-time-seconds ) ─┘

►───┬──────────────────────────────────────┬──┬──────────────────────────────►
    └─ WHERE( FilterCondition ) ─┘           └─ PREWHERE( FilterCondition ) ─┘

    ┌─ MAXRECS( ev-value ) ─┐  ┌─ MAXRESP( 100 ) ─────┐
►───┤                       ├──┤                      ├──┬──────────────┬──┬─────►◄
    └─ MAXRECS( integer ) ─┘  └─ MAXRESP( integer ) ─┘  └─┤ attributes ├─┘  └─ ALL ─┘
```

11.12.2 Parameter descriptions for DISPLAY ACCTMQI

(wildcarded-appl-name)

The application name for which accounting records are to be displayed. This can be a wildcarded string.

When using the PCF interface, this is an MQCFST parameter with identifier MQCACF_APPL_NAME.

EVQMGR

The queue manager associated with the records to be displayed.

This parameter is used to specify which queue manager you wish to see data from. If it is not specified, the output shows records from the queue manager you are connected to.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_Q_MGR_NAME.

GAPFILL

Whether to add zeroed records where none exist to produce a set of records that will graph appropriately.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_GAP_FILL.

Possible Values are:-

NO

Do not fill in the gaps with zeroed records.
The PCF value for this is MQG_GAPFILL_NO.

INNER

Produce zeroed records for inner gaps in the set of records.
The PCF value for this is MQG_GAPFILL_INNER.

ACTIVITY

Add zeroed records to produce a set of records that spans the time period where there is any activity on this queue, not just the activity shown by the attributes displayed.
The PCF value for this is MQG_GAPFILL_ACTIVITY.

FULL

Add zeroed records to produce a set of records that spans the entire time period requested.
The PCF value for this is MQG_GAPFILL_FULL

SUM

Whether the records are summed together, and if so how they are totalled. Multiple values can be provided in a comma-separated list, except where indicated that a value cannot be combined with any others.

When multiple values are provided, a record will be returned for each unique combination, e.g. `SUM(APPL, CHANNEL)` will return one record for each unique combination of application name and channel name.

When using the PCF interface, this can be an MQCFIN parameter (should you only need to supply one value) or an MQCFIL parameter (should you need to supply multiple values) with identifier MQG_ATTR_SUM.

When any of the `SUM` values apart from `NONE` and `TOTAL` are used without the `INTVL` attribute, this will result in a single record per unique key combination being returned. If used with the `INTVL` attribute, one record per application will be returned for each interval.

For more information about the returned records and their interval start and end times, please see Chapter 10 Returned Interval Times on page 48.

Possible Values are:-

NONE

Do not add together any records. The command returns each individual record as reported by IBM MQ. This value cannot be combined with any others.
The PCF value for this is MQG_SUM_NONE.

APPL

A record will be returned for each unique application name. This is the default value.
The PCF value for this is MQG_SUM_APPLICATION_NAME.

CHANNEL

A record will be returned for each unique channel name.
The PCF value for this is MQG_SUM_CHANNEL_NAME.

CONNAME

A record will be returned for each unique connection name.
The PCF value for this is MQG_SUM_CONNECTION_NAME.

USERID

A record will be returned for each unique user identifier.
The PCF value for this is MQG_SUM_USER_ID.

PID

A record will be returned for each unique process identifier.
The PCF value for this is MQG_SUM_PID.

TID

A record will be returned for each unique thread identifier.
The PCF value for this is MQG_SUM_TID.

RPRODUCT

A record will be returned for each unique remote product.
The PCF value for this is MQG_SUM_RPRODUCT.

RVERSION

A record will be returned for each unique remote version.
The PCF value for this is MQG_SUM_RVERSION.

CONNID

A record will be returned for each unique connection identifier.
The PCF value for this is MQG_SUM_CONNECTION_ID.

STREAM

A record will be returned for each MQEV stream
The PCF value for this is MQG_SUM_STREAM.

TOTAL

Add together all of the records that match the other criteria (e.g. `WHERE` clause) on the display command. If used without the `INTVL` attribute, this will result in only a single record being returned. If used with the `INTVL` attribute, one record will be returned for each interval. The various key fields reported on the returned record will reflect the fact that the numbers for many applications might have been added together. Fields that represent multiple values will be shown with an '*'.
This value cannot be combined with any others.
The PCF value for this is MQG_SUM_TOTAL.

ZEROVALS

How to handle zero values in the records displayed.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_ZERO_VALUES.

Possible Values are:-

HIDE

Do not show any zero values. This is the default value.
The PCF value for this is MQG_ZEROVALS_HIDE.

SHOW

Show zero values in records.
The PCF value for this is MQG_ZEROVALS_SHOW.

INTVL

When summing records, the reported records are totalled in intervals of the specified length. That is, it you request an INTVL(4hour), reported intervals will be at midnight, 4am, 8am, noon and so on. This parameter is only valid when you are not using `SUM(NONE)`. The maximum length of this string is MQG_DATE_TIME_LENGTH (30).

For more information about the returned records and their interval start and end times, please see Chapter 10 Returned Interval Times on page 48.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_INTERVAL.

Many different parameter formats are supported to provide an interval. The following are supported:

Values	Meaning
2day (or 2d)	Two days
4hour (or 4hr or 4h)	Four hours
3minute (or 3min or 3m)	Three minutes
1d4h	Values can be combined without spaces

INTVLSTA

This is an integer representation[13] of the date and time of the start of the interval.

When using the PCF interface, this is an MQCFIN64 parameter with identifier MQG_ATTR_START_OF_INTERVAL. This can be both and input and output parameter.

Either use `INTVLSTA` and `INTVLEND`, or use `FROM` and `TO`. You cannot use both. It is expected that users will mainly use `FROM` and `TO`. `INTVLSTA` and `INTVLEND` are designed for programmable interfaces to input values previously returned on earlier commands.

For more information about the returned records and their interval start and end times, please see Chapter 10 Returned Interval Times on page 48.

INTVLEND

This is an integer representation[13] of the date and time of the end of the interval.

When using the PCF interface, this is an MQCFIN64 parameter with identifier MQG_ATTR_END_OF_INTERVAL. This can be both and input and output parameter.

Please refer to the description of INTVLSTA for advice on when to use this parameter.

13 The number of seconds since 1st January 1970 – also known as Unix time or Epoch time.

TO

The time up to when records should be returned. The maximum length of this string is MQG_DATE_TIME_LENGTH (30).

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_TO.

For more information about the returned records and their interval start and end times, please see Chapter 10 Returned Interval Times on page 48.

Please see the description of the `FROM` parameter for the allowed values.

If not specified then the value 'now' will be used.

FROM

The time from which records should be returned. The maximum length of this string is MQG_DATE_TIME_LENGTH (30).

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_FROM.

For more information about the returned records and their interval start and end times, please see Chapter 10 Returned Interval Times on page 48.

Many different parameter formats are supported, the time can be specified in either absolute or relative terms.

If not specified then the value '-24hour' will be used.

The following are supported:

Absolute values	**Meaning**
now	The current time
8	Eight AM
8.30	Eight thirty AM
8.31.46	Eight thirty-one and 46 seconds
04-12	12th April (this year) The day and month fields must always be two digits
2018-10-18	18th October 2018 The year field must always be four digits
2018-10-18 8.31.46	An explicit date and time
Relative values	**Relative to 'the other' time parameter** **Note that both times can not be relative**
-2day (or -2d)	Two days before
+1d	One day after
-4hour (or -4hr or -4h)	Four hours before
-3minute (or -3min or -3m)	Three minutes before
-10second (-10 sec or -10s or -10)	Ten seconds before
-1d4h3m6s	Values can be combined without spaces

TZ

The bias, in minutes, of the time zone that the `FROM` and `TO` parameters are specified in, and any provided `INTVL` will also be aligned to this time zone.

When using the PCF interface, this is an MQCFIN parameter with identifier MQC_ATTR_TIMEZONE.

If not specified then the time zone will be assumed to be UTC.

Here are some examples:-

Time zone	TZ Value
Auckland, New Zealand	`TZ(-720)`
UTC	`TZ(0)`
Portland, Oregon, USA	`TZ(480)`

If you are using MO71 or MQSCX, you do not need to manually provide this attribute as those tools automatically include it from known configuration in those tools.

MAXRECS

The number of source records which should be used to construct the response. This prevents inadvertent consumption of CPU when issuing queries against large amounts of data.

The default value is taken from the **`EV`** object which itself has a default value of 1,000,000.

This value can not be larger than 100,000,000.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_MAX_RECORDS.

MAXRESP

The number of responses to be returned to this DISPLAY command. The default value is 100.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_MAX_RESPONSES.

PREWHERE

Specify a filter condition to only total records that satisfy the selection criterion of the filter condition. For more about the WHERE clause see Chapter 8 Where Clause() on page 42.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_PREWHERE.

WHERE

Specify a filter condition to only display records that satisfy the selection criterion of the filter condition. For more about the WHERE clause see Chapter 8 Where Clause() on page 42.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_WHERE.

ACCTMQI Attributes

The attribute list can specify any of the following values. When using the PCF interface, this is an MQCFIL parameter with identifier MQG_ATTR_EV_ACCT_MQI_ATTRS.

Those highlighted are constructed attributes for your convenience and are not stored in the accounting message. For this reason they are not available in the `MQEVAcctMQI` function.

For a description of these fields please read the MQ documentation

MQSC Value	PCF Constant	PCF Type	Description
`ALL`	`MQIACF_ALL`	`N/A`	All attributes
`CMDLEVEL`	`MQIA_COMMAND_LEVEL`	`MQCFIN`	IBM MQ Command Level
`CONNID`	`MQBACF_CONNECTION_ID`	`MQCFBS`	Connection Identifier
`USERID`	`MQCACF_USER_IDENTIFIER`	`MQCFST`	User Identifier
`CHANNEL`	`MQCACH_CHANNEL_NAME`	`MQCFST`	Channel Name
`CONNAME`	`MQCACH_CONNECTION_NAME`	`MQCFST`	Connection Name

MQSC Value	PCF Constant	PCF Type	Description
RPRODUCT	MQCACH_REMOTE_PRODUCT	MQCFST	Remote Product
RVERSION	MQCACH_REMOTE_VERSION	MQCFST	Remote Version
CONNDURN	MQG_ACCST_CONNECTION_DURATION	MQCFIN	Duration of connection (in seconds)
CONNTI	MQG_ACCST_CONN_TIME	MQCFIN64	UTC Time when application connected [14]
DISCTI	MQG_ACCST_DISC_TIME	MQCFIN64	UTC Time when application disconnected [14]
PID	MQIACF_PROCESS_ID	MQCFIN	Process ID
TID	MQIACF_THREAD_ID	MQCFIN	Thread ID
SEQNUM	MQIACF_SEQUENCE_NUMBER	MQCFIN	Sequence number of record
DISCTYPE	MQIAMO_DISC_TYPE	MQCFIN	Disconnect Type (Normal | Implicit | QMgr)
OPENQ	MQG_ACCST_OPEN_QUEUE	MQCFIN	Open Queue count
OPENNL	MQG_ACCST_OPEN_NAMELIST	MQCFIN	Open Namelist count
OPENPR	MQG_ACCST_OPEN_PROCESS	MQCFIN	Open Process count
OPENQM	MQG_ACCST_OPEN_Q_MGR	MQCFIN	Open QMgr count
OPENTP	MQG_ACCST_OPEN_TOPIC	MQCFIN	Open Topic count
OPENQFL	MQG_ACCST_OPEN_FAIL_QUEUE	MQCFIN	Open Queue fail count
OPENNLFL	MQG_ACCST_OPEN_FAIL_NAMELIST	MQCFIN	Open Namelist fail count
OPENPRFL	MQG_ACCST_OPEN_FAIL_PROCESS	MQCFIN	Open Process fail count
OPENQMFL	MQG_ACCST_OPEN_FAIL_Q_MGR	MQCFIN	Open QMgr fail count
OPENTPFL	MQG_ACCST_OPEN_FAIL_TOPIC	MQCFIN	Open Topic fail count
CLOSEQ	MQG_ACCST_CLOSE_QUEUE	MQCFIN	Close Queue count
CLOSENL	MQG_ACCST_CLOSE_NAMELIST	MQCFIN	Close Namelist count
CLOSEPR	MQG_ACCST_CLOSE_PROCESS	MQCFIN	Close Process count
CLOSEQM	MQG_ACCST_CLOSE_Q_MGR	MQCFIN	Close QMgr count
CLOSETP	MQG_ACCST_CLOSE_TOPIC	MQCFIN	Close Topic count
CLOSEQFL	MQG_ACCST_CLOSE_FAIL_QUEUE	MQCFIN	Close Queue fail count
CLOSENLFL	MQG_ACCST_CLOSE_FAIL_NAMELIST	MQCFIN	Close Namelist fail count
CLOSEPRFL	MQG_ACCST_CLOSE_FAIL_PROCESS	MQCFIN	Close Process fail count
CLOSEQMFL	MQG_ACCST_CLOSE_FAIL_Q_MGR	MQCFIN	Close QMgr fail count
CLOSETPFL	MQG_ACCST_CLOSE_FAIL_TOPIC	MQCFIN	Close Topic fail count
ALLPUT	MQG_ACCST_ALL_PUTS	MQCFIN	All Puts Constructed from sum of PUT and PUT1
PUT	MQG_ACCST_PUTS	MQCFIN	Puts Constructed from sum of PUTNP and PUTP
PUTNP	MQG_ACCST_PUTS_NP	MQCFIN	Puts (Non-persistent)
PUTP	MQG_ACCST_PUTS_P	MQCFIN	Puts (Persistent)
PUTBYTE	MQG_ACCST_64_PUT_BYTES	MQCFIN64	Put Bytes Constructed from sum of PUTBYTENP and PUTBYTEP
PUTBYTENP	MQG_ACCST_64_PUT_BYTES_NP	MQCFIN64	Put Bytes (Non-persistent)
PUTBYTEP	MQG_ACCST_64_PUT_BYTES_P	MQCFIN64	Put Bytes(Persistent)

14 The number of seconds since 1st January 1970 – also known as Unix time or Epoch time.

MQSC Value	PCF Constant	PCF Type	Description
PUTFAIL	MQG_ACCST_PUTS_FAILED	MQCFIN	Puts Failed
PUT1	MQG_ACCST_PUT1S	MQCFIN	Put1s Constructed from sum of PUT1NP and PUT1P
PUT1NP	MQG_ACCST_PUT1S_NP	MQCFIN	Put1s (Non-persistent)
PUT1P	MQG_ACCST_PUT1S_P	MQCFIN	Put1s (Persistent)
PUT1FAIL	MQG_ACCST_PUT1S_FAILED	MQCFIN	Put1s Failed
GET	MQG_ACCST_GETS	MQCFIN	Gets Constructed from sum of GETNP and GETP
GETNP	MQG_ACCST_GETS_NP	MQCFIN	Gets (Non-persistent)
GETP	MQG_ACCST_GETS_P	MQCFIN	Gets (Persistent)
GETFAIL	MQG_ACCST_GETS_FAILED	MQCFIN	Gets Failed
GETBYTE	MQG_ACCST_64_GET_BYTES	MQCFIN64	Get Bytes Constructed from sum of GETBYTENP and GETBYTEP
GETBYTENP	MQG_ACCST_64_GET_BYTES_NP	MQCFIN64	Get Bytes (Non-persistent)
GETBYTEP	MQG_ACCST_64_GET_BYTES_P	MQCFIN64	Get Bytes(Persistent)
BRS	MQG_ACCST_BROWSES	MQCFIN	Browses Constructed from sum of BRSNP and BRSP
BRSNP	MQG_ACCST_BROWSES_NP	MQCFIN	Browses (Non-persistent)
BRSP	MQG_ACCST_BROWSES_P	MQCFIN	Browses (Persistent)
BRSFAIL	MQG_ACCST_BROWSES_FAILED	MQCFIN	Browses Failed
BRSBYTE	MQG_ACCST_64_BROWSE_BYTES	MQCFIN64	Browses Bytes Constructed from sum of BRSBYTENP and BRSBYTEP
BRSBYTENP	MQG_ACCST_64_BROWSE_BYTES_NP	MQCFIN64	Browses Bytes (Non-persistent)
BRSBYTEP	MQG_ACCST_64_BROWSE_BYTES_P	MQCFIN64	Browses Bytes(Persistent)
COMMIT	MQG_ACCST_COMMIT	MQCFIN	Commit count
COMMITFL	MQG_ACCST_COMMIT_FAIL	MQCFIN	Commit fail count
BACKOUT	MQG_ACCST_BACKOUT	MQCFIN	Backout count
INQQ	MQG_ACCST_INQ_QUEUE	MQCFIN	Inquire Queue count
INQNL	MQG_ACCST_INQ_NAMELIST	MQCFIN	Inquire Namelist count
INQPR	MQG_ACCST_INQ_PROCESS	MQCFIN	Inquire Process count
INQQM	MQG_ACCST_INQ_Q_MGR	MQCFIN	Inquire QMgr count
INQQFL	MQG_ACCST_INQ_FAIL_QUEUE	MQCFIN	Inquire Queue fail count
INQNLFL	MQG_ACCST_INQ_FAIL_NAMELIST	MQCFIN	Inquire Namelist fail count
INQPRFL	MQG_ACCST_INQ_FAIL_PROCESS	MQCFIN	Inquire Process fail count
INQQMFL	MQG_ACCST_INQ_FAIL_Q_MGR	MQCFIN	Inquire QMgr fail count
SETQ	MQG_ACCST_SET_QUEUE	MQCFIN	Set Queue count
SETQFL	MQG_ACCST_SET_FAIL_QUEUE	MQCFIN	Set Queue fail count
SUBDURCR	MQG_ACCST_SUB_DUR_CREATED	MQCFIN	Durable Sub created
SUBDURAL	MQG_ACCST_SUB_DUR_ALTERED	MQCFIN	Durable Sub altered
SUBDURRS	MQG_ACCST_SUB_DUR_RESUMED	MQCFIN	Durable Sub resumed
SUBNDURCR	MQG_ACCST_SUB_NONDUR_CREATED	MQCFIN	Non-durable Sub created

MQSC Value	PCF Constant	PCF Type	Description
SUBNDURAL	MQG_ACCST_SUB_NONDUR_ALTERED	MQCFIN	Non-durable Sub altered
SUBNDURRS	MQG_ACCST_SUB_NONDUR_RESUMED	MQCFIN	Non-durable Sub resumed
SUBFL	MQG_ACCST_SUB_FAIL	MQCFIN	Sub fail count
UNSUBCLS	MQG_ACCST_UNSUB_DUR_CLOSED	MQCFIN	Un-sub close count
UNSUBREM	MQG_ACCST_UNSUB_DUR_REMOVED	MQCFIN	Un-sub removed count
UNSUBNCLS	MQG_ACCST_UNSUB_NONDUR_CLOSED	MQCFIN	Un-sub non-durable close count
UNSUBNREM	MQG_ACCST_UNSUB_NONDUR_REMOVED	MQCFIN	Un-sub non-durable removed count
UNSUBFL	MQG_ACCST_UNSUB_FAIL	MQCFIN	Un-sub fail count
SUBRQ	MQG_ACCST_SUBRQ	MQCFIN	SubRq count
SUBRQFL	MQG_ACCST_SUBRQ_FAIL	MQCFIN	SubRq fail count
CBS	MQG_ACCST_CBS	MQCFIN	MQCB calls Constructed from sum of CBCREATE, CBREMOVE, CBRESUME and CBSUSPEND.
CBCREATE	MQG_ACCST_CBS_CREATED	MQCFIN	MQCB calls using MQOP_REGISTER
CBREMOVE	MQG_ACCST_CBS_REMOVED	MQCFIN	MQCB calls using MQOP_DEREGISTER
CBRESUME	MQG_ACCST_CBS_RESUMED	MQCFIN	MQCB calls using MQOP_RESUME
CBSUSPEND	MQG_ACCST_CBS_SUSPENDED	MQCFIN	MQCB calls using MQOP_SUSPEND
CBSFAIL	MQG_ACCST_CBS_FAILED	MQCFIN	MQCB calls failed
CTLSTA	MQG_ACCST_CTL_STARTED	MQCFIN	MQCTL calls using MQOP_START*
CTLSTP	MQG_ACCST_CTL_STOPPED	MQCFIN	MQCTL calls using MQOP_STOP
CTLRES	MQG_ACCST_CTL_RESUMED	MQCFIN	MQCTL calls using MQOP_RESUME
CTLSUS	MQG_ACCST_CTL_SUSPENDED	MQCFIN	MQCTL calls using MQOP_SUSPEND
CTLFL	MQG_ACCST_CTL_FAIL	MQCFIN	MQCTL calls failed
STAT	MQG_ACCST_STAT	MQCFIN	MQSTAT call count
STATFL	MQG_ACCST_STAT_FAIL	MQCFIN	MQSTAT call fail count
PUTTOP	MQG_ACCST_PUT_TOPIC	MQCFIN	Puts to Topic Constructed from sum of PUTTOPNP and PUTTOPP
PUTTOPNP	MQG_ACCST_PUT_TOPIC_NP	MQCFIN	Puts to Topic (Non-persistent)
PUTTOPP	MQG_ACCST_PUT_TOPIC_P	MQCFIN	Puts to Topic (Persistent)
PUTTOPFL	MQG_ACCST_PUT_TOPIC_FAILED	MQCFIN	Puts to Topic Failed
PUT1TOP	MQG_ACCST_PUT1_TOPIC	MQCFIN	Put1s to Topic Constructed from sum of PUT1TOPNP and PUT1TOPP
PUT1TOPNP	MQG_ACCST_PUT1_TOPIC_NP	MQCFIN	Put1s to Topic (Non-persistent)
PUT1TOPP	MQG_ACCST_PUT1_TOPIC_P	MQCFIN	Put1s to Topic (Persistent)
PUT1TOPFL	MQG_ACCST_PUT1_TOPIC_FAILED	MQCFIN	Put1s to Topic Failed
PUTTOPBYTE	MQG_ACCST_PUT_TOPIC_BYTES	MQCFIN64	Put to Topic Bytes Constructed from sum of PUTTOPBYTENP and PUTTOPBYTEP
PUTTOPBYTENP	MQG_ACCST_PUT_TOPIC_BYTES_NP	MQCFIN64	Put to Topic Bytes (Non-persistent)
PUTTOPBYTEP	MQG_ACCST_PUT_TOPIC_BYTES_P	MQCFIN64	Put to Topic Bytes(Persistent)
CONNS	MQG_ACCST_CONNECTIONS	MQCFIN	Count of connections

MQSC Value	PCF Constant	PCF Type	Description
RECORDS	MQG_ATTR_RECORDS	MQCFIN	Number of records totalled together
INTVLSTA	MQG_ATTR_START_OF_INTERVAL	MQCFIN64	UTC Interval Start time [15]
INTVLEND	MQG_ATTR_END_OF_INTERVAL	MQCFIN64	UTC Interval End time [15]

15 The number of seconds since 1st January 1970 – also known as Unix time or Epoch time.

11.13 DISPLAY ACCTQ

Use the MQSC command **`DISPLAY ACCTQ`** (or it's equivalent PCF command **`MQG_CMD_DISPLAY_ACCT_Q`**) to display the Queue accounting records processed and stored by **MQEV**. For a description of the accounting fields provided by MQ please read the MQ documentation

11.13.1 Syntax diagram for DISPLAY ACCTQ

```
►►─ DISPLAY ACCTQ( wildcarded-queue-name ) ─┬──────────────────────────┬──────────────►
                                            └─ EVQMGR( qmgr-name ) ─┘

    ┌─ GAPFILL( NO ) ──────────────┐     ┌─ SUM( QUEUE ) ──────────────┐
►───┤                              ├─────┤                             ├──────────────►
    └─ GAPFILL( ─┬─ INNER ────────┬─ ) ─┘     ├─ SUM( ─┬─ NONE ─────────┬─ ) ─┤
                 ├─ ACTIVITY ─────┤                   └─ TOTAL ────────┘     │
                 └─ FULL ─────────┘                  ┌─,──────────────┐      │
                                                     ↓                │      │
                                                     └─ SUM( ─┬─ QUEUE ────────┬─ ) ─┘
                                                              ├─ APPL ─────────┤
                                                              ├─ CHANNEL───────┤
                                                              ├─ CONNAME ──────┤
                                                              ├─ USERID ───────┤
                                                              ├─ PID ──────────┤
                                                              ├─ TID ──────────┤
                                                              ├─ STREAM ───────┤
                                                              └─ CONNID ───────┘
    ┌─ ZEROVALS( HIDE ) ─┐
►───┤                    ├──────────────────────────────────────────────────────────►
    └─ ZEROVALS( SHOW ) ─┘

    ┌─ FROM( '-24h' ) ─────────┐   ┌─ TO( 'now' ) ──────────┐   ┌─ TZ( 0 ) ──────────┐
►───┤                          ├───┤                        ├───┤                    ├──►
    └─ FROM( date-and-time ) ─┘   └─ TO( date-and-time ) ─┘   └─ TZ( integer ) ─┘

    ┌─ INTVLALGN( CLOCK ) ─┐   ┌─ INTVL( ' ' ) ──────────┐
►───┤                      ├───┤                         ├──────────────────────────►
    └─ INTVLALGN( MQ ) ────┘   └─ INTVL( time-period ) ─┘

►───┬──────────────────────────────────────────┬──────────────────────────────────►
    └─ INTVLSTA( date-and-time-seconds ) ─┘   └─ INTVLEND( date-and-time-seconds ) ─┘

►───┬──────────────────────────────────────────┬──────────────────────────────────►
    └─ WHERE( FilterCondition ) ─┘   └─ PREWHERE( FilterCondition ) ─┘

    ┌─ MAXRECS( ev-value ) ─┐   ┌─ MAXRESP( 100 ) ──────┐
►───┤                       ├───┤                       ├───┬──────────────┬───┬─────┬──►◄
    └─ MAXRECS( integer ) ──┘   └─ MAXRESP( integer ) ─┘   └┤ attributes ├┘   └─ ALL ─┘
```

11.13.2 Parameter descriptions for DISPLAY ACCTQ

(wildcarded-queue-name)

The queue name for which accounting records are to be displayed. This can be a wildcarded string. When using the PCF interface, this is an MQCFST parameter with identifier MQCA_Q_NAME.

EVQMGR

The queue manager associated with the records to be displayed.

This parameter is used to specify which queue manager you wish to see data from. If it is not specified, the output shows records from the queue manager you are connected to.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_Q_MGR_NAME.

GAPFILL

Whether to add zeroed records where none exist to produce a set of records that will graph appropriately.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_GAP_FILL.

Possible Values are:-

NO

Do not fill in the gaps with zeroed records.
The PCF value for this is MQG_GAPFILL_NO.

INNER

Produce zeroed records for inner gaps in the set of records.
The PCF value for this is MQG_GAPFILL_INNER.

ACTIVITY

Add zeroed records to produce a set of records that spans the time period where there is any activity on this queue, not just the activity shown by the attributes displayed.
The PCF value for this is MQG_GAPFILL_ACTIVITY.

FULL

Add zeroed records to produce a set of records that spans the entire time period requested.
The PCF value for this is MQG_GAPFILL_FULL

SUM

Whether the records are summed together, and if so how they are totalled. Multiple values can be provided in a comma-separated list, except where indicated that a value cannot be combined with any others.

When multiple values are provided, a record will be returned for each unique combination, e.g. `SUM(QUEUE, APPL)` will return one record for each unique combination of queue name and application name.

When using the PCF interface, this can be an MQCFIN parameter (should you only need to supply one value) or an MQCFIL parameter (should you need to supply multiple values) with identifier MQG_ATTR_SUM.

When any of the `SUM` values apart from `NONE` and `TOTAL` are used without the `INTVL` attribute, this will result in a single record per unique key combination being returned. If used with the `INTVL` attribute, one record per application will be returned for each interval.

For more information about the returned records and their interval start and end times, please see Chapter 10 Returned Interval Times on page 48.

Possible Values are:-

NONE

Do not add together any records. The command returns each individual record as reported by IBM MQ. This value cannot be combined with any others.
The PCF value for this is MQG_SUM_NONE.

APPL

A record will be returned for each unique application name.
The PCF value for this is MQG_SUM_APPLICATION_NAME.

CHANNEL

A record will be returned for each unique channel name.
The PCF value for this is MQG_SUM_CHANNEL_NAME.

CONNAME

A record will be returned for each unique connection name.
The PCF value for this is MQG_SUM_CONNECTION_NAME.

QUEUE

A record will be returned for each unique queue name.
The PCF value for this is MQG_SUM_QUEUE.

USERID

A record will be returned for each unique user identifier.
The PCF value for this is MQG_SUM_USER_ID.

PID

A record will be returned for each unique process identifier.
The PCF value for this is MQG_SUM_PID.

TID

A record will be returned for each unique thread identifier.
The PCF value for this is MQG_SUM_TID.

CONNID

A record will be returned for each unique connection identifier.
The PCF value for this is MQG_SUM_CONNECTION_ID.

STREAM

A record will be returned for each MQEV stream
The PCF value for this is MQG_SUM_STREAM.

TOTAL

Add together all of the records that match the other criteria (e.g. `WHERE` clause) on the display command. If used without the `INTVL` attribute, this will result in only a single record being returned. If used with the `INTVL` attribute, one record will be returned for each interval. The various key fields reported on the returned record will reflect the fact that the numbers for many applications might have been added together. Fields that represent multiple values will be shown with an '*'.
This value cannot be combined with any others.
The PCF value for this is MQG_SUM_TOTAL.

ZEROVALS

How to handle zero values in the records displayed.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_ZERO_VALUES.

Possible Values are:-

HIDE

Do not show any zero values. This is the default value.
The PCF value for this is MQG_ZEROVALS_HIDE.

SHOW

Show zero values in records.
The PCF value for this is MQG_ZEROVALS_SHOW.

INTVL

When summing records, the reported records are totalled in intervals of the specified length. That is, it you request an INTVL(4hour), reported intervals will be at midnight, 4am, 8am, noon and so on. This parameter is only valid when you are not using `SUM(NONE)`. The maximum length of this string is MQG_DATE_TIME_LENGTH (30).

For more information about the returned records and their interval start and end times, please see Chapter 10 Returned Interval Times on page 48.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_INTERVAL.

Many different parameter formats are supported to provide an interval. The following are supported:

Values	Meaning
2day (or 2d)	Two days
4hour (or 4hr or 4h)	Four hours
3minute (or 3min or 3m)	Three minutes
1d4h	Values can be combined without spaces

INTVLALGN

How to align the reported intervals of records. This parameter is only used when an `INTVL` is set.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_INTERVAL_ALIGN.

CLOCK

Reported intervals are aligned to the clock. That is, if you request an `INTVL(4hour)`, reported intervals will be at midnight, 4am, 8am, noon and so on. This is the default value.
The PCF value for this is MQG_ALIGN_CLOCK.

MQ

Reported intervals are aligned to the intervals reported by IBM MQ.
The PCF value for this is MQG_ALIGN_MQ.

TO

The time up to when records should be returned. The maximum length of this string is MQG_DATE_TIME_LENGTH (30).

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_TO.

For more information about the returned records and their interval start and end times, please see Chapter 10 Returned Interval Times on page 48.

Please see the description of the `FROM` parameter for the allowed values.

If not specified then the value 'now' will be used.

FROM

The time from which records should be returned. The maximum length of this string is MQG_DATE_TIME_LENGTH (30).

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_FROM.

For more information about the returned records and their interval start and end times, please see Chapter 10 Returned Interval Times on page 48.

Many different parameter formats are supported, the time can be specified in either absolute or relative terms.

If not specified then the value '-24hour' will be used.

The following are supported:

Absolute values	Meaning
now	The current time
8	Eight AM
8.30	Eight thirty AM
8.31.46	Eight thirty-one and 46 seconds
04-12	12th April (this year) The day and month fields must always be two digits
2018-10-18	18th October 2018 The year field must always be four digits
2018-10-18 8.31.46	An explicit date and time
Relative values	**Relative to 'the other' time parameter** **Note that both times can not be relative**
-2day (or -2d)	Two days before
+1d	One day after
-4hour (or -4hr or -4h)	Four hours before
-3minute (or -3min or -3m)	Three minutes before
-10second (-10 sec or -10s or -10)	Ten seconds before
-1d4h3m6s	Values can be combined without spaces

TZ

The bias, in minutes, of the time zone that the `FROM` and `TO` parameters are specified in, and any provided `INTVL` will also be aligned to this time zone.

When using the PCF interface, this is an MQCFIN parameter with identifier MQC_ATTR_TIMEZONE.

If not specified then the time zone will be assumed to be UTC.

Here are some examples:-

Time zone	TZ Value
Auckland, New Zealand	`TZ(-720)`
UTC	`TZ(0)`
Portland, Oregon, USA	`TZ(480)`

If you are using MO71 or MQSCX, you do not need to manually provide this attribute as those tools automatically include it from known configuration in those tools.

INTVLSTA

This is an integer representation[16] of the date and time of the start of the interval.

When using the PCF interface, this is an MQCFIN64 parameter with identifier MQG_ATTR_START_OF_INTERVAL. This can be both and input and output parameter.

Either use `INTVLSTA` and `INTVLEND`, or use `FROM` and `TO`. You cannot use both. It is expected that users will mainly use `FROM` and `TO`. `INTVLSTA` and `INTVLEND` are designed for programmable interfaces to input values previously returned on earlier commands..

For more information about the returned records and their interval start and end times, please see Chapter 10 Returned Interval Times on page 48.

16 The number of seconds since 1st January 1970 – also known as Unix time or Epoch time.

INTVLEND

This is an integer representation[17] of the date and time of the end of the interval.

When using the PCF interface, this is an MQCFIN64 parameter with identifier MQG_ATTR_END_OF_INTERVAL. This can be both and input and output parameter.

Please refer to the description of INTVLSTA for advice on when to use this parameter.

MAXRECS

The number of source records which should be used to construct the response. This prevents inadvertent consumption of CPU when issuing queries against large amounts of data.

The default value is taken from the **EV** object which itself has a default value of 1,000,000.

This value can not be larger than 100,000,000.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_MAX_RECORDS.

MAXRESP

The number of responses to be returned to this DISPLAY command. The default value is 100.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_MAX_RESPONSES.

PREWHERE

Specify a filter condition to only total records that satisfy the selection criterion of the filter condition. For more about the WHERE clause see Chapter 8 Where Clause() on page 42.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_PREWHERE.

WHERE

Specify a filter condition to only display records that satisfy the selection criterion of the filter condition. For more about the WHERE clause see Chapter 8 Where Clause() on page 42.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_WHERE.

ACCTQ Attributes

The attribute list can specify any of the following values. When using the PCF interface, this is an MQCFIL parameter with identifier MQG_ATTR_EV_ACCT_Q_ATTRS.

Those highlighted are constructed attributes for your convenience and are not stored in the accounting message. For this reason they are not available in the `MQEVAcctQ` function.

The `CHANNEL` and `CONNAME` fields in an **`ACCTQ`** record will contain the value "`<Not Present>`" for records generated by a queue manager older than V9.2.0 and V9.3.0 respectively. This is done to differentiate between a record about a local connection which would have blanks in those fields, and a record from a queue manager too old to provide the details, which might or might not be a client connection.

For a description of these fields please read the MQ documentation

MQSC Value	PCF Constant	PCF Type	Description
ALL	MQIACF_ALL	N/A	All attributes
ACCTQ	MQCA_Q_NAME	MQCFST	Queue Name
APPLNAME	MQCACF_APPL_NAME	MQCFST	Application Name
CHANNEL	MQCACH_CHANNEL_NAME	MQCFST	Channel Name

17 The number of seconds since 1st January 1970 – also known as Unix time or Epoch time.

MQSC Value	PCF Constant	PCF Type	Description
CONNAME	MQCACH_CONNECTION_NAME	MQCFST	Connection Name
CMDLEVEL	MQIA_COMMAND_LEVEL	MQCFIN	IBM MQ Command Level
CONNID	MQBACF_CONNECTION_ID	MQCFBS	Connection Identifier
USERID	MQCACF_USER_IDENTIFIER	MQCFST	User Identifier
PID	MQIACF_PROCESS_ID	MQCFIN	Process ID
TID	MQIACF_THREAD_ID	MQCFIN	Thread ID
SEQNUM	MQIACF_SEQUENCE_NUMBER	MQCFIN	Sequence number of record
QTYPE	MQIA_Q_TYPE	MQCFIN	Queue Type
DEFTYPE	MQIA_DEFINITION_TYPE	MQCFIN	Queue Definition Type
OPENCNT	MQG_ACCST_OPEN_COUNT	MQCFIN	Count of queues opened in this interval
OPENTI	MQG_ACCST_OPEN_TIME	MQCFIN64	UTC time the queue was first opened
CLOSECNT	MQG_ACCST_CLOSE_COUNT	MQCFIN	Count of queues closed in this interval
CLOSETI	MQG_ACCST_CLOSE_TIME	MQCFIN64	UTC time of the final close of this queue in this interval
ALLPUT	MQG_ACCST_ALL_PUTS	MQCFIN	All Puts Constructed from sum of PUT and PUT1
PUT	MQG_ACCST_PUTS	MQCFIN	Puts Constructed from sum of PUTNP and PUTP
PUTNP	MQG_ACCST_PUTS_NP	MQCFIN	Puts (Non-persistent)
PUTP	MQG_ACCST_PUTS_P	MQCFIN	Puts (Persistent)
PUTFAIL	MQG_ACCST_PUTS_FAILED	MQCFIN	Puts Failed
PUT1	MQG_ACCST_PUT1S	MQCFIN	Put1s Constructed from sum of PUT1NP and PUT1P
PUT1NP	MQG_ACCST_PUT1S_NP	MQCFIN	Put1s (Non-persistent)
PUT1P	MQG_ACCST_PUT1S_P	MQCFIN	Put1s (Persistent)
PUT1FAIL	MQG_ACCST_PUT1S_FAILED	MQCFIN	Put1s Failed
PUTBYTE	MQG_ACCST_64_PUT_BYTES	MQCFIN64	Put Bytes Constructed from sum of PUTBYTENP and PUTBYTEP
PUTBYTENP	MQG_ACCST_64_PUT_BYTES_NP	MQCFIN64	Put Bytes (Non-persistent)
PUTBYTEP	MQG_ACCST_64_PUT_BYTES_P	MQCFIN64	Put Bytes(Persistent)
PUTMINBYTE	MQG_ACCST_PUT_MIN_BYTES	MQCFIN	Minimum size of message put Constructed from minimum of PUTMINBYTENP and PUTMINBYTEP
PUTMINBYTENP	MQG_ACCST_PUT_MIN_BYTES_NP	MQCFIN	Minimum size of non persistent message put
PUTMINBYTEP	MQG_ACCST_PUT_MIN_BYTES_P	MQCFIN	Minimum size of persistent message put
PUTMAXBYTE	MQG_ACCST_PUT_MAX_BYTES	MQCFIN	Maximum size of message put Constructed from maximum of PUTMAXBYTENP and PUTMAXBYTEP

MQSC Value	PCF Constant	PCF Type	Description
PUTMAXBYTENP	MQG_ACCST_PUT_MAX_BYTES_NP	MQCFIN	Maximum size of non-persistent message put
PUTMAXBYTEP	MQG_ACCST_PUT_MAX_BYTES_P	MQCFIN	Maximum size of persistent message put
GENMSGCNT	MQG_ACCST_GENERATED_MSG_COUNT	MQCFIN	Number of generated message in the interval
GET	MQG_ACCST_GETS	MQCFIN	Gets Constructed from sum of GETNP and GETP
GETNP	MQG_ACCST_GETS_NP	MQCFIN	Gets (Non-persistent)
GETP	MQG_ACCST_GETS_P	MQCFIN	Gets (Persistent)
GETFAIL	MQG_ACCST_GETS_FAILED	MQCFIN	Gets Failed
GETBYTE	MQG_ACCST_64_GET_BYTES	MQCFIN64	Get Bytes Constructed from sum of GETBYTENP and GETBYTEP
GETBYTENP	MQG_ACCST_64_GET_BYTES_NP	MQCFIN64	Get Bytes (Non-persistent)
GETBYTEP	MQG_ACCST_64_GET_BYTES_P	MQCFIN64	Get Bytes(Persistent)
GETMINBYTE	MQG_ACCST_GET_MIN_BYTES	MQCFIN	Minimum size of message got Constructed from minimum of GETMINBYTENP and GETMINBYTEP
GETMINBYTENP	MQG_ACCST_GET_MIN_BYTES_NP	MQCFIN	Minimum size of non persistent message got
GETMINBYTEP	MQG_ACCST_GET_MIN_BYTES_P	MQCFIN	Minimum size of persistent message got
GETMAXBYTE	MQG_ACCST_GET_MAX_BYTES	MQCFIN	Maximum size of message got Constructed from maximum of GETMAXBYTENP and GETMAXBYTEP
GETMAXBYTENP	MQG_ACCST_GET_MAX_BYTES_NP	MQCFIN	Maximum size of non-persistent message got
GETMAXBYTEP	MQG_ACCST_GET_MAX_BYTES_P	MQCFIN	Maximum size of persistent message got
BRS	MQG_ACCST_BROWSES	MQCFIN	Browses Constructed from sum of BRSNP and BRSP
BRSNP	MQG_ACCST_BROWSES_NP	MQCFIN	Browses (Non-persistent)
BRSP	MQG_ACCST_BROWSES_P	MQCFIN	Browses (Persistent)
BRSFAIL	MQG_ACCST_BROWSES_FAILED	MQCFIN	Browses Failed
BRSBYTE	MQG_ACCST_64_BROWSE_BYTES	MQCFIN64	Browses Bytes Constructed from sum of BRSBYTENP and BRSBYTEP
BRSBYTENP	MQG_ACCST_64_BROWSE_BYTES_NP	MQCFIN64	Browses Bytes (Non-persistent)
BRSBYTEP	MQG_ACCST_64_BROWSE_BYTES_P	MQCFIN64	Browses Bytes(Persistent)
BRSMINBYTE	MQG_ACCST_BRS_MIN_BYTES	MQCFIN	Minimum size of message browsed Constructed from minimum of BRSMINBYTENP and BRSMINBYTEP
BRSMINBYTENP	MQG_ACCST_BRS_MIN_BYTES_NP	MQCFIN	Minimum size of non persistent message browsed
BRSMINBYTEP	MQG_ACCST_BRS_MIN_BYTES_P	MQCFIN	Minimum size of persistent message browsed
BRSMAXBYTE	MQG_ACCST_BRS_MAX_BYTES	MQCFIN	Maximum size of message browsed Constructed from maximum of BRSMAXBYTENP and

MQSC Value	PCF Constant	PCF Type	Description
			BRSMAXBYTEP
BRSMAXBYTENP	MQG_ACCST_BRS_MAX_BYTES_NP	MQCFIN	Maximum size of non-persistent message browsed
BRSMAXBYTEP	MQG_ACCST_BRS_MAX_BYTES_P	MQCFIN	Maximum size of persistent message browsed
CBS	MQG_ACCST_CBS	MQCFIN	MQCB calls Constructed from the sum of CBCREATE, CBREMOVE, CBSUSPEND and CBRESUME
CBCREATE	MQG_ACCST_CBS_CREATED	MQCFIN	MQCB calls using MQOP_REGISTER
CBREMOVE	MQG_ACCST_CBS_REMOVED	MQCFIN	MQCB calls using MQOP_DEREGISTER
CBRESUME	MQG_ACCST_CBS_RESUMED	MQCFIN	MQCB calls using MQOP_RESUME
CBSUSPEND	MQG_ACCST_CBS_SUSPENDED	MQCFIN	MQCB calls using MQOP_SUSPEND
CBFAIL	MQG_ACCST_CBS_FAILED	MQCFIN	MQCB calls failed
ONQMINNPTI	MQG_ACCST_64_TIME_ON_Q_MIN_NP	MQCFIN64	Shortest time (in micro-seconds) a non-persistent message remained on the queue
ONQMINPTI	MQG_ACCST_64_TIME_ON_Q_MIN_P	MQCFIN64	Shortest time (in micro-seconds) a persistent message remained on the queue
ONQMAXNPTI	MQG_ACCST_64_TIME_ON_Q_MAX_NP	MQCFIN64	Longest time (in micro-seconds) a non-persistent message remained on the queue
ONQMAXPTI	MQG_ACCST_64_TIME_ON_Q_MAX_P	MQCFIN64	Longest time (in micro-seconds) a persistent message remained on the queue
ONQAVGNPTI	MQG_ACCST_64_TIME_ON_Q_AVG_NP	MQCFIN64	Average time (in micro-seconds) a non-persistent message remained on the queue
ONQAVGPTI	MQG_ACCST_64_TIME_ON_Q_AVG_P	MQCFIN64	Average time (in micro-seconds) a persistent message remained on the queue
RECORDS	MQG_ATTR_RECORDS	MQCFIN	Number of records totalled together

11.14 DISPLAY EV

Use the MQSC command **DISPLAY EV** (or it's equivalent PCF command **MQG_CMD_DISPLAY_EV**) to display the main overall configuration and status of the **MQEV** event processor.

11.14.1 Syntax diagram for DISPLAY EV

```
►►─ DISPLAY EV ─┬──────────────────┬─┬──────────────────┬─┬─────────┬──────►◄
                ├─ ACCTMQICT ──────┤ │ ├─ LICREMIND ──────┤ │└─ ALL ─┘
                ├─ ACCTQCT ────────┤ │ ├─ LOGPATH ────────┤
                ├─ ALERTCT ────────┤ │ ├─ LOGRET ─────────┤
                ├─ ALERTRET ───────┤ │ ├─ MAXLOGAGE ──────┤
                ├─ BUILD ──────────┤ │ ├─ MAXLOGSZ ───────┤
                ├─ CMDLEVEL ───────┤ │ ├─ MSGS ───────────┤
                ├─ CMDS ───────────┤ │ ├─ NAME ───────────┤
                ├─ CONNTI ─────────┤ │ ├─ RESETTI ────────┤
                ├─ DEFMAXRECS──────┤ │ ├─ STARTTI ────────┤
                ├─ DEADQ ──────────┤ │ ├─ STATCHLCT ──────┤
                ├─ DESCR ──────────┤ │ ├─ STATMQICT ──────┤
                ├─ DISPCMDS ───────┤ │ ├─ STATQCT ────────┤
                ├─ EVENTCT ────────┤ │ ├─ STORMINT ───────┤
                ├─ EVENTS ─────────┤ │ ├─ STORMTHR ───────┤
                ├─ LICDAYS ────────┤ │ ├─ USEDLQ ─────────┤
                └─ LICENDDA ───────┘ │ └─ VERCHECK ───────┘
```

11.14.2 Parameter descriptions for DISPLAY EV

EV Attributes

The attribute list can specify any of the following values. When using the PCF interface, this is an MQCFIL parameter with identifier MQG_ATTR_EV_ATTRS.

MQSC Value	PCF Constant	PCF Type	Description
ALL	MQIACF_ALL	N/A	All attributes
ACCTMQICT	MQG_ATTR_ACCT_MQI_COUNT	MQCFIN	The total number of MQI Accounting records processed and stored by MQEV since last RESET.
ACCTQCT	MQG_ATTR_ACCT_Q_COUNT	MQCFIN	The total number of Queue Accounting records processed and stored by MQEV since last RESET.
ALERTCT	MQG_ATTR_ALERT_COUNT	MQCFIN	The number of alerts in the system in total. This is the total of all DISPLAY EVQMGR ALERTS
ALERTRET	MQG_ATTR_DEF_ALERT_ RETENTION_INTERVAL	MQCFIN	The retention interval, in days, for alerts.
BUILD	MQG_ATTR_BUILD	MQCFST	The build date of this version of MQEV.
CMDLEVEL	MQG_ATTR_COMMAND_LEVEL	MQCFIN	The command level of the MQEV command server. For Version 9.3.0 this has the value 930.
CMDS	MQG_ATTR_NUM_COMMANDS	MQCFIN	The number of commands processed by the MQEV command server since this instance of MQEV started up
CONNTI	MQG_ATTR_EV_CONNECT_TIME	MQCFIN64	The date and time, in local time, this instance of MQEV last connected to the queue manager. [18]

18 The number of seconds since 1st January 1970 – also known as Unix time or Epoch time.

MQSC Value	PCF Constant	PCF Type	Description
DEADQ	MQG_ATTR_DEAD_LETTER_QUEUE	MQCFST	The name of the dead-letter queue to use. If this is blank, the queue manager defined DEADQ is used.
DEFMAXRECS	MQG_ATTR_DEFAULT_MAX_RECORDS	MQCFIN	The default number of source records which will be used to satisfy a DISPLAY command of the Accounting or Statistics data.
DESCR	MQG_ATTR_EV_DESC	MQCFST	A text description of this **MQEV** instance.
DISPCMDS	MQG_ATTR_DISPLAY_COMMANDS	MQCFIN	How to handle command events received that record DISPLAY commands.
EVENTCT	MQG_ATTR_EVENT_COUNT	MQCFIN	The total number of events processed and stored by **MQEV** since last RESET. This count will not include discarded command events.
EVENTS	MQG_ATTR_NUM_EVENTS	MQCFIN	The number of event messages processed since this instance of **MQEV** started up. This number will include Command events that describe DISPLAY commands even if they have been discarded due to the DISPCMDS setting.
LICDAYS	MQG_ATTR_LICENCE_TIME_LEFT	MQCFIN	The time left, in days, on your **MQEV** licence.
LICENDDA	MQG_ATTR_LICENCE_END_DATE	MQCFIN64	The date when your **MQEV** licence expires. [19]
LICREMIND	MQG_ATTR_LICENCE_REMIND_TIME	MQCFIN	The time left on your **MQEV** licence, in days, after which you will begin to get reminders.
LOGPATH	MQG_ATTR_LOG_PATH	MQCFST	The location of the **MQEV** log files. For more information on how to set the location of MQEV log file see Chapter 7 Logging on page 41.
LOGRET	MQG_ATTR_LOG_RETENTION_ INTERVAL	MQCFIN	The retention interval, in days, for **MQEV** log files.
MAXLOGAGE	MQG_ATTR_MAX_LOG_AGE	MQCFIN	The maximum age, in minutes, of an **MQEV** log file.
MAXLOGSZ	MQG_ATTR_MAX_LOG_SIZE	MQCFIN	The maximum size, in kilobytes, of an **MQEV** log file.
MSGS	MQG_ATTR_NUM_MESSAGES	MQCFIN	The number of all messages (both commands and events) processed since this instance of **MQEV** started up.
NAME	MQG_ATTR_EV_NAME	MQCFST	The name of this **MQEV** instance.
RESETTI	MQG_ATTR_COUNT_RESET_TIME	MQCFIN64	The date and time, in local time, when the event counts were last reset. If they have never been reset, this will show the time when **MQEV** was first started. [19]
STARTTI	MQG_ATTR_EV_START_TIME	MQCFIN64	The date and time, in local time, this instance of **MQEV** started up. [19]
STATCHLCT	MQG_ATTR_STAT_CHL_COUNT	MQCFIN	The total number of Channel Statistics records processed and stored by **MQEV** since last RESET.
STATMQICT	MQG_ATTR_STAT_MQI_COUNT	MQCFIN	The total number of MQI Statistics records processed and stored by **MQEV** since last RESET.
STATQCT	MQG_ATTR_STAT_Q_COUNT	MQCFIN	The total number of Queue Statistics records processed and stored by **MQEV** since last RESET.
STORMINT	MQG_ATTR_STORM_INTERVAL	MQCFIN	The time period, in seconds, within which a number of identical events are received (configured by STORMTHR) before it is considered to be an event storm.

19 The number of seconds since 1st January 1970 – also known as Unix time or Epoch time.

MQSC Value	PCF Constant	PCF Type	Description
STORMTHR	MQG_ATTR_STORM_THRESHOLD	MQCFIN	The number of identical events received in a time period (configured by STORMINT) before it is considered to be an event storm.
VERCHECK	MQG_ATTR_VERSION_CHECK	MQCFIN	Whether to check for newer versions of the product.
USEDLQ	MQG_ATTR_USE_DEAD_LETTER_Q	MQCFIN	Whether the Dead-letter queue is used.

11.15 DISPLAY EVALERT

Use the MQSC command **DISPLAY EVALERT** (or it's equivalent PCF command **MQG_CMD_DISPLAY_EV_ALERT**) to display the attributes of an alert. Alerts can be used as reminders or notifications. Learn more about alerts in Chapter 12 Alerts on page 153.

11.15.1 Syntax diagram for DISPLAY EVALERT

```
►►─ DISPLAY EVALERT─┬──────────────────────┬─┬──────────────────────────────────┬──►
                    └─ ( alert-id ) ───────┘ └─ CATEGORY( wildcarded-string ) ──┘

►─┬──────────────────────────────────┬─┬──────────────────────────────────────┬──►
  └─ EVQMGR( wildcarded-string ) ────┘ └─ EVOBJNAME( wildcarded-string )────────┘

  ┌─MINSEV( WARN ) ──────────────┐
►─┼──────────────────────────────┼─┬──────────────────────────────────────────┬──►
  └─ MINSEV( ─┬─ TERM ──────┬─ ) ─┘ └─ EVOBJTYPE( ─┬─ AUTHINFO ──┬─ ) ─┘
              ├─ SEVERE ────┤                       ├─ AUTHREC ───┤
              ├─ ERROR ─────┤                       ├─ CFSTRUCT ──┤
              └─ INFO ──────┘                       ├─ CHANNEL ───┤
                                                    ├─ CHLAUTH ───┤
                                                    ├─ CLNTCONN ──┤
                                                    ├─ COMMINFO ──┤
                                                    ├─ EVQ ───────┤
                                                    ├─ LISTENER ──┤
                                                    ├─ NAMELIST ──┤
                                                    ├─ POLICY ────┤
                                                    ├─ PROCESS ───┤
                                                    ├─ QUEUE ─────┤
                                                    ├─ QMGR ──────┤
                                                    ├─ RQMNAME ───┤
                                                    ├─ SERVICE ───┤
                                                    ├─ SUB ───────┤
                                                    ├─ STGCLASS ──┤
                                                    ├─ TOPIC ─────┤
                                                    └─ TOPICSTR ──┘
►─┬──────────────────────────────────┬─┬──────────────────────────────────────┬─┬──────────┬─►◄
  └─ WHERE( FilterCondition ) ───────┘ │ ┌─ ALERTTI ────┐ │ ┌─ EXPIRETI ───┐ │ └─ ALL ────┘
                                       │ ├─ CATEGORY ───┤ │ ├─ RETINTVL ───┤ │
                                       │ ├─ EVENTID ────┤ │ ├─ SEVERITY ───┤ │
                                       │ ├─ EVOBJNAME ──┤ │ ├─ EVSTREAM ───┤ │
                                       │ ├─ EVOBJTYPE ──┤ │ └─ TEXT ───────┘ │
                                       │ └─ EVQMGR ─────┘ │                  │
```

11.15.2 Parameter descriptions for DISPLAY EVALERT

(alert-id)

The unique ID of the alert. This parameter is optional.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_ALERT_ID.

CATEGORY

The category of the alerts to be displayed.

This parameter can be used to limit the number of alerts that are displayed. If it is not specified, the display is not limited in this way.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_ALERT_CATEGORY.

EVOBJTYPE

The object type associated with the alerts to be displayed.

This parameter can be used to limit the number of alerts that are displayed. If it is not specified, the display is not limited in this way.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_OBJECT_TYPE.

Possible Values are:-

MQSC value	Meaning	PCF constant
AUTHINFO	Authentication information object	MQOT_AUTH_INFO
AUTHREC	Authorization records	MQOT_AUTH_REC
CFSTRUCT	CF Structure	MQOT_CF_STRUC
CHANNEL	Channel	MQOT_CHANNEL
CHLAUTH	Channel Authentication records	MQOT_CHLAUTH
CLNTCONN	Client connection channel	MQOT_CLNTCONN_CHANNEL
COMMINFO	Communication information object	MQOT_COMM_INFO
EVQ	MQEV Event Queue	MQG_OT_EVENT_QUEUE
LISTENER	Listener	MQOT_LISTENER
NAMELIST	Namelist	MQOT_NAMELIST
NONE	None	MQOT_NONE
POLICY	Protection Policy	MQOT_PROT_POLICY
PROCESS	Process	MQOT_PROCESS
QUEUE	Queue	MQOT_Q
QMGR	Queue manager	MQOT_Q_MGR
RQMNAME	Remote queue manager	MQOT_REMOTE_Q_MGR_NAME
SERVICE	Service object	MQOT_SERVICE
STGCLASS	Storage Class	MQOT_STORAGE_CLASS
SUB	Subscription	MQG_OT_SUB
TOPIC	Topic	MQOT_TOPIC
TOPICSTR	Topic String	MQG_OT_TOPICSTR

EVOBJNAME

The object associated with the alerts to be displayed.

This parameter can be used to limit the number of alerts that are displayed. If it is not specified, the display is not limited in this way.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_OBJECT.

EVQMGR

The queue manager associated with the alerts to be displayed.

This parameter can be used to limit the number of alerts that are displayed. If it is not specified, the display is not limited in this way.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_ALERT_Q_MGR.

MINSEV

The minimum severity of the alerts to be displayed.

This parameter can be used to limit the number of alerts that are displayed. If it is not specified, then a value of WARN (MQG_SEVERITY_WARN) is used. Or, put another way, alerts of severity INFO are not displayed by default.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_MINIMUM_SEVERITY.

Possible values are:-

MQSC value	Meaning	PCF constant
TERM	Termination	MQG_SEVERITY_TERM
SEVERE	Severe Error	MQG_SEVERITY_SEVERE
ERROR	Error.	MQG_SEVERITY_ERROR
WARN	Warning. This is the default value on DISPLAY	MQG_SEVERITY_WARN
INFO	Information	MQG_SEVERITY_INFO

WHERE

Specify a filter condition to only display alerts that satisfy the selection criterion of the filter condition. For more about the WHERE clause see Chapter 8 Where Clause() on page 42.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_WHERE.

EVALERT Attributes

The attribute list can specify any of the following values. When using the PCF interface, this is an MQCFIL parameter with identifier MQG_ATTR_ALERT_ATTRS.

MQSC Value	PCF Constant	PCF Type	Description
ALL	MQIACF_ALL	N/A	All attributes
ALERTTI	MQG_ATTR_ALERT_TIME	MQCFIN64	The date and time, in local time, the alert was raised. [20]
CATEGORY	MQG_ATTR_ALERT_CATEGORY	MQCFST	The category of the alert.
EVENTID	MQG_ATTR_EVENT_ID	MQCFIN	The unique ID of the event this alert is about.
EVOBJNAME	MQG_ATTR_OBJECT	MQCFST	The object name to which this alert is associated.
EVOBJTYPE	MQG_ATTR_OBJECT_TYPE	MQCFIN	The type of object that the object name references. If the alert was added with EVOBJTYPE(NONE) or by omitting EVOBJTYPE, this parameter will be empty on output.
EVSTREAM	MQG_ATTR_STREAM_NAME	MQCFST	The stream name to which this alert is associated.
EVQMGR	MQG_ATTR_ALERT_Q_MGR	MQCFST	The queue manager to which this alert is associated.
EXPIRETI	MQG_ATTR_EXPIRY_TIME	MQCFIN64	The date and time, in local time, the alert will expire. [20]
RETINTVL	MQG_ATTR_RETENTION_INTERVAL	MQCFIN	The retention interval, in seconds, of this alert.
SEVERITY	MQG_ATTR_ALERT_SEVERITY	MQCFIN	The severity of the alert.
TEXT	MQG_ATTR_ALERT_TEXT	MQCFST	The text of the alert.

20 The number of seconds since 1st January 1970 – also known as Unix time or Epoch time.

11.16 DISPLAY EVEMIT

Use the MQSC command **DISPLAY EVEMIT** (or it's equivalent PCF command **MQG_CMD_DISPLAY_EV_EMIT**) to display the emitter object configuration.

11.16.1 Syntax diagram for DISPLAY EVEMIT

```
►►─ DISPLAY EVEMIT( emit-object-name ) ─┬──────────────────────────────────┬──────────►
                                        └─ WHERE( FilterCondition ) ─┘
►──┬──────────────┬──┬──────────────┬──┬──────────┬──────────────────────────────────►◄
   ├─ DESCR ──────┤  ├─ MSGPERS ────┤  └─ ALL ────┘
   ├─ ERRORCT ────┤  ├─ ONERROR ────┤
   ├─ FILE ───────┤  ├─ PUT ────────┤
   ├─ FORMAT ─────┤  ├─ QUEUE ──────┤
   ├─ GROUPING────┤  ├─ READER ─────┤
   ├─ MAXSIZE ────┤  └─ ZEROVALS ───┘
   └─ MSGCT ──────┘
```

11.16.2 Parameter descriptions for DISPLAY EVEMIT

(emit-object-name)

The emitter object name to be displayed. This can be a wildcarded string.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_EMIT_NAME.

WHERE

Specify a filter condition to only display emitter objects that satisfy the selection criterion of the filter condition. For more about the WHERE clause see Chapter 8 Where Clause() on page 42.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_WHERE.

EVEMIT Attributes

The attribute list can specify any of the following values. When using the PCF interface, this is an MQCFIL parameter with identifier MQG_ATTR_EV_EMIT_ATTRS.

MQSC Value	PCF Constant	PCF Type	Description
ALL	MQIACF_ALL	N/A	All attributes
DESCR	MQG_ATTR_EMIT_DESC	MQCFST	A text description of the emit object.
ERRORCT	MQG_ATTR_ERROR_COUNT	MQCFIN	The number of messages that were unable to be written to the EVEMIT queue or file.
FILE	MQG_ATTR_EMIT_FILE_NAME	MQCFST	The file name to emit data to.
FORMAT	MQG_ATTR_EMIT_FORMAT	MQCFIN	The format that the data will be emitted in.
GROUPING	MQG_ATTR_EMIT_GROUPING	MQCFIN	Whether event data should remain grouped as MQ issued it.
MAXSIZE	MQG_ATTR_MAX_SIZE	MQCFIN	The maximum size of emitted messages
MSGCT	MQG_ATTR_MSG_COUNT	MQCFIN	The number of messages that have been written to the queue or file since MQEV started. This number is not hardened, and starts counting from zero each time MQEV starts up.
MSGPERS	MQG_ATTR_MSG_PERSISTENCE	MQCFIN	The persistence of emitted messages
ONERROR	MQG_ATTR_ON_ERROR	MQCFIN	How to behave when an emitted message cannot be put.

MQSC Value	PCF Constant	PCF Type	Description
PUT	`MQG_ATTR_PUT_TRANSACT`	`MQCFIN`	The transactionality of emitted messages
QUEUE	`MQG_ATTR_EMIT_Q_NAME`	`MQCFST`	The queue to emit data to.
READER	`MQG_ATTR_READER`	`MQCFIN`	The intended reader of the emitted data.
ZEROVALS	`MQG_ATTR_ZERO_VALUES`	`MQCFIN`	Whether zero values are included in the emitted data.

11.17 DISPLAY EVENTS

Use the MQSC command **`DISPLAY EVENTS`** (or it's equivalent PCF command **`MQG_CMD_DISPLAY_EVENTS`**) to display the events processed and stored by **MQEV**.

11.17.1 Syntax diagram for DISPLAY EVENTS

```
►►─ DISPLAY EVENTS( wildcarded-stream-name ) ──┬──────────────────────────────────────►
                                               └─ EVQMGR( wildcarded-qmgr-name ) ─┘

    ┌─ DISPCMDS( ev-obj-value ) ─┐    ┌─ DISTYPE( SUMMARY ) ──────────┐
►───┤                            ├────┤                               ├──────────────────►
    └─ DISPCMDS( ─┬─ HIDE─┬─ ) ─┘    └─ DISTYPE( ─┬─ DETAIL──────┬─ ) ─┘
                  └─ SHOW ─┘                       ├─ CONDENSE ─┤
                                                   └─ MINIMUM ──┘

►───┬──────────────────────────┬──────────────────────────────────────────────────────►
    └─ EVENTID( event-id ) ─┘    └─ EVREASON( ─┬─ CFGCHGOBJ ─┬─ ) ─┘
                                               ├─ CHLNOTAVL ─┤
                                               ├─ INHPUT ────┤
                                               :             :

►───┬──────────────────────────────────────────┬──────────────────────────────────────►
    └─ EVTYPE( ─┬─ AUTHOR ──┬─ ) ─┘            └─ EVOBJTYPE( ─┬─ AUTHINFO ─┬─ ) ─┘
                ├─ CHANNEL ─┤                                 ├─ AUTHREC ──┤
                ├─ COMMAND ─┤                                 ├─ CFSTRUCT ─┤
                ├─ CONFIG ──┤                                 ├─ CHANNEL ──┤
                ├─ IMSBR ───┤                                 ├─ CHLAUTH ──┤
                ├─ INHIBIT ─┤                                 ├─ CLNTCONN ─┤
                ├─ LOCAL ───┤                                 ├─ COMMINFO ─┤
                ├─ LOGGER ──┤                                 ├─ LISTENER ─┤
                ├─ PERFM ───┤                                 ├─ NAMELIST ─┤
                ├─ REMOTE ──┤                                 ├─ PROCESS ──┤
                ├─ SSL ─────┤                                 ├─ QUEUE ────┤
                └─ STRSTP ──┘                                 ├─ QMGR ─────┤
                                                              ├─ RQMNAME ──┤
                                                              ├─ SERVICE ──┤
                                                              ├─ STGCLASS ─┤
                                                              ├─ SUB ──────┤
                                                              ├─ TOPIC ────┤
                                                              └─ TOPICSTR ─┘
    ┌─ FROM( '-24h' ) ──────────┐    ┌─ TO( 'now' ) ──────────┐
►───┤                           ├────┤                        ├──────────────────────────►
    └─ FROM( date-and-time ) ─┘    └─ TO( date-and-time ) ─┘    └─ TZ( integer ) ─┘

    ┌─ MAXRESP( 100 ) ─────┐    ┌─ MAXRECS( ev-value ) ─┐
►───┤                      ├────┤                       ├──────────────────────────────►
    └─ MAXRESP( integer ) ─┘    └─ MAXRECS( integer ) ─┘

►───┬──────────────────────────────────────────┬──────────────────────────────────────►
    └─ EVOBJNAME( wildcarded-object-name ) ─┘    └─ EVUSERID( userid ) ─┘

►───┬──────────────────────────┬──┬──────────────┬──┬──────────┬──────────────────────►◄
    └─ WHERE( FilterCondition ) ─┘  └─┤ attributes ├─┘  └─ ALL ─┘
```

11.17.2 Parameter descriptions for DISPLAY EVENTS

(wildcarded-stream-name)

The stream name from which events are to be displayed. This can be a wildcarded string.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_STREAM_NAME.

DISPCMDS

How to handle command events stored that record DISPLAY commands. This over-rides the value on the EV object. If not specified, that value is used.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_DISPLAY_COMMANDS.

Possible Values are:-

HIDE

Do not show any DISPLAY command events by default.
The PCF value for this is MQG_DISPCMDS_HIDE.

SHOW

Show DISPLAY command events.
The PCF value for this is MQG_DISPCMDS_SHOW.

DISTYPE

The type of display to return. IBM MQ Events can contain a lot of fields. It can therefore be useful to be able to choose how many of these fields are displayed by default.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_DISPLAY_TYPE.

Possible Values are:-

DETAIL

Show all event fields. This display type will display the Queue Manager, the Event Type, the Event Time, the Object and Object Type of the event.

CONDENSE

Show a condensed view. This display type will display the same fields as for `DETAIL`, but fields with no value are not shown. When displaying the `BEFORE` and `AFTER` details of a configuration change event, only the changed values are shown.

SUMMARY

Show a summary view. This is the default value. This display type will just display the Queue Manager, the Event Time and the field, `SUMMARY`, which contains a text description of the event.

MINIMUM

Show a minimum view. This display type will just display the Queue Manager and Event Type of the event. If you want to see a very specific set of event fields, use this as a starting point and add the additional fields you want to see.

Of course with any display type you can add additional fields to the display as required.

EVENTID

The unique ID of an event on a particular queue manager. If this attribute is specified, the `EVQMGR` must not be generic.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_EVENT_ID.

EVOBJNAME

The name of the object to show related events about. This can be a wildcarded string. The maximum length of this string is MQ_OBJECT_NAME_LENGTH (48).

This parameter can be used to limit the number of events that are displayed. If it is not specified, the display is not limited in this way.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_OBJECT.

EVOBJTYPE

The object type of events to be displayed.

This parameter can be used to limit the number of events that are displayed. If it is not specified, the display is not limited in this way.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_OBJECT_TYPE.

Possible Values are:-

MQSC value	Meaning	PCF constant
AUTHINFO	Authentication information object	MQOT_AUTH_INFO
AUTHREC	Authorization records (this OBJTYPE is only applicable to configuration events for AUTHRECS	MQOT_AUTH_REC
CFSTRUCT	CF Structure (this OBJTYPE is only applicable to events on z/OS queue managers)	MQOT_CF_STRUC
CHANNEL	Channel	MQOT_CHANNEL
CHLAUTH	Channel Authentication records	MQOT_CHLAUTH
CLNTCONN	Client connection channel	MQOT_CLNTCONN_CHANNEL
COMMINFO	Communication information object (this OBJTYPE is only applicable to events on non-z/OS queue managers)	MQOT_COMM_INFO
LISTENER	Listener (this OBJTYPE is only applicable to events on non-z/OS queue managers)	MQOT_LISTENER
NAMELIST	Namelist	MQOT_NAMELIST
PROCESS	Process	MQOT_PROCESS
QUEUE	Queue	MQOT_Q
QMGR	Queue manager	MQOT_Q_MGR
RQMNAME	Remote queue manager (this OBJTYPE is only applicable within configuration events for AUTHRECS)	MQOT_REMOTE_Q_MGR_NAME
SERVICE	Service object (this OBJTYPE is only applicable to events on non-z/OS queue managers)	MQOT_SERVICE
STGCLASS	Storage Class (this OBJTYPE is only applicable to events on z/OS queue managers)	MQOT_STORAGE_CLASS
SUB	Subscription	MQG_OT_SUB
TOPIC	Topic	MQOT_TOPIC
TOPICSTR	Topic String	MQG_OT_TOPICSTR

EVREASON

The reason of the events to be displayed. This parameter can be used to limit the number of events that are displayed. If it is not specified, the display is not limited in this way.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_EVENT_REASON.

Values such as the following:-

MQSC Value	Meaning	PCF Constant
AUTHOR	Any authority event	MQG_EVENT_REASON_AUTH
AUTCON	Connect not authorised	MQG_EVENT_REASON_AUTH_CONN_NOT_AUTH
AUTSYSCON	System connection not authorised	MQG_EVENT_REASON_AUTH_SYS_CONN_NOT_AUTH
AUTCSP	CSP not authorised	MQG_EVENT_REASON_AUTH_CSP_NOT_AUTH
AUTOPEN	Open not authorised	MQG_EVENT_REASON_AUTH_OPEN_NOT_AUTH
AUTCLOSE	Close not authorised	MQG_EVENT_REASON_AUTH_CLOSE_NOT_AUTH
…	…	…

For a complete list please refer to Appendix D. Event Reasons on page 215.

EVTYPE

The type of events to be displayed. This parameter can be used to limit the number of events that are displayed. If it is not specified, the display is not limited in this way.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_EVENT_TYPE.

Possible Values are:-

MQSC value	Meaning	PCF constant
AUTHOR	Authority events	MQG_EVENT_REASON_AUTH
CHANNEL	Channel events	MQG_EVENT_REASON_CHANNEL
COMMAND	Command events	MQG_EVENT_REASON_CMD
CONFIG	Configuration events	MQG_EVENT_REASON_CONFIG
IMSBR	IMS Bridge events	MQG_EVENT_REASON_BRIDGE
INHIBIT	Inhibit events	MQG_EVENT_REASON_INHIBIT
LOCAL	Local events	MQG_EVENT_REASON_LOCAL
LOGGER	Logger events	MQG_EVENT_REASON_LOGGER
PERFM	Performance events	MQG_EVENT_REASON_PERFM
REMOTE	Remote events	MQG_EVENT_REASON_REMOTE
SSL	SSL/TLS events	MQG_EVENT_REASON_SSL
STRSTP	Start/Stop events	MQG_EVENT_REASON_STRSTP
IMSBR	IMS Bridge events	MQG_EVENT_REASON_IMSBRIDGE
UNKNOWN	Unknown events	MQG_EVENT_REASON_UNKNOWN

EVQMGR

The queue manager associated with the events to be displayed. This can be a wildcarded string.

This parameter can be used to limit the number of events that are displayed. If it is not specified, the display is not limited in this way.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_Q_MGR_NAME.

EVUSERID

The userid (if any) that is associated with the event. The maximum length of this string is MQ_USER_ID_LENGTH (12).

This parameter can be used to limit the number of events that are displayed. If it is not specified, the display is not limited in this way.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_USER_ID.

MAXRECS

The number of source records which should be used to construct the response. This prevents inadvertent consumption of CPU when issuing queries against large amounts of data.

The default value is taken from the **EV** object which itself has a default value of 1,000,000.

This value can not be larger than 100,000,000.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_MAX_RECORDS.

MAXRESP

The number of responses to be returned to this DISPLAY command. The default value is 100.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_MAX_RESPONSES.

FROM

The time before which no events should be returned. The maximum length of this string is MQG_DATE_TIME_LENGTH (30).

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_FROM.

Many different parameter formats are supported, the time can be specified in either absolute or relative terms. The following are supported:

Absolute values	**Meaning**
now	The current time
8	Eight AM
8.30	Eight thirty AM
8.31.46	Eight thirty-one and 46 seconds
04-12	12th April (this year) The day and month fields must always be two digits
2018-10-18	18th October 2018 The year field must always be four digits
2018-10-18 8.31.46	An explicit date and time
Relative values	**Relative to 'the other' time parameter** **Note that both times can not be relative**
-2day (or -2d)	Two days before

+1d	One day after
-4hour (or -4hr or -4h)	Four hours before
-3minute (or -3min or -3m)	Three minutes before
-10second (-10 sec or -10s or -10)	Ten seconds before
-1d4h3m6s	Values can be combined without spaces

If not specified then the value '-24hour' will be used.

TO

The time after which no events should be returned. The maximum length of this string is MQG_DATE_TIME_LENGTH (30).

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_TO.

Please see the description of the `FROM` parameter for the allowed values.

If not specified then the value 'now' will be used.

TZ

The bias, in minutes, of the time zone that the results should be returned in.

When using the PCF interface, this is an MQCFIN parameter with identifier MQC_ATTR_TIMEZONE.

If not specified then the results will be returned in UTC.

WHERE

Specify a filter condition to only display events that satisfy the selection criterion of the filter condition. For more about the WHERE clause see Chapter 8 Where Clause() on page 42.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_WHERE.

EVENTS Attributes

The attribute list can specify any of the following values. When using the PCF interface, this is an MQCFIL parameter with identifier MQG_ATTR_EV_EVENTS_ATTRS.

MQSC Value	PCF Constant	Description
`ALL`	`MQIACF_ALL`	All attributes
Event Field	As per IBM MQ command reference	

11.18 DISPLAY EVQ

Use the MQSC command **DISPLAY EVQ** (or it's equivalent PCF command **MQG_CMD_DISPLAY_EV_Q**) to display the the event queues processed by the **MQEV** event processor.

11.18.1 Syntax diagram for DISPLAY EVQ

```
►►─ DISPLAY EVQ( wildcarded-queue-name ) ─┬──────────────────────────────┬──►
                                          └─ WHERE( FilterCondition ) ─┘

►─┬──────────────┬─┬──────────────┬─┬─────────┬──────────────────────────────►◄
  ├─ ACCTMQICT ─┤ ├─ FWDPSIST ──┤ └─ ALL ──┘
  ├─ ACCTQCT ───┤ ├─ LSTMSGTI───┤
  ├─ BORQCT ────┤ ├─ MSGS ──────┤
  ├─ DESCR──────┤ ├─ STATCHLCT ─┤
  ├─ DLQCT ─────┤ ├─ STATMQICT ─┤
  ├─ ERRORCT ───┤ ├─ STATQCT ───┤
  ├─ EVENTCT ───┤ ├─ STATUS ────┤
  └─ FWDQ ──────┘ ├─ SUSPENDED ─┤
                  └─ TEMPQ ─────┘
```

11.18.2 Parameter descriptions for DISPLAY EVQ

(wildcarded-queue-name)

The event queue name to be displayed. This can be a wildcarded string.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_EVENT_Q_NAME.

WHERE

Specify a filter condition to only display event queues that satisfy the selection criterion of the filter condition. For more about the WHERE clause see Chapter 8 Where Clause() on page 42.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_WHERE.

EVQ Attributes

The attribute list can specify any of the following values. When using the PCF interface, this is an MQCFIL parameter with identifier MQG_ATTR_EV_QUEUE_ATTRS.

MQSC Value	PCF Constant	PCF Type	Description
ALL	MQIACF_ALL	N/A	All attributes
ACCTMQICT	MQG_ATTR_ACCT_MQI_COUNT	MQCFIN	The number of MQI Accounting records processed on this queue since last RESET. The total of this value for all queues on this queue manager will equal DISPLAY EVQMGR ACCTMQICT
ACCTQCT	MQG_ATTR_ACCT_Q_COUNT	MQCFIN	The number of Queue Accounting records processed on this queue since last RESET. The total of this value for all queues on this queue manager will equal DISPLAY EVQMGR ACCTQCT
BORQCT	MQG_ATTR_BORQ_COUNT	MQCFIN	The number of poison messages that have been written to the Backout Requeue queue.

MQSC Value	PCF Constant	PCF Type	Description
DESCR	MQG_ATTR_EVENT_Q_DESC	MQCFST	A description of the event queue.
DLQCT	MQG_ATTR_DLQ_COUNT	MQCFIN	The number of messages successfully sent to the dead-letter queue. An error will be written to the **MQEV** log indicating the reason given for any failed puts to the dead-letter queue.
ERRORCT	MQG_ATTR_ERROR_COUNT	MQCFIN	The number of messages found on this queue which were not event messages.
EVENTCT	MQG_ATTR_EVENT_COUNT	MQCFIN	The number of event message processed on this queue since last RESET. The total of this value for all queues on this queue manager will equal DISPLAY EVQMGR EVENTCT
FWDQ	MQG_ATTR_FORWARD_Q_NAME	MQCFST	The queue name event messages should be forwarded to for daisy-chaining purposes.
FWDPSIST	MQG_ATTR_FORWARD_PERSISTENCE	MQCFIN	The persistence of messages that are forwarded to the queue named in FWDQ.
LSTMSGTI	MQG_ATTR_LAST_MSG_TIME	MQCFIN64	The time when the last message was processed from this queue. [21]
MSGS	MQG_ATTR_NUM_MESSAGES	MQCFIN	The number of messages processed from this queue since this instance of MQEV started up.
STATCHLCT	MQG_ATTR_STAT_CHL_COUNT	MQCFIN	The number of Channel Statistics records processed on this queue since last RESET. The total of this value for all queues on this queue manager will equal DISPLAY EVQMGR STATCHLCT
STATMQICT	MQG_ATTR_STAT_MQI_COUNT	MQCFIN	The number of MQI Statistics records processed on this queue since last RESET. The total of this value for all queues on this queue manager will equal DISPLAY EVQMGR STATMQICT
STATQCT	MQG_ATTR_STAT_Q_COUNT	MQCFIN	The number of Queue Statistics records processed on this queue since last RESET. The total of this value for all queues on this queue manager will equal DISPLAY EVQMGR STATQCT
STATUS	MQG_ATTR_Q_STATUS	MQCFIN	Whether this queue is being successfully used by **MQEV**.
SUSPENDED	MQG_ATTR_SUSPENDED	MQCFIN	Whether this queue is currently suspended from processing.
TEMPQ	MQG_ATTR_TEMPQ_DISP	MQCFIN	How statistics records for temporary queues are handled.

21 The number of seconds since 1st January 1970 – also known as Unix time or Epoch time.

11.19 DISPLAY EVQMGR

Use the MQSC command **DISPLAY EVQMGR** (or it's equivalent PCF command **MQG_CMD_DISPLAY_EV_Q_MGR**) to display the details about each different queue manager for which items (events, accounting and statistics records) have been processed.

The event counts displayed on this command can be reset using the **RESET EV** command. These counts are not affected by event data being expired when it passes its retention interval. It is expected that counts will be reset on a more frequent basis than data will be expired.

11.19.1 Syntax diagram for DISPLAY EVQMGR

```
►►─ DISPLAY EVQMGR( wildcarded-qmgr-name ) ──┬──────────────────────────────┬──►
                                             └─ WHERE( FilterCondition ) ─┘

►──┬──────────────┬──┬──────────────┬──┬──────────────┬──┬─────────┬──►◄
   ├─ ACCTMQICT ─┤  ├─ EVENTCT ───┤  ├─ REMOTECT ──┤  └─ ALL ──┘
   ├─ ACCTQCT ───┤  ├─ IMSBRCT ───┤  ├─ SSLCT ─────┤
   ├─ ALERTCT ───┤  ├─ INHIBITCT ─┤  ├─ STATCHLCT ─┤
   ├─ AUTHORCT ──┤  ├─ LOCALCT ───┤  ├─ STATMQICT ─┤
   ├─ CHLCT ─────┤  ├─ LOGGERCT ──┤  ├─ STATQCT ───┤
   ├─ CMDCT ─────┤  └─ PERFMCT ───┘  └─ STRSTPCT ──┘
   └─ CONFIGCT ──┘
```

11.19.2 Parameter descriptions for DISPLAY EVQMGR

(wildcarded-qmgr-name)

The queue manager name to be displayed. This can be a wildcarded string.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_Q_MGR_NAME.

WHERE

Specify a filter condition to only display queue managers that satisfy the selection criterion of the filter condition. For more about the WHERE clause see Chapter 8 Where Clause() on page 42.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_WHERE.

EVQMGR Attributes

The attribute list can specify any of the following values. When using the PCF interface, this is an MQCFIL parameter with identifier MQG_ATTR_EV_QMGR_ATTRS.

The total number of all the event type counts will equal the `EVENTCT` attribute.

MQSC Value	PCF Constant	PCF Type	Description
ALL	MQIACF_ALL	N/A	All attributes
ACCTQCT	MQG_ATTR_ACCT_Q_COUNT	MQCFIN	The number of Queue Accounting records processed for this queue manager. These are the records controlled by the IBM MQ ACCTQ switch.
ACCTMQICT	MQG_ATTR_ACCT_MQI_COUNT	MQCFIN	The number of MQI Accounting records processed for this queue manager. These are the records controlled by the IBM MQ ACCTMQI switch.

MQSC Value	PCF Constant	PCF Type	Description
ALERTCT	MQG_ATTR_ALERT_COUNT	MQCFIN	The number of alerts for this queue manager. The total of this number for all queue managers will equal DISPLAY EV ALERTS.
AUTHORCT	MQG_ATTR_EVENT_AUTHOR_COUNT	MQCFIN	The number of Authority events processed for this queue manager. These are the events controlled by the IBM MQ AUTHOREV switch.
CHLCT	MQG_ATTR_EVENT_CHL_COUNT	MQCFIN	The number of Channel events processed for this queue manager. These are the events controlled by the IBM MQ CHLEV switch.
CMDCT	MQG_ATTR_EVENT_CMD_COUNT	MQCFIN	The number of Command events processed for this queue manager. These are the events controlled by the IBM MQ CMDEV switch.
CONFIGCT	MQG_ATTR_EVENT_CONFIG_COUNT	MQCFIN	The number of Configuration events processed for this queue manager. These are the events controlled by the IBM MQ CONFIGEV switch.
EVENTCT	MQG_ATTR_EVENT_COUNT	MQCFIN	The total number of events processed for this queue manager. The total of this number for all queue managers will equal DISPLAY EV TOTALEV.
IMSBRCT	MQG_ATTR_EVENT_BRIDGE_COUNT	MQCFIN	The number of IMS Bridge events processed for this queue manager.
INHIBITCT	MQG_ATTR_EVENT_INHIBIT_COUNT	MQCFIN	The number of Inhibit events processed for this queue manager. These are the events controlled by the IBM MQ INHIBITEV switch.
LOCALCT	MQG_ATTR_EVENT_LOCAL_COUNT	MQCFIN	The number of Local events processed for this queue manager. These are the events controlled by the IBM MQ LOCALEV switch.
LOGGERCT	MQG_ATTR_EVENT_LOGGER_COUNT	MQCFIN	The number of Logger events processed for this queue manager. These are the events controlled by the IBM MQ LOGGEREV switch.
PERFMCT	MQG_ATTR_EVENT_PERFM_COUNT	MQCFIN	The number of Performance events processed for this queue manager. These are the events controlled by the IBM MQ PERFMEV switch.
REMOTECT	MQG_ATTR_EVENT_REMOTE_COUNT	MQCFIN	The number of Remote events processed for this queue manager. These are the events controlled by the IBM MQ REMOTEEV switch.
SSLCT	MQG_ATTR_EVENT_SSL_COUNT	MQCFIN	The number of SSL events processed for this queue manager. These are the events controlled by the IBM MQ SSLEV switch.
STATCHLCT	MQG_ATTR_STAT_CHL_COUNT	MQCFIN	The number of Channel Statistics records processed for this queue manager. These are the records controlle by the IBM MQ STATCHL switch.
STATMQICT	MQG_ATTR_STAT_MQI_COUNT	MQCFIN	The number of MQI Statistics records processed for this queue manager. These are the records controlled by the IBM MQ STATMQI switch.
STATQCT	MQG_ATTR_STAT_Q_COUNT	MQCFIN	The number of Queue Statistics records processed for this queue manager. These are the records controlled by the IBM MQ STATQ switch.
STRSTPCT	MQG_ATTR_EVENT_STRSTP_COUNT	MQCFIN	The number of Start/Stop events processed for this queue manager. These are the events controlled by the IBM MQ STRSTPEV switch.

11.20 DISPLAY EVSTREAM

Use the MQSC command **DISPLAY EVSTREAM** (or it's equivalent PCF command **MQG_CMD_DISPLAY_EV_STREAM**) to display the attributes of a stream.
Streams are used to store the records (events, accounting and statistics) processed by **MQEV**.

11.20.1 Syntax diagram for DISPLAY EVSTREAM

```
►►─ DISPLAY EVSTREAM( wildcarded-stream-name ) ──┬──────────────────────────────┬──►
                                                 └─TYPE( ─┬─ EVENTS ─────┬─ ) ─┘
                                                          ├─ STATQ ──────┤
                                                          ├─ STATCHL ────┤
                                                          ├─ STATMQI ────┤
                                                          ├─ ACCTQ ──────┤
                                                          └─ ACCTMQI ────┘

►─┬──────────────────────────────┬─┬──────────────────────┬─┬──────────────┬──────────►◄
  └─ WHERE( FilterCondition ) ─┘   ├─── DEFAULT ─────┤   └─── ALL ────┘
                                   ├─── DESCR ───────┤
                                   ├─── EVEMIT ──────┤
                                   ├─── RETINTVL ────┤
                                   └─── TYPE ────────┘
```

11.20.2 Parameter descriptions for DISPLAY EVSTREAM

(wildcarded-stream-name)

The stream name to be displayed. This can be a wildcarded string.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_STREAM_NAME.

TYPE

The type of streams to display.

This parameter can be used to limit the number of streams that are displayed. If it is not specified, the display is not limited in this way.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_STREAM_TYPE.

Possible values are:-

EVENTS

Display only streams that contain event data.
The PCF value for this is MQG_STREAM_TYPE_EVENTS.

STATQ

Display only streams that contain queue statistics data.
The PCF value for this is MQG_STREAM_TYPE_STAT_Q.

STATCHL

Display only streams that contain channel statistics data.
The PCF value for this is MQG_STREAM_TYPE_STAT_CHL.

STATMQI

Display only streams that contain MQI statistics data.
The PCF value for this is MQG_STREAM_TYPE_STAT_MQI.

ACCTQ

Display only streams that contain queue accounting data.
The PCF value for this is MQG_STREAM_TYPE_ACCT_Q.

ACCTMQI

Display only streams that contain MQI accounting data.
The PCF value for this is MQG_STREAM_TYPE_ACCT_MQI.

WHERE

Specify a filter condition to only display streams that satisfy the selection criterion of the filter condition. For more about the WHERE clause see Chapter 8 Where Clause() on page 42.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_WHERE.

EVSTREAM Attributes

The attribute list can specify any of the following values. When using the PCF interface, this is an MQCFIL parameter with identifier MQG_ATTR_EV_STREAM_ATTRS.

MQSC Value	PCF Constant	PCF Type	Description
ALL	MQIACF_ALL	N/A	All attributes
DEFAULT	MQG_ATTR_DEFAULT_STREAM	MQCFIN	Whether this is the default stream.
DESCR	MQG_ATTR_STREAM_DESC	MQCFST	A text description of the stream.
EVEMIT	MQG_ATTR_EMIT_NAME	MQCFST	The name of an EVEMIT object.
RETINTVL	MQG_ATTR_RETENTION_INTERVAL	MQCFIN	The retention interval for data on this stream (in days).
TYPE	MQG_ATTR_STREAM_TYPE	MQCFIN	The type of data on this stream.

11.21 DISPLAY EVSTRMST

Use the MQSC command **`DISPLAY EVSTRMST`** (or it's equivalent PCF command **`MQG_CMD_DISPLAY_EV_STREAM_STATUS`**) to display the run-time status of streams in use on various queue managers.

Streams are used to store the items (events, accounting and statistics records) processed by MQEV.

11.21.1 Syntax diagram for DISPLAY EVSTRMST

```
►►─ DISPLAY EVSTRMST( wildcarded-stream-name ) ──┬──────────────────────────────┬──►
                                                 └─TYPE( ─┬─ EVENTS ─────┬─ ) ─┘
                                                          ├─ STATQ ──────┤
                                                          ├─ STATCHL ────┤
                                                          ├─ STATMQI ────┤
                                                          ├─ ACCTQ ──────┤
                                                          └─ ACCTMQI ────┘

►─┬──────────────────────────────────┬─┬──────────────────────────────────────────►
  └─ EVQMGR( wildcarded-qmgr-name ) ─┘ └─ WHERE( FilterCondition ) ─┘

►─┬──────────────────┬─┬──────────────────────────────────────────────────────►◄
  ├── DATAMSGS ──────┤ └── ALL ──┘
  ├── EVQMGR ────────┤
  ├── ITEMS ─────────┤
  ├── LSTMSGTI───────┤
  ├── MEMORY ────────┤
  ├── OLDEST ────────┤
  └── TYPE ──────────┘
```

11.21.2 Parameter descriptions for DISPLAY EVSTRMST

(wildcarded-stream-name)

The stream name to be displayed. This can be a wildcarded string.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_STREAM_NAME.

EVQMGR

The queue manager associated with the stream-name to be displayed. This can be a wildcarded string.

This parameter can be used to limit the number of streams that are displayed. If it is not specified, the display is not limited in this way.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_Q_MGR_NAME.

TYPE

The type of streams to display.

This parameter can be used to limit the number of streams that are displayed. If it is not specified, the display is not limited in this way.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_STREAM_TYPE.

Possible values are:-

EVENTS

Display only streams that contain event data.
The PCF value for this is MQG_STREAM_TYPE_EVENTS.

STATQ

Display only streams that contain queue statistics data.
The PCF value for this is MQG_STREAM_TYPE_STAT_Q.

STATCHL

Display only streams that contain channel statistics data.
The PCF value for this is MQG_STREAM_TYPE_STAT_CHL.

STATMQI

Display only streams that contain MQI statistics data.
The PCF value for this is MQG_STREAM_TYPE_STAT_MQI.

ACCTQ

Display only streams that contain queue accounting data.
The PCF value for this is MQG_STREAM_TYPE_ACCT_Q.

ACCTMQI

Display only streams that contain MQI accounting data.
The PCF value for this is MQG_STREAM_TYPE_ACCT_MQI.

WHERE

Specify a filter condition to only display status information for those streams that satisfy the selection criterion of the filter condition. For more about the WHERE clause see Chapter 8 Where Clause() on page 42.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_WHERE.

EVSTRMST Attributes

The attribute list can specify any of the following values. When using the PCF interface, this is an MQCFIL parameter with identifier MQG_ATTR_EV_STREAM_STATUS_ATTRS.

MQSC Value	PCF Constant	PCF Type	Description
`ALL`	MQIACF_ALL	N/A	All attributes
`DATAMSGS`	MQG_ATTR_DATA_MSGS	MQCFIN	The number of messages being used to store data for this stream. This will be less than the number of items since many items are compressed into each message.
`EVQMGR`	MQG_ATTR_Q_MGR_NAME	MQCFST	The queue manager name related to this stream.
`LSTMSGTI`	MQG_ATTR_LAST_MSG_TIME	MQCFIN64	The time when the last messages was added to this stream[22]. If this is empty, no messages have been added to this stream since **MQEV** last started.
`ITEMS`	MQG_ATTR_NUM_ITEMS	MQCFIN	The number of items (events, accounting or statistics records) that have been stored on this stream.
`MEMORY`	MQG_ATTR_MEMORY	MQCFIN	The number of kilobytes of data stored on this stream (rounded up).
`OLDEST`	MQG_ATTR_OLDEST_ITEM	MQCFIN	The age, in days, of the oldest item stored on this stream.
`TYPE`	MQG_ATTR_STREAM_TYPE	MQCFIN	The type of this stream.

22 The number of seconds since 1st January 1970 – also known as Unix time or Epoch time.

11.22 DISPLAY STATCHL

Use the MQSC command **`DISPLAY STATCHL`** (or it's equivalent PCF command **`MQG_CMD_DISPLAY_STAT_CHL`**) to display the channel statistics records processed and stored by **MQEV**. For a description of the statistics fields provided by MQ please read the MQ documentation

11.22.1 Syntax diagram for DISPLAY STATCHL

```
►►─ DISPLAY STATCHL( wildcarded-channel-name ) ─┬──────────────────────────┬──►
                                                └─ EVQMGR( qmgr-name ) ─┘

    ┌─ GAPFILL( NO ) ──────────────┐    ┌─ SUM( CHANNEL ) ──────────┐
►───┤                              ├────┤                           ├──────────►
    └─ GAPFILL( ─┬─ INNER ────┬─ ) ─┘    ├─ SUM( ─┬─ NONE ───┬─ ) ─┤
                 ├─ ACTIVITY ─┤          │        └─ TOTAL ──┘      │
                 └─ FULL ─────┘          │        ┌─,──────────┐    │
                                         │        ↓            │    │
                                         └─ SUM( ─┬─ CHANNEL ──┬─ ) ─┘
                                                  ├─ CONNAME ──┤
                                                  ├─ RQMNAME ──┤
    ┌─ ZEROVALS( HIDE ) ─┐                        └─ STREAM ───┘
►───┤                    ├──────────────────────────────────────────────────►
    └─ ZEROVALS( SHOW ) ─┘

    ┌─ FROM( '-24h' ) ──────┐   ┌─ TO( 'now' ) ──────────┐   ┌─ TZ( 0 ) ──────┐
►───┤                       ├───┤                        ├───┤                ├──►
    └─ FROM( date-and-time ) ─┘   └─ TO( date-and-time ) ─┘   └─ TZ( integer ) ─┘

    ┌─ INTVLALGN( CLOCK ) ─┐   ┌─ INTVL( ' ' ) ──────┐
►───┤                      ├───┤                     ├──────────────────────►
    └─ INTVLALGN( MQ ) ────┘   └─ INTVL( time-period ) ─┘

►──┬──────────────────────────────────────────────────┬──┬──────────────────────────────────────────────────┬──►
   └─ INTVLSTA( date-and-time-seconds ) ─┘            └─ INTVLEND( date-and-time-seconds ) ─┘

►──┬──────────────────────────────────┬──┬──────────────────────────────────┬──►
   └─ WHERE( FilterCondition ) ─┘       └─ PREWHERE( FilterCondition ) ─┘

    ┌─ MAXRECS( ev-value ) ─┐   ┌─ MAXRESP( 100 ) ──────┐
►───┤                       ├───┤                       ├──┬──────────────┬──┬─────┬──►◄
    └─ MAXRECS( integer ) ──┘   └─ MAXRESP( integer ) ─┘  └─┤ attributes ├─┘  └─ ALL ─┘
```

11.22.2 Parameter descriptions for DISPLAY STATCHL

(wildcarded-channel-name)

The queue name for which statistics records are to be displayed. This can be a wildcarded string.

When using the PCF interface, this is an MQCFST parameter with identifier MQCACH_CHANNEL_NAME.

EVQMGR

The queue manager associated with the records to be displayed.

This parameter is used to specify which queue manager you wish to see data from. If it is not specified, the output shows records from the queue manager you are connected to.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_Q_MGR_NAME.

GAPFILL

Whether to add zeroed records where none exist to produce a set of records that will graph appropriately.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_GAP_FILL.

Possible Values are:-

NO

Do not fill in the gaps with zeroed records.
The PCF value for this is MQG_GAPFILL_NO.

INNER

Produce zeroed records for inner gaps in the set of records.
The PCF value for this is MQG_GAPFILL_INNER.

ACTIVITY

Add zeroed records to produce a set of records that spans the time period where there is any activity on this queue, not just the activity shown by the attributes displayed.
The PCF value for this is MQG_GAPFILL_ACTIVITY.

FULL

Add zeroed records to produce a set of records that spans the entire time period requested.
The PCF value for this is MQG_GAPFILL_FULL

SUM

Whether the records are summed together, and if so how they are totalled. Multiple values can be provided in a comma-separated list, except where indicated that a value cannot be combined with any others.

When multiple values are provided, a record will be returned for each unique combination, e.g. `SUM(CHANNEL, CONNAME)` will return one record for each unique combination of channel name and connection name.

When using the PCF interface, this can be an MQCFIN parameter (should you only need to supply one value) or an MQCFIL parameter (should you need to supply multiple values) with identifier MQG_ATTR_SUM.

When any of the `SUM` values apart from `NONE` and `TOTAL` are used without the `INTVL` attribute, this will result in a single record per unique key combination being returned. If used with the `INTVL` attribute, one record per application will be returned for each interval.

For more information about the returned records and their interval start and end times, please see Chapter 10 Returned Interval Times on page 48.

Possible Values are:-

NONE

Do not add together any records. The command returns each individual record as reported by IBM MQ. This value cannot be combined with any others.
The PCF value for this is MQG_SUM_NONE.

CHANNEL

Add together records that have the same channel name. If used without the `INTVL` attribute, this will result in a single record per queue being returned. If used with the `INTVL` attribute, one record per channel will be returned for each interval. This is the default value.
The PCF value for this is MQG_SUM_CHANNEL_NAME.

CONNAME

A record will be returned for each unique connection name
The PCF value for this is MQG_SUM_CONNECTION_NAME.

RQMNAME

A record will be returned for each unique connection name
The PCF value for this is MQG_SUM_REMOTE_QMGR.

STREAM

A record will be returned for each MQEV stream
The PCF value for this is MQG_SUM_STREAM.

TOTAL

Add together all of the records that match the other criteria (e.g. `WHERE` clause) on the display command. If used without the `INTVL` attribute, this will result in only a single record being returned. If used with the `INTVL` attribute, one record will be returned for each interval. The queue name reported on the returned record will reflect the fact that the numbers for many queue names might have been added together.
Using `SUM(TOTAL)` may be very useful when treating multiple queue names as the same name for the purposes of monitoring or charge-back. For example, a set of dynamic queues that all have similar names, could be totalled as if they were a single queue.
This value cannot be combined with any others.
The PCF value for this is MQG_SUM_TOTAL.

ZEROVALS

How to handle zero values in the records displayed.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_ZERO_VALUES.

Possible Values are:-

HIDE

Do not show any zero values. This is the default value.
The PCF value for this is MQG_ZEROVALS_HIDE.

SHOW

Show zero values in records.
The PCF value for this is MQG_ZEROVALS_SHOW.

INTVLALGN

How to align the reported intervals of records. This parameter is only used when an `INTVL` is set.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_INTERVAL_ALIGN.

CLOCK

Reported intervals are aligned to the clock. That is, if you request an `INTVL(4hour)`, reported intervals will be at midnight, 4am, 8am, noon and so on. This is the default value.
The PCF value for this is MQG_ALIGN_CLOCK.

MQ

Reported intervals are aligned to the intervals reported by IBM MQ.
The PCF value for this is MQG_ALIGN_MQ.

INTVL

When summing records, the reported records are totalled in intervals of the specified length. See `INTVLALGN` for details of the time alignment of these intervals. This parameter is only valid when you are not using `SUM(NONE)`. The maximum length of this string is MQG_DATE_TIME_LENGTH (30).

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_INTERVAL.

The interval supplied must be a multiple of the interval used in IBM MQ to collect the statistics in the first instance, that is the `STATINT` value. If you are collecting statistics records every 2 hours, you cannot request that MQEV display records summed into intervals of 45 minutes.

For more information about the returned records and their interval start and end times, please see Chapter 10 Returned Interval Times on page 48.

Many different parameter formats are supported to provide an interval. The following are supported:

Values	**Meaning**
2day (or 2d)	Two days
4hour (or 4hr or 4h)	Four hours
3minute (or 3min or 3m)	Three minutes
1d4h	Values can be combined without spaces

FROM

The time from which records should be returned. The maximum length of this string is MQG_DATE_TIME_LENGTH (30).

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_FROM.

For more information about the returned records and their interval start and end times, please see Chapter 10 Returned Interval Times on page 48.

Many different parameter formats are supported, the time can be specified in either absolute or relative terms. The following are supported:

Absolute values	**Meaning**
now	The current time
8	Eight AM
8.30	Eight thirty AM
8.31.46	Eight thirty-one and 46 seconds
04-12	12th April (this year) The day and month fields must always be two digits
2018-10-18	18th October 2018 The year field must always be four digits
2018-10-18 8.31.46	An explicit date and time
Relative values	**Relative to 'the other' time parameter** **Note that both times can not be relative**
-2day (or -2d)	Two days before
+1d	One day after
-4hour (or -4hr or -4h)	Four hours before
-3minute (or -3min or -3m)	Three minutes before
-10second (-10 sec or -10s or -10)	Ten seconds before
-1d4h3m6s	Values can be combined without spaces

If not specified then the value '-24hour' will be used.

TO

The time up to when records should be returned. The maximum length of this string is MQG_DATE_TIME_LENGTH (30).

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_TO.

For more information about the returned records and their interval start and end times, please see Chapter 10 Returned Interval Times on page 48.

Please see the description of the `FROM` parameter for the allowed values.

If not specified then the value 'now' will be used.

TZ

The bias, in minutes, of the time zone that the `FROM` and `TO` parameters are specified in, and any provided `INTVL` will also be aligned to this time zone.

When using the PCF interface, this is an MQCFIN parameter with identifier MQC_ATTR_TIMEZONE.

If not specified then the time zone will be assumed to be UTC.

Here are some examples:-

Time zone	TZ Value
Auckland, New Zealand	`TZ(-720)`
UTC	`TZ(0)`
Portland, Oregon, USA	`TZ(480)`

If you are using MO71 or MQSCX, you do not need to manually provide this attribute as those tools automatically include it from known configuration in those tools.

INTVLSTA

This is an integer representation[23] of the date and time of the start of the interval.

When using the PCF interface, this is an MQCFIN64 parameter with identifier MQG_ATTR_START_OF_INTERVAL. This can be both and input and output parameter.

Either use `INTVLSTA` and `INTVLEND`, or use `FROM` and `TO`. You cannot use both. It is expected that users will mainly use `FROM` and `TO`. `INTVLSTA` and `INTVLEND` are designed for programmable interfaces to input values previously returned on earlier commands..

For more information about the returned records and their interval start and end times, please see Chapter 10 Returned Interval Times on page 48.

INTVLEND

This is an integer representation[23] of the date and time of the end of the interval.

When using the PCF interface, this is an MQCFIN64 parameter with identifier MQG_ATTR_END_OF_INTERVAL. This can be both and input and output parameter.

Please refer to the description of INTVLSTA for advice on when to use this parameter.

23 The number of seconds since 1st January 1970 – also known as Unix time or Epoch time.

MAXRECS

The number of source records which should be used to construct the response. This prevents inadvertent consumption of CPU when issuing queries against large amounts of data.

The default value is taken from the **EV** object which itself has a default value of 1,000,000.

This value can not be larger than 100,000,000.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_MAX_RECORDS.

MAXRESP

The number of responses to be returned to this DISPLAY command. The default value is 100.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_MAX_RESPONSES.

PREWHERE

Specify a filter condition to only total records that satisfy the selection criterion of the filter condition. For more about the WHERE clause see Chapter 8 Where Clause() on page 42.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_PREWHERE.

WHERE

Specify a filter condition to only display records that satisfy the selection criterion of the filter condition. For more about the WHERE clause see Chapter 8 Where Clause() on page 42.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_WHERE.

STATCHL Attributes

The attribute list can specify any of the following values. When using the PCF interface, this is an MQCFIL parameter with identifier MQG_ATTR_EV_STAT_CHL_ATTRS.

For a description of these fields please read the MQ documentation

MQSC Value	PCF Constant	PCF Type	Description
ALL	MQIACF_ALL	N/A	All attributes
STATCHL	MQCACH_CHANNEL_NAME	MQCFST	Channel Name
CHLTYPE	MQIACH_CHANNEL_TYPE	MQCFIN	Channel Type
CMDLEVEL	MQIA_COMMAND_LEVEL	MQCFIN	IBM MQ Command Level
CONNAME	MQCACH_CONNECTION_NAME	MQCFST	Connection Name
RQMNAME	MQCA_REMOTE_Q_MGR_NAME	MQCFST	Remote Queue Manager Name
MSGS	MQG_ACCST_MSGS	MQCFIN	The number of messages sent and received.
BYTES	MQG_ACCST_64_BYTES	MQCFIN64	The number of bytes sent and received
NETTIMMIN	MQG_ACCST_NET_TIME_MIN	MQCFIN	The shortest recorded channel round-trip
NETTIMAVG	MQG_ACCST_NET_TIME_AVG	MQCFIN	The average recorded channel round-trip
NETTIMMAX	MQG_ACCST_NET_TIME_MAX	MQCFIN	The longest recorded channel round-trip
EXTTIMMIN	MQG_ACCST_EXIT_TIME_MIN	MQCFIN	The shortest recorded time executing a channel exit
EXTTIMAVG	MQG_ACCST_EXIT_TIME_AVG	MQCFIN	The average recorded time executing a channel exit
EXTTIMMAX	MQG_ACCST_EXIT_TIME_MAX	MQCFIN	The longest recorded time executing a channel exit

FULLBATCH	MQG_ACCST_FULL_BATCHES	MQCFIN	The number of full batches sent in the interval
ICMPBATCH	MQG_ACCST_INCOMPLETE_BATCHES	MQCFIN	The number of incomplete batches sent in the interval
AVGBATCH	MQG_ACCST_AVG_BATCHES	MQCFIN	The average batch size during the interval
PUTRETRIES	MQG_ACCST_PUT_RETRIES	MQCFIN	The number of times a message failed to be put and entered a retry loop.
RECORDS	MQG_ATTR_RECORDS	MQCFIN	Number of records totalled together

11.23 DISPLAY STATMQI

Use the MQSC command **`DISPLAY STATMQI`** (or it's equivalent PCF command **`MQG_CMD_DISPLAY_STAT_MQI`**) to display the MQI statistics records processed and stored by **MQEV**. For a description of the statistics fields provided by MQ please read the MQ documentation

11.23.1 Syntax diagram for DISPLAY STATMQI

```
►►─ DISPLAY STATMQI( wildcarded-stream-name ) ──┬──────────────────────────────┬──►
                                                └─ EVQMGR( qmgr-name ) ────────┘

    ┌─ GAPFILL( NO ) ──────────────┐  ┌─ SUM( STREAM ) ──────┐  ┌─ ZEROVALS( HIDE ) ─┐
►───┤                              ├──┤                      ├──┤                    ├──►
    └─ GAPFILL( ─┬─ INNER ────┬─ ) ┘  └─ SUM( ─┬─ NONE ───┬─ ) ┘  └─ ZEROVALS( SHOW ) ─┘
                 ├─ ACTIVITY ─┤                └─ TOTAL ──┘
                 └─ FULL ─────┘

    ┌─ FROM( '-24h' ) ──────┐  ┌─ TO( 'now' ) ────────┐  ┌─ TZ( 0 ) ──────┐
►───┤                       ├──┤                      ├──┤                ├──────►
    └─ FROM( date-and-time ) ┘  └─ TO( date-and-time ) ┘  └─ TZ( integer ) ┘

    ┌─ INTVLALGN( CLOCK ) ─┐  ┌─ INTVL( ' ' ) ──────┐
►───┤                      ├──┤                     ├──────────────────────────►
    └─ INTVLALGN( MQ ) ────┘  └─ INTVL( time-period ) ┘

►───┬──────────────────────────────┬──┬──────────────────────────────────┬──────►
    └─ INTVLSTA( date-and-time-seconds ) ┘  └─ INTVLEND( date-and-time-seconds ) ┘

►───┬──────────────────────────┬──┬──────────────────────────────┬──────────────►
    └─ WHERE( FilterCondition ) ┘  └─ PREWHERE( FilterCondition ) ┘

    ┌─ MAXRECS( ev-value ) ─┐  ┌─ MAXRESP( 100 ) ──────┐
►───┤                       ├──┤                       ├──┬──────────────┬──┬─────┬──►◄
    └─ MAXRECS( integer ) ──┘  └─ MAXRESP( integer ) ──┘  └─┤ attributes ├─┘  └─ ALL ─┘
```

11.23.2 Parameter descriptions for DISPLAY STATMQI

(wildcarded-stream-name)

The stream name for which statistics records are to be displayed. This can be a wildcarded string.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_STREAM_NAME.

EVQMGR

The queue manager associated with the records to be displayed.

This parameter is used to specify which queue manager you wish to see data from. If it is not specified, the output shows records from the queue manager you are connected to.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_Q_MGR_NAME.

GAPFILL

Whether to add zeroed records where none exist to produce a set of records that will graph appropriately.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_GAP_FILL.

Possible Values are:-

NO

Do not fill in the gaps with zeroed records.
The PCF value for this is MQG_GAPFILL_NO.

INNER

Produce zeroed records for inner gaps in the set of records.
The PCF value for this is MQG_GAPFILL_INNER.

ACTIVITY

Add zeroed records to produce a set of records that spans the time period where there is any activity on this queue, not just the activity shown by the attributes displayed.
The PCF value for this is MQG_GAPFILL_ACTIVITY.

FULL

Add zeroed records to produce a set of records that spans the entire time period requested.
The PCF value for this is MQG_GAPFILL_FULL

SUM

Whether the records are summed together, and if so how they are totalled.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_SUM.

Possible Values are:-

NONE

Do not add together any records. The command returns each individual record as reported by IBM MQ.
The PCF value for this is MQG_SUM_NONE.

STREAM

A record will be returned for each MQEV stream. This is the default value.
The PCF value for this is MQG_SUM_STREAM.

TOTAL

Add together all of the records that match the other criteria (e.g. `WHERE` clause) on the display command. If used without the `INTVL` attribute, this will result in only a single record being returned. If used with the `INTVL` attribute, one record will be returned for each interval. The queue name reported on the returned record will reflect the fact that the numbers for many queue names might have been added together.
Using `SUM(TOTAL)` may be very useful when treating multiple queue names as the same name for the purposes of monitoring or charge-back. For example, a set of dynamic queues that all have similar names, could be totalled as if they were a single queue.
The PCF value for this is MQG_SUM_TOTAL.

ZEROVALS

How to handle zero values in the records displayed.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_ZERO_VALUES.

Possible Values are:-

HIDE

Do not show any zero values. This is the default value.
The PCF value for this is MQG_ZEROVALS_HIDE.

SHOW

Show zero values in records.
The PCF value for this is MQG_ZEROVALS_SHOW.

INTVLALGN

How to align the reported intervals of records. This parameter is only used when an `INTVL` is set.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_INTERVAL_ALIGN.

CLOCK

Reported intervals are aligned to the clock. That is, if you request an `INTVL(4hour)`, reported intervals will be at midnight, 4am, 8am, noon and so on. This is the default value.
The PCF value for this is MQG_ALIGN_CLOCK.

MQ

Reported intervals are aligned to the intervals reported by IBM MQ.
The PCF value for this is MQG_ALIGN_MQ.

INTVL

When summing records, the reported records are totalled in intervals of the specified length. See `INTVLALGN` for details of the time alignment of these intervals. This parameter is only valid when you are not using `SUM(NONE)`. The maximum length of this string is MQG_DATE_TIME_LENGTH (30).

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_INTERVAL.

The interval supplied must be a multiple of the interval used in IBM MQ to collect the statistics in the first instance, that is the `STATINT` value. If you are collecting statistics records every 2 hours, you cannot request that MQEV display records summed into intervals of 45 minutes.

For more information about the returned records and their interval start and end times, please see Chapter 10 Returned Interval Times on page 48.

Many different parameter formats are supported to provide an interval. The following are supported:

Values	Meaning
2day (or 2d)	Two days
4hour (or 4hr or 4h)	Four hours
3minute (or 3min or 3m)	Three minutes
1d4h	Values can be combined without spaces

TO

The time up to when records should be returned. The maximum length of this string is MQG_DATE_TIME_LENGTH (30).

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_TO.

For more information about the returned records and their interval start and end times, please see Chapter 10 Returned Interval Times on page 48.

Please see the description of the `FROM` parameter for the allowed values.

If not specified then the value 'now' will be used.

FROM

The time from which records should be returned. The maximum length of this string is MQG_DATE_TIME_LENGTH (30).

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_FROM.

For more information about the returned records and their interval start and end times, please see Chapter 10 Returned Interval Times on page 48.

Many different parameter formats are supported, the time can be specified in either absolute or relative terms.

If not specified then the value '-24hour' will be used.

The following are supported:

Absolute values	**Meaning**
now	The current time
8	Eight AM
8.30	Eight thirty AM
8.31.46	Eight thirty-one and 46 seconds
04-12	12th April (this year) The day and month fields must always be two digits
2018-10-18	18th October 2018 The year field must always be four digits
2018-10-18 8.31.46	An explicit date and time
Relative values	**Relative to 'the other' time parameter** **Note that both times can not be relative**
-2day (or -2d)	Two days before
+1d	One day after
-4hour (or -4hr or -4h)	Four hours before
-3minute (or -3min or -3m)	Three minutes before
-10second (-10 sec or -10s or -10)	Ten seconds before
-1d4h3m6s	Values can be combined without spaces

TZ

The bias, in minutes, of the time zone that the `FROM` and `TO` parameters are specified in, and any provided `INTVL` will also be aligned to this time zone.

When using the PCF interface, this is an MQCFIN parameter with identifier MQC_ATTR_TIMEZONE.

If not specified then the time zone will be assumed to be UTC.

Here are some examples:-

Time zone	TZ Value
Auckland, New Zealand	`TZ(-720)`
UTC	`TZ(0)`
Portland, Oregon, USA	`TZ(480)`

If you are using MO71 or MQSCX, you do not need to manually provide this attribute as those tools automatically include it from known configuration in those tools.

INTVLSTA

This is an integer representation[24] of the date and time of the start of the interval.

When using the PCF interface, this is an MQCFIN64 parameter with identifier MQG_ATTR_START_OF_INTERVAL. This can be both and input and output parameter.

Either use `INTVLSTA` and `INTVLEND`, or use `FROM` and `TO`. You cannot use both. It is expected that users will mainly use `FROM` and `TO`. `INTVLSTA` and `INTVLEND` are designed for programmable interfaces to input values previously returned on earlier commands..

For more information about the returned records and their interval start and end times, please see Chapter 10 Returned Interval Times on page 48.

INTVLEND

This is an integer representation[24] of the date and time of the end of the interval.

When using the PCF interface, this is an MQCFIN64 parameter with identifier MQG_ATTR_END_OF_INTERVAL. This can be both and input and output parameter.

Please refer to the description of INTVLSTA for advice on when to use this parameter.

MAXRECS

The number of source records which should be used to construct the response. This prevents inadvertent consumption of CPU when issuing queries against large amounts of data.

The default value is taken from the **`EV`** object which itself has a default value of 1,000,000.

This value can not be larger than 100,000,000.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_MAX_RECORDS.

MAXRESP

The number of responses to be returned to this DISPLAY command. The default value is 100.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_MAX_RESPONSES.

PREWHERE

Specify a filter condition to only total records that satisfy the selection criterion of the filter condition. For more about the WHERE clause see Chapter 8 Where Clause() on page 42.

24 The number of seconds since 1st January 1970 – also known as Unix time or Epoch time.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_PREWHERE.

WHERE

Specify a filter condition to only display records that satisfy the selection criterion of the filter condition. For more about the WHERE clause see Chapter 8 Where Clause() on page 42.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_WHERE.

STATMQI Attributes

The attribute list can specify any of the following values. When using the PCF interface, this is an MQCFIL parameter with identifier MQG_ATTR_EV_STAT_MQI_ATTRS.

Those highlighted are constructed attributes for your convenience and are not stored in the statistics message. For this reason they are not available in the `MQEVStatMQI` function.

For a description of these fields please read the MQ documentation

MQSC Value	PCF Constant	PCF Type	Description
ALL	MQIACF_ALL	N/A	All attributes
STATMQI	MQG_ATTR_STREAM_NAME	MQCFST	Stream Name
CMDLEVEL	MQIA_COMMAND_LEVEL	MQCFIN	IBM MQ Command Level
CONNS	MQG_ACCST_CONNECTIONS	MQCFIN	Number of success connections in the interval
CONNSFAIL	MQG_ACCST_CONNECTIONS_FAILED	MQCFIN	Number of failed connections in the interval
CONNSMAX	MQG_ACCST_CONNECTIONS_MAX	MQCFIN	Maximum number of concurrent connections
DISCSNORM	MQG_ACCST_DISC_NORMAL	MQCFIN	Number of normal disconnects
DISCSIMPL	MQG_ACCST_DISC_IMPLICIT	MQCFIN	Number of implicit disconnects
DISCSQMGR	MQG_ACCST_DISC_Q_MGR	MQCFIN	Number of Queue Manager disconnects
OPENQ	MQG_ACCST_OPEN_QUEUE	MQCFIN	Open Queue count
OPENNL	MQG_ACCST_OPEN_NAMELIST	MQCFIN	Open Namelist count
OPENPR	MQG_ACCST_OPEN_PROCESS	MQCFIN	Open Process count
OPENQM	MQG_ACCST_OPEN_Q_MGR	MQCFIN	Open QMgr count
OPENTP	MQG_ACCST_OPEN_TOPIC	MQCFIN	Open Topic count
OPENQFL	MQG_ACCST_OPEN_FAIL_QUEUE	MQCFIN	Open Queue fail count
OPENNLFL	MQG_ACCST_OPEN_FAIL_NAMELIST	MQCFIN	Open Namelist fail count
OPENPRFL	MQG_ACCST_OPEN_FAIL_PROCESS	MQCFIN	Open Process fail count
OPENQMFL	MQG_ACCST_OPEN_FAIL_Q_MGR	MQCFIN	Open QMgr fail count
OPENTPFL	MQG_ACCST_OPEN_FAIL_TOPIC	MQCFIN	Open Topic fail count
CLOSEQ	MQG_ACCST_CLOSE_QUEUE	MQCFIN	Close Queue count
CLOSENL	MQG_ACCST_CLOSE_NAMELIST	MQCFIN	Close Namelist count
CLOSEPR	MQG_ACCST_CLOSE_PROCESS	MQCFIN	Close Process count
CLOSEQM	MQG_ACCST_CLOSE_Q_MGR	MQCFIN	Close QMgr count
CLOSETP	MQG_ACCST_CLOSE_TOPIC	MQCFIN	Close Topic count

MQSC Value	PCF Constant	PCF Type	Description
CLOSEQFL	MQG_ACCST_CLOSE_FAIL_QUEUE	MQCFIN	Close Queue fail count
CLOSENLFL	MQG_ACCST_CLOSE_FAIL_NAMELIST	MQCFIN	Close Namelist fail count
CLOSEPRFL	MQG_ACCST_CLOSE_FAIL_PROCESS	MQCFIN	Close Process fail count
CLOSEQMFL	MQG_ACCST_CLOSE_FAIL_Q_MGR	MQCFIN	Close QMgr fail count
CLOSETPFL	MQG_ACCST_CLOSE_FAIL_TOPIC	MQCFIN	Close Topic fail count
INQQ	MQG_ACCST_INQ_QUEUE	MQCFIN	Inquire Queue count
INQNL	MQG_ACCST_INQ_NAMELIST	MQCFIN	Inquire Namelist count
INQPR	MQG_ACCST_INQ_PROCESS	MQCFIN	Inquire Process count
INQQM	MQG_ACCST_INQ_Q_MGR	MQCFIN	Inquire QMgr count
INQQFL	MQG_ACCST_INQ_FAIL_QUEUE	MQCFIN	Inquire Queue fail count
INQNLFL	MQG_ACCST_INQ_FAIL_NAMELIST	MQCFIN	Inquire Namelist fail count
INQPRFL	MQG_ACCST_INQ_FAIL_PROCESS	MQCFIN	Inquire Process fail count
INQQMFL	MQG_ACCST_INQ_FAIL_Q_MGR	MQCFIN	Inquire QMgr fail count
SETQ	MQG_ACCST_SET_QUEUE	MQCFIN	Set Queue count
SETQFL	MQG_ACCST_SET_FAIL_QUEUE	MQCFIN	Set Queue fail count
ALLPUT	MQG_ACCST_ALL_PUTS	MQCFIN	All Puts Constructed from sum of PUT and PUT1
PUT	MQG_ACCST_PUTS	MQCFIN	Puts Constructed from sum of PUTNP and PUTP
PUTNP	MQG_ACCST_PUTS_NP	MQCFIN	Puts (Non-persistent)
PUTP	MQG_ACCST_PUTS_P	MQCFIN	Puts (Persistent)
PUTFAIL	MQG_ACCST_PUTS_FAILED	MQCFIN	Puts Failed
PUT1	MQG_ACCST_PUT1S	MQCFIN	Put1s Constructed from sum of PUT1NP and PUT1P
PUT1NP	MQG_ACCST_PUT1S_NP	MQCFIN	Put1s (Non-persistent)
PUT1P	MQG_ACCST_PUT1S_P	MQCFIN	Put1s (Persistent)
PUT1FAIL	MQG_ACCST_PUT1S_FAILED	MQCFIN	Put1s Failed
PUTBYTE	MQG_ACCST_64_PUT_BYTES	MQCFIN64	Put Bytes Constructed from sum of PUTBYTENP and PUTBYTEP
PUTBYTENP	MQG_ACCST_64_PUT_BYTES_NP	MQCFIN64	Put Bytes (Non-persistent)
PUTBYTEP	MQG_ACCST_64_PUT_BYTES_P	MQCFIN64	Put Bytes(Persistent)
GET	MQG_ACCST_GETS	MQCFIN	Gets Constructed from sum of GETNP and GETP
GETNP	MQG_ACCST_GETS_NP	MQCFIN	Gets (Non-persistent)
GETP	MQG_ACCST_GETS_P	MQCFIN	Gets (Persistent)
GETFAIL	MQG_ACCST_GETS_FAILED	MQCFIN	Gets Failed

MQSC Value	PCF Constant	PCF Type	Description
GETBYTE	MQG_ACCST_64_GET_BYTES	MQCFIN64	Get Bytes Constructed from sum of GETBYTENP and GETBYTEP
GETBYTENP	MQG_ACCST_64_GET_BYTES_NP	MQCFIN64	Get Bytes (Non-persistent)
GETBYTEP	MQG_ACCST_64_GET_BYTES_P	MQCFIN64	Get Bytes(Persistent)
BRS	MQG_ACCST_BROWSES	MQCFIN	Browses Constructed from sum of BRSNP and BRSP
BRSNP	MQG_ACCST_BROWSES_NP	MQCFIN	Browses (Non-persistent)
BRSP	MQG_ACCST_BROWSES_P	MQCFIN	Browses (Persistent)
BRSFAIL	MQG_ACCST_BROWSES_FAILED	MQCFIN	Browses Failed
BRSBYTE	MQG_ACCST_64_BROWSE_BYTES	MQCFIN64	Browses Bytes Constructed from sum of BRSBYTENP and BRSBYTEP
BRSBYTENP	MQG_ACCST_64_BROWSE_BYTES_NP	MQCFIN64	Browses Bytes (Non-persistent)
BRSBYTEP	MQG_ACCST_64_BROWSE_BYTES_P	MQCFIN64	Browses Bytes(Persistent)
COMMIT	MQG_ACCST_COMMIT	MQCFIN	Commit count
COMMITFL	MQG_ACCST_COMMIT_FAIL	MQCFIN	Commit fail count
BACKOUT	MQG_ACCST_BACKOUT	MQCFIN	Backout count
EXPIRED	MQG_ACCST_MSGS_EXPIRED	MQCFIN	Number of messages discarded due to expiry
PURGED	MQG_ACCST_MSGS_PURGED	MQCFIN	Number of times the queue has been cleared
SUBDURCR	MQG_ACCST_SUB_DUR_CREATED	MQCFIN	Number of durable subscriptions created
SUBDURAL	MQG_ACCST_SUB_DUR_ALTERED	MQCFIN	Number of durable subscriptions altered
SUBDURRS	MQG_ACCST_SUB_DUR_RESUMED	MQCFIN	Number of durable subscriptions resumed
SUBNDURCR	MQG_ACCST_SUB_NONDUR_CREATED	MQCFIN	Number of non-durable subscriptions created
SUBNDURAL	MQG_ACCST_SUB_NONDUR_ALTERED	MQCFIN	Number of non-durable subscriptions altered
SUBNDURRS	MQG_ACCST_SUB_NONDUR_RESUMED	MQCFIN	Number of non-durable subscriptions resumed
SUBFL	MQG_ACCST_SUB_FAIL	MQCFIN	Subscription fail count
UNSUBDCLS	MQG_ACCST_UNSUB_DUR_CLOSED	MQCFIN	Number of durable unsubscribe closes
UNSUBDREM	MQG_ACCST_UNSUB_DUR_REMOVED	MQCFIN	Number of durable unsubscribe removes
UNSUBNCLS	MQG_ACCST_UNSUB_NONDUR_CLOSED	MQCFIN	Number of non-durable unsubscribe closes
UNSUBNREM	MQG_ACCST_UNSUB_NONDUR_REMOVED	MQCFIN	Number of non-durable unsubscribe removes
UNSUBFL	MQG_ACCST_UNSUB_FAIL	MQCFIN	Unsubscribe fail count
SUBRQ	MQG_ACCST_SUBRQ	MQCFIN	Number of successful MQSUBRQ requests
SUBRQFL	MQG_ACCST_SUBRQ_FAIL	MQCFIN	Number of unsuccessful MQSUBRQ requests
CBS	MQG_ACCST_CBS	MQCFIN	MQCB calls Constructed from sum of CBCREATE, CBREMOVE, CBRESUME, and CBSUSPEND.

MQSC Value	PCF Constant	PCF Type	Description
CBCREATE	MQG_ACCST_CBS_CREATED	MQCFIN	MQCB calls using MQOP_REGISTER
CBREMOVE	MQG_ACCST_CBS_REMOVED	MQCFIN	MQCB calls using MQOP_DEREGISTER
CBRESUME	MQG_ACCST_CBS_RESUMED	MQCFIN	MQCB calls using MQOP_RESUME
CBSUSPEND	MQG_ACCST_CBS_SUSPENDED	MQCFIN	MQCB calls using MQOP_SUSPEND
CBFAIL	MQG_ACCST_CBS_FAILED	MQCFIN	MQCB calls failed
CTLSTA	MQG_ACCST_CTL_STARTED	MQCFIN	MQCTL calls using MQOP_START*
CTLSTP	MQG_ACCST_CTL_STOPPED	MQCFIN	MQCTL calls using MQOP_STOP
CTLRES	MQG_ACCST_CTL_RESUMED	MQCFIN	MQCTL calls using MQOP_RESUME
CTLSUS	MQG_ACCST_CTL_SUSPENDED	MQCFIN	MQCTL calls using MQOP_SUSPEND
CTLFL	MQG_ACCST_CTL_FAIL	MQCFIN	MQCTL calls failed
STAT	MQG_ACCST_STAT	MQCFIN	MQSTAT count
STATFL	MQG_ACCST_STAT_FAIL	MQCFIN	MQSTAT fail count
SUBDHWALL	MQG_ACCST_SUB_DUR_HIGHWATER_ALL	MQCFIN	High-water mark on number of durable subscriptions
SUBDHWAPI	MQG_ACCST_SUB_DUR_HIGHWATER_API	MQCFIN	High-water mark on number of durable API subscriptions
SUBDHWADM	MQG_ACCST_SUB_DUR_HIGHWATER_ADMIN	MQCFIN	High-water mark on number of durable Admin subscriptions
SUBDHWPRX	MQG_ACCST_SUB_DUR_HIGHWATER_PROXY	MQCFIN	High-water mark on number of durable Proxy subscriptions
SUBDLWALL	MQG_ACCST_SUB_DUR_LOWWATER_ALL	MQCFIN	Low-water mark on number of durable subscriptions
SUBDLWAPI	MQG_ACCST_SUB_DUR_LOWWATER_API	MQCFIN	Low-water mark on number of durable API subscriptions
SUBDLWADM	MQG_ACCST_SUB_DUR_LOWWATER_ADMIN	MQCFIN	Low-water mark on number of durable Admin subscriptions
SUBDLWPRX	MQG_ACCST_SUB_DUR_LOWWATER_PROXY	MQCFIN	Low-water mark on number of durable Proxy subscriptions
SUBNHWALL	MQG_ACCST_SUB_NDUR_HIGHWATER_ALL	MQCFIN	High-water mark on number of non-durable subscriptions
SUBNHWAPI	MQG_ACCST_SUB_NDUR_HIGHWATER_API	MQCFIN	High-water mark on number of non-durable API subscriptions
SUBNHWADM	MQG_ACCST_SUB_NDUR_HIGHWATER_ADMIN	MQCFIN	High-water mark on number of non-durable Admin subscriptions
SUBNHWPRX	MQG_ACCST_SUB_NDUR_HIGHWATER_PROXY	MQCFIN	High-water mark on number of non-durable Proxy subscriptions
SUBNLWALL	MQG_ACCST_SUB_NDUR_LOWWATER_ALL	MQCFIN	Low-water mark on number of non-durable subscriptions
SUBNLWAPI	MQG_ACCST_SUB_NDUR_LOWWATER_API	MQCFIN	Low-water mark on number of non-durable API

MQSC Value	PCF Constant	PCF Type	Description
			subscriptions
SUBNLWADM	MQG_ACCST_SUB_NDUR_LOWWATER_ADMIN	MQCFIN	Low-water mark on number of non-durable Admin subscriptions
SUBNLWPRX	MQG_ACCST_SUB_NDUR_LOWWATER_PROXY	MQCFIN	Low-water mark on number of non-durable Proxy subscriptions
PUTTOP	MQG_ACCST_PUT_TOPIC	MQCFIN	Number of messages put to a topic Constructed from sum of PUTTOPNP and PUTTOPP
PUTTOPNP	MQG_ACCST_PUT_TOPIC_NP	MQCFIN	Number of non-persistent messages put to a topic
PUTTOPP	MQG_ACCST_PUT_TOPIC_P	MQCFIN	Number of persistent messages put to a topic
PUTTOPFL	MQG_ACCST_PUT_TOPIC_FAIL	MQCFIN	Number of failed puts to a topic
PUT1TOP	MQG_ACCST_PUT1_TOPIC	MQCFIN	Number of messages put1 to a topic Constructed from sum of PUT1TOPNP and PUT1TOPP
PUT1TOPNP	MQG_ACCST_PUT1_TOPIC_NP	MQCFIN	Number of non-persistent messages put1 to a topic
PUT1TOPP	MQG_ACCST_PUT1_TOPIC_P	MQCFIN	Number of persistent messages put1 to a topic
PUT1TOPFL	MQG_ACCST_PUT1_TOPIC_FAIL	MQCFIN	Number of failed put1s to a topic
PUTTOPBYTE	MQG_ACCST_64_PUT_TOPIC_BYTES	MQCFIN64	Number of bytes put to a topic Constructed from sum of PUTTOPBYTENP and PUTTOPBYTEP
PUTTOPBYTENP	MQG_ACCST_64_PUT_TOPIC_BYTES_NP	MQCFIN64	Number of non-persistent bytes put to a topic
PUTTOPBYTEP	MQG_ACCST_64_PUT_TOPIC_BYTES_P	MQCFIN64	Number of persistent bytes put to a topic
PUBMSG	MQG_ACCST_PUBLISH_MSG	MQCFIN	Number of messages delivered to subscribers Constructed from sum of PUBMSGNP and PUTMSGP
PUBMSGNP	MQG_ACCST_PUBLISH_MSG_NP	MQCFIN	Number of non-persistent messages delivered to subscribers
PUBMSGP	MQG_ACCST_PUBLISH_MSG_P	MQCFIN	Number of persistent messages delivered to subscribers
PUBMSGBYTE	MQG_ACCST_64_PUBLISH_MSG_BYTES	MQCFIN64	Number of bytes delivered to subscribers Constructed from sum of PUBMSGBYTENP and PUTMSGBYTEP
PUBMSGBYTENP	MQG_ACCST_64_PUBLISH_MSG_BYTES_NP	MQCFIN64	Number of non-persistent bytes delivered to subscribers
PUBMSGBYTEP	MQG_ACCST_64_PUBLISH_MSG_BYTES_P	MQCFIN64	Number of persistent bytes delivered to subscribers
RECORDS	MQG_ATTR_RECORDS	MQCFIN	Number of records totalled together

11.24 DISPLAY STATQ

Use the MQSC command **`DISPLAY STATQ`** (or it's equivalent PCF command **`MQG_CMD_DISPLAY_STAT_Q`**) to display the queue statistics records processed and stored by **MQEV**. For a description of the statistics fields provided by MQ please read the MQ documentation

11.24.1 Syntax diagram for DISPLAY STATQ

```
►►─ DISPLAY STATQ( wildcarded-queue-name ) ─┬──────────────────────────┬──────────►
                                            └─ EVQMGR( qmgr-name ) ─┘

    ┌─ GAPFILL( NO ) ──────────────┐     ┌─ SUM( QUEUE ) ──────────────┐
►───┤                              ├─────┤                             ├──────────►
    └─ GAPFILL( ─┬─ INNER ────┬─ ) ─┘     ├─ SUM( ─┬─ NONE ───────┬─ ) ─┤
                 ├─ ACTIVITY ─┤           │       └─ TOTAL ──────┘     │
                 └─ FULL ─────┘           │       ┌─,──────────────┐   │
                                          │       ↓                │   │
                                          └─ SUM( ─┬─ QUEUE ──────┬─ ) ─┘
                                                   └─ STREAM ─────┘
    ┌─ ZEROVALS( HIDE ) ─┐
►───┤                    ├──────────────────────────────────────────────────────►
    └─ ZEROVALS( SHOW ) ─┘

    ┌─ FROM( '-24h' ) ──────────┐   ┌─ TO( 'now' ) ──────────┐   ┌─ TZ( 0 ) ──────────┐
►───┤                           ├───┤                        ├───┤                    ├──────►
    └─ FROM( date-and-time ) ─┘   └─ TO( date-and-time ) ─┘   └─ TZ( integer ) ─┘

    ┌─ INTVLALGN( CLOCK ) ─┐   ┌─ INTVL( ' ' ) ──────────┐
►───┤                      ├───┤                         ├──────────────────────────►
    └─ INTVLALGN( MQ ) ────┘   └─ INTVL( time-period ) ─┘

►───┬──────────────────────────────────────────┬──────────────────────────────────►
    └─ INTVLSTA( date-and-time-seconds ) ─┘   └─ INTVLEND( date-and-time-seconds ) ─┘

►───┬──────────────────────────────────────────┬──────────────────────────────────►
    └─ WHERE( FilterCondition ) ─┘   └─ PREWHERE( FilterCondition ) ─┘

    ┌─ MAXRECS( ev-value ) ─┐   ┌─ MAXRESP( 100 ) ─────┐
►───┤                       ├───┤                      ├──┬──────────────┬──┬─────┬──►◄
    └─ MAXRECS( integer ) ─┘   └─ MAXRESP( integer ) ─┘  └─┤ attributes ├─┘  └─ ALL ─┘
```

11.24.2 Parameter descriptions for DISPLAY STATQ

(wildcarded-queue-name)

The queue name for which statistics records are to be displayed. This can be a wildcarded string.

When using the PCF interface, this is an MQCFST parameter with identifier MQCA_Q_NAME.

EVQMGR

The queue manager associated with the records to be displayed.

This parameter is used to specify which queue manager you wish to see data from. If it is not specified, the output shows records from the queue manager you are connected to.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_Q_MGR_NAME.

GAPFILL

Whether to add zeroed records where none exist to produce a set of records that will graph appropriately.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_GAP_FILL.

Possible Values are:-

NO

Do not fill in the gaps with zeroed records.
The PCF value for this is MQG_GAPFILL_NO.

INNER

Produce zeroed records for inner gaps in the set of records.
The PCF value for this is MQG_GAPFILL_INNER.

ACTIVITY

Add zeroed records to produce a set of records that spans the time period where there is any activity on this queue, not just the activity shown by the attributes displayed.
The PCF value for this is MQG_GAPFILL_ACTIVITY.

FULL

Add zeroed records to produce a set of records that spans the entire time period requested.
The PCF value for this is MQG_GAPFILL_FULL

SUM

Whether the records are summed together, and if so how they are totalled. Multiple values can be provided in a comma-separated list, except where indicated that a value cannot be combined with any others.

When multiple values are provided, a record will be returned for each unique combination, e.g. `SUM(QUEUE, STREAM)` will return one record for each unique combination of queue name and stream name.

When using the PCF interface, this can be an MQCFIN parameter (should you only need to supply one value) or an MQCFIL parameter (should you need to supply multiple values) with identifier MQG_ATTR_SUM.

When any of the `SUM` values apart from `NONE` and `TOTAL` are used without the `INTVL` attribute, this will result in a single record per unique key combination being returned. If used with the `INTVL` attribute, one record per application will be returned for each interval.

For more information about the returned records and their interval start and end times, please see Chapter 10 Returned Interval Times on page 48.

Possible Values are:-

NONE

Do not add together any records. The command returns each individual record as reported by IBM MQ. This value cannot be combined with any others.
The PCF value for this is MQG_SUM_NONE.

QUEUE

Add together records that have the same queue name. If used without the `INTVL` attribute, this will result in a single record per queue being returned. If used with the `INTVL` attribute, one record per queue will be returned for each interval. This is the default value.
The PCF value for this is MQG_SUM_QUEUE.

STREAM

A record will be returned for each MQEV stream
The PCF value for this is MQG_SUM_STREAM.

TOTAL

Add together all of the records that match the other criteria (e.g. `WHERE` clause) on the display command. If used without the `INTVL` attribute, this will result in only a single record being returned. If used with the `INTVL` attribute, one record will be returned for each interval. The queue name reported on the returned record will reflect the fact that the numbers for many queue names might have been added together.
Using `SUM(TOTAL)` may be very useful when treating multiple queue names as the same name for the purposes of monitoring or charge-back. For example, a set of dynamic queues that all have similar names, could be totalled as if they were a single queue.
This value cannot be combined with any others.
The PCF value for this is MQG_SUM_TOTAL.

ZEROVALS

How to handle zero values in the records displayed.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_ZERO_VALUES.

Possible Values are:-

HIDE

Do not show any zero values. This is the default value.
The PCF value for this is MQG_ZEROVALS_HIDE.

SHOW

Show zero values in records.
The PCF value for this is MQG_ZEROVALS_SHOW.

INTVLALGN

How to align the reported intervals of records. This parameter is only used when an `INTVL` is set.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_INTERVAL_ALIGN.

CLOCK

Reported intervals are aligned to the clock. That is, if you request an `INTVL(4hour)`, reported intervals will be at midnight, 4am, 8am, noon and so on. This is the default value.
The PCF value for this is MQG_ALIGN_CLOCK.

MQ

Reported intervals are aligned to the intervals reported by IBM MQ.
The PCF value for this is MQG_ALIGN_MQ.

INTVL

When summing records, the reported records are totalled in intervals of the specified length. See `INTVLALGN` for details of the time alignment of these intervals. This parameter is only valid when you are not using `SUM(NONE)`. The maximum length of this string is MQG_DATE_TIME_LENGTH (30).

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_INTERVAL.

The interval supplied must be a multiple of the interval used in IBM MQ to collect the statistics in the first instance, that is the `STATINT` value. If you are collecting statistics records every 2 hours, you cannot request that MQEV display records summed into intervals of 45 minutes.

For more information about the returned records and their interval start and end times, please see Chapter 10 Returned Interval Times on page 48.

Many different parameter formats are supported to provide an interval. The following are supported:

Values	Meaning
2day (or 2d)	Two days
4hour (or 4hr or 4h)	Four hours
3minute (or 3min or 3m)	Three minutes
1d4h	Values can be combined without spaces

FROM

The time from which records should be returned. The maximum length of this string is MQG_DATE_TIME_LENGTH (30).

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_FROM.

For more information about the returned records and their interval start and end times, please see Chapter 10 Returned Interval Times on page 48.

Many different parameter formats are supported, the time can be specified in either absolute or relative terms. The following are supported:

Absolute values	Meaning
now	The current time
8	Eight AM
8.30	Eight thirty AM
8.31.46	Eight thirty-one and 46 seconds
04-12	12th April (this year) The day and month fields must always be two digits
2018-10-18	18th October 2018 The year field must always be four digits
2018-10-18 8.31.46	An explicit date and time
Relative values	**Relative to 'the other' time parameter** **Note that both times can not be relative**
-2day (or -2d)	Two days before
+1d	One day after
-4hour (or -4hr or -4h)	Four hours before
-3minute (or -3min or -3m)	Three minutes before
-10second (-10 sec or -10s or -10)	Ten seconds before
-1d4h3m6s	Values can be combined without spaces

If not specified then the value '-24hour' will be used.

TO

The time up to when records should be returned. The maximum length of this string is MQG_DATE_TIME_LENGTH (30).

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_TO.

For more information about the returned records and their interval start and end times, please see Chapter 10 Returned Interval Times on page 48.

Please see the description of the `FROM` parameter for the allowed values.

If not specified then the value 'now' will be used.

TZ

The bias, in minutes, of the time zone that the `FROM` and `TO` parameters are specified in, and any provided `INTVL` will also be aligned to this time zone.

When using the PCF interface, this is an MQCFIN parameter with identifier MQC_ATTR_TIMEZONE.

If not specified then the time zone will be assumed to be UTC.

Here are some examples:-

Time zone	TZ Value
Auckland, New Zealand	`TZ(-720)`
UTC	`TZ(0)`
Portland, Oregon, USA	`TZ(480)`

If you are using MO71 or MQSCX, you do not need to manually provide this attribute as those tools automatically include it from known configuration in those tools.

INTVLSTA

This is an integer representation[25] of the date and time of the start of the interval.

When using the PCF interface, this is an MQCFIN64 parameter with identifier MQG_ATTR_START_OF_INTERVAL. This can be both and input and output parameter.

Either use `INTVLSTA` and `INTVLEND`, or use `FROM` and `TO`. You cannot use both. It is expected that users will mainly use `FROM` and `TO`. `INTVLSTA` and `INTVLEND` are designed for programmable interfaces to input values previously returned on earlier commands..

For more information about the returned records and their interval start and end times, please see Chapter 10 Returned Interval Times on page 48.

INTVLEND

This is an integer representation[25] of the date and time of the end of the interval.

When using the PCF interface, this is an MQCFIN64 parameter with identifier MQG_ATTR_END_OF_INTERVAL. This can be both and input and output parameter.

Please refer to the description of INTVLSTA for advice on when to use this parameter.

MAXRECS

The number of source records which should be used to construct the response. This prevents inadvertent consumption of CPU when issuing queries against large amounts of data.

The default value is taken from the **`EV`** object which itself has a default value of 1,000,000.

This value can not be larger than 100,000,000.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_MAX_RECORDS.

MAXRESP

The number of responses to be returned to this DISPLAY command. The default value is 100.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_MAX_RESPONSES.

25 The number of seconds since 1st January 1970 – also known as Unix time or Epoch time.

PREWHERE

Specify a filter condition to only total records that satisfy the selection criterion of the filter condition. For more about the WHERE clause see Chapter 8 Where Clause() on page 42.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_PREWHERE.

WHERE

Specify a filter condition to only display records that satisfy the selection criterion of the filter condition. For more about the WHERE clause see Chapter 8 Where Clause() on page 42.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_WHERE.

STATQ Attributes

The attribute list can specify any of the following values. When using the PCF interface, this is an MQCFIL parameter with identifier MQG_ATTR_EV_STAT_Q_ATTRS.

Those highlighted are constructed attributes for your convenience and are not stored in the statistics message. For this reason they are not available in the `MQEVStatQ` function.

For a description of these fields please read the MQ documentation

MQSC Value	PCF Constant	PCF Type	Description
ALL	MQIACF_ALL	N/A	All attributes
INTVLS	MQG_ATTR_INTERVALS	MQCFIN	Intervals
CMDLEVEL	MQIA_COMMAND_LEVEL	MQCFIN	IBM MQ Command Level
ALLPUT	MQG_ACCST_ALL_PUTS	MQCFIN	All Puts Constructed from sum of PUT and PUT1
PUT	MQG_ACCST_PUTS	MQCFIN	Puts Constructed from sum of PUTNP and PUTP
PUTNP	MQG_ACCST_PUTS_NP	MQCFIN	Puts (Non-persistent)
PUTP	MQG_ACCST_PUTS_P	MQCFIN	Puts (Persistent)
PUTBYTE	MQG_ACCST_64_PUT_BYTES	MQCFIN64	Put Bytes Constructed from sum of PUTBYTENP and PUTBYTEP
PUTBYTENP	MQG_ACCST_64_PUT_BYTES_NP	MQCFIN64	Put Bytes (Non-persistent)
PUTBYTEP	MQG_ACCST_64_PUT_BYTES_P	MQCFIN64	Put Bytes(Persistent)
PUTFAIL	MQG_ACCST_PUTS_FAILED	MQCFIN	Puts Failed
PUT1	MQG_ACCST_PUT1S	MQCFIN	Put1s Constructed from sum of PUT1NP and PUT1P
PUT1NP	MQG_ACCST_PUT1S_NP	MQCFIN	Put1s (Non-persistent)
PUT1P	MQG_ACCST_PUT1S_P	MQCFIN	Put1s (Persistent)
PUT1FAIL	MQG_ACCST_PUT1S_FAILED	MQCFIN	Put1s Failed
GET	MQG_ACCST_GETS	MQCFIN	Gets Constructed from sum of GETNP and GETP
GETNP	MQG_ACCST_GETS_NP	MQCFIN	Gets (Non-persistent)
GETP	MQG_ACCST_GETS_P	MQCFIN	Gets (Persistent)
GETFAIL	MQG_ACCST_GETS_FAILED	MQCFIN	Gets Failed
GETBYTE	MQG_ACCST_64_GET_BYTES	MQCFIN64	Get Bytes Constructed from sum of GETBYTENP and GETBYTEP
GETBYTENP	MQG_ACCST_64_GET_BYTES_NP	MQCFIN64	Get Bytes (Non-persistent)

MQSC Value	PCF Constant	PCF Type	Description
GETBYTEP	MQG_ACCST_64_GET_BYTES_P	MQCFIN64	Get Bytes(Persistent)
BRS	MQG_ACCST_BROWSES	MQCFIN	Browses Constructed from sum of BRSNP and BRSP
BRSNP	MQG_ACCST_BROWSES_NP	MQCFIN	Browses (Non-persistent)
BRSP	MQG_ACCST_BROWSES_P	MQCFIN	Browses (Persistent)
BRSFAIL	MQG_ACCST_BROWSES_FAILED	MQCFIN	Browses Failed
BRSBYTE	MQG_ACCST_64_BROWSE_BYTES	MQCFIN64	Browses Bytes Constructed from sum of BRSBYTENP and BRSBYTEP
BRSBYTENP	MQG_ACCST_64_BROWSE_BYTES_NP	MQCFIN64	Browses Bytes (Non-persistent)
BRSBYTEP	MQG_ACCST_64_BROWSE_BYTES_P	MQCFIN64	Browses Bytes(Persistent)
NOTQ	MQG_ACCST_MSGS_NOT_QUEUED	MQCFIN	Messages not queued (Put to waiting getter)
EXPIRED	MQG_ACCST_MSGS_EXPIRED	MQCFIN	Messages expired
PURGED	MQG_ACCST_MSGS_PURGED	MQCFIN	Messages purged (CLEAR QL)
CBS	MQG_ACCST_CBS	MQCFIN	MQCB calls Constructed from sum of CBCREATE, CBREMOVE, CBRESUME and CBSUSPEND.
CBCREATE	MQG_ACCST_CBS_CREATED	MQCFIN	MQCB calls using MQOP_REGISTER
CBREMOVE	MQG_ACCST_CBS_REMOVED	MQCFIN	MQCB calls using MQOP_DEREGISTER
CBRESUME	MQG_ACCST_CBS_RESUMED	MQCFIN	MQCB calls using MQOP_RESUME
CBSUSPEND	MQG_ACCST_CBS_SUSPENDED	MQCFIN	MQCB calls using MQOP_SUSPEND
CBSFAIL	MQG_ACCST_CBS_FAILED	MQCFIN	MQCB calls failed
AVGQNP	MQG_ACCST_64_AVG_Q_TIME_NP	MQCFIN64	Average QTIME (Non-persistent)
AVGQP	MQG_ACCST_64_AVG_Q_TIME_P	MQCFIN64	Average QTIME (Persistent)
MINDEPTH	MQG_ACCST_MIN_DEPTH	MQCFIN	Minimum depth
MAXDEPTH	MQG_ACCST_MAX_DEPTH	MQCFIN	Maximum depth
RECORDS	MQG_ATTR_RECORDS	MQCFIN	Number of records totalled together

11.25 PURGE EVSTRMST

Use the MQSC command **`PURGE EVSTRMST`** (or it's equivalent PCF command **`MQG_CMD_PURGE_EV_STREAM_STATUS`**) to purge the run-time status of streams in use on various queue managers.

Streams are used to store the items (events, accounting and statistics records) processed by **MQEV**.

11.25.1 Syntax diagram for PURGE EVSTRMST

```
►►─ PURGE EVSTRMST( wildcarded-stream-name ) ─ EVQMGR( wildcarded-qmgr-name ) ─►

►── TYPE( ─┬─ ALL ───────┬─ ) ──────────────────────────────────────────────►◄
          ├─ EVENTS ────┤
          ├─ STATQ ─────┤
          ├─ STATCHL ───┤
          ├─ STATMQI ───┤
          ├─ ACCTQ ─────┤
          └─ ACCTMQI ───┘
```

11.25.2 Parameter descriptions for PURGE EVSTRMST

(wildcarded-stream-name)

The stream name to be purged. This can be a wildcarded string.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_STREAM_NAME.

EVQMGR

The queue manager name related to the stream to be purged. This can be a wildcarded string.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_Q_MGR_NAME.

TYPE

The type of streams to purge. This is a mandatory parameter. If you wish to purge all streams, you can use `TYPE(ALL)` along with a wildcarded string for the stream name.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_STREAM_TYPE.

Possible values are:-

ALL

All stream types are to be purged.
The PCF value for this is MQG_STREAM_TYPE_ALL.

EVENTS

Purge only streams that contain event data.
The PCF value for this is MQG_STREAM_TYPE_EVENTS.

STATQ

Purge only streams that contain queue statistics data.
The PCF value for this is MQG_STREAM_TYPE_STAT_Q.

STATCHL

Purge only streams that contain channel statistics data.
The PCF value for this is MQG_STREAM_TYPE_STAT_CHL.

STATMQI

Purge only streams that contain MQI statistics data.
The PCF value for this is MQG_STREAM_TYPE_STAT_MQI.

ACCTQ

Purge only streams that contain queue accounting data.
The PCF value for this is MQG_STREAM_TYPE_ACCT_Q.

ACCTMQI

Purge only streams that contain MQI accounting data.
The PCF value for this is MQG_STREAM_TYPE_ACCT_MQI.

11.26 REMOVE EVALERT

Use the MQSC command **REMOVE EVALERT** (or it's equivalent PCF command **MQG_CMD_REMOVE_EV_ALERT**) to remove an alert from the system. Alerts can be used as reminders or notifications. Learn more about alerts in Chapter 12 Alerts on page 153.

A log file entry will be written by this command showing the alert ID, and details, which was removed.

11.26.1 Syntax diagram for REMOVE EVALERT

```
►►─ REMOVE EVALERT ─┬─ ( alert-id ) ──────────────────────────────────►◄
                    └─┤ alert-matching-block ├─┘

Alert Matching Block

►►─ TEXT( string ) ─── CATEGORY( wildcarded-string ) ──────────────────►

►── EVOBJNAME( wildcarded-object-name ) ── EVQMGR( wildcarded-qmgr-name ) ──►

►── EVOBJTYPE( ─┬─ AUTHINFO ─┬─ )──────────────────────────────────────►◄
               ├─ AUTHREC ──┤
               ├─ CFSTRUCT ─┤
               ├─ CHANNEL ──┤
               ├─ CHLAUTH ──┤
               ├─ CLNTCONN ─┤
               ├─ COMMINFO ─┤
               ├─ LISTENER ─┤
               ├─ NAMELIST ─┤
               ├─ NONE ─────┤
               ├─ PROCESS ──┤
               ├─ QUEUE ────┤
               ├─ QMGR ─────┤
               ├─ RQMNAME ──┤
               ├─ SERVICE ──┤
               ├─ SUB ──────┤
               ├─ STGCLASS ─┤
               ├─ TOPIC ────┤
               └─ TOPICSTR ─┘
```

11.26.2 Parameter descriptions for REMOVE EVALERT

(alert-id)

The unique ID of the alert. If specified, this must be an individual alert ID.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_ALERT_ID.

CATEGORY

The category of the alert. This can be a wildcarded string.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_ALERT_CATEGORY.

EVOBJNAME

The object name that this alert refers to. This can be a wildcarded string.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_OBJECT.

EVOBJTYPE

The object type that this alert refers to.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_OBJECT_TYPE.

Possible Values are:-

MQSC value	Meaning	PCF constant
AUTHINFO	Authentication information object	MQOT_AUTH_INFO
AUTHREC	Authorization records	MQOT_AUTH_REC
CFSTRUCT	CF Structure	MQOT_CF_STRUC
CHANNEL	Channel	MQOT_CHANNEL
CHLAUTH	Channel Authentication records	MQOT_CHLAUTH
CLNTCONN	Client connection channel	MQOT_CLNTCONN_CHANNEL
COMMINFO	Communication information object	MQOT_COMM_INFO
LISTENER	Listener	MQOT_LISTENER
NAMELIST	Namelist	MQOT_NAMELIST
NONE	None.	MQOT_NONE
PROCESS	Process	MQOT_PROCESS
QUEUE	Queue	MQOT_Q
QMGR	Queue manager	MQOT_Q_MGR
RQMNAME	Remote queue manager	MQOT_REMOTE_Q_MGR_NAME
SERVICE	Service object	MQOT_SERVICE
STGCLASS	Storage Class	MQOT_STORAGE_CLASS
SUB	Subscription	MQG_OT_SUB
TOPIC	Topic	MQOT_TOPIC
TOPICSTR	Topic String	MQG_OT_TOPICSTR

EVQMGR

The queue manager to which this alert is associated. This can be a wildcarded string.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_ALERT_Q_MGR.

TEXT

The text of the alert. This must match exactly.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_ALERT_TEXT.

11.27 REMOVE EVQ

Use the MQSC command **`REMOVE EVQ`** (or it's equivalent PCF command **`MQG_CMD_REMOVE_EV_Q`**) to remove a queue from being processed by the **MQEV** event processor.

A log file entry will be written by this command showing the queue name that was removed.

11.27.1 Syntax diagram for REMOVE EVQ

```
►►─ REMOVE EVQ( queue-name ) ──────►◄
```

11.27.2 Parameter descriptions for REMOVE EVQ

(queue-name)

The name of the IBM MQ event queue being removed from **MQEV** processing.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_EVENT_Q_NAME.

11.28 REMOVE EVQMGR

Use the MQSC command **REMOVE EVQMGR** (or it's equivalent PCF command **MQG_CMD_REMOVE_EV_Q_MGR**) to remove details about a queue manager which events had previously been processed for.

A log file entry will be written by this command showing that the queue manager has been removed, and also entries to show the purge as per the PURGE command if **PURGE(YES)** is specified.

11.28.1 Syntax diagram for DISPLAY EVQMGR

```
                                   ┌─ PURGE( NO ) ─┐
►►─ REMOVE EVQMGR( qmgr-name ) ────┼───────────────┼──────────►◄
                                   └─ PURGE( YES ) ─┘
```

11.28.2 Parameter descriptions for REMOVE EVQMGR

(qmgr-name)

The name of the queue manager whose data is being removed from MQEV.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_Q_MGR_NAME.

PURGE

Whether to purge data about this queue manager from MQEV.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_Q_MGR_PURGE.

Possible values are:-

YES

Purge the data at the same time as removing the queue manager.
Using **PURGE(YES)** is equivalent to (and saves you from doing):-
PURGE EVSTRMST(*) EVQMGR(qmgr-name) TYPE(ALL)
The PCF value for this is MQG_PURGE_YES.

NO

Don't purge the data when removing the queue manager..
The PCF value for this is MQG_PURGE_NO.

11.29 RENAME EVSTREAM

Use the MQSC command **`RENAME EVSTREAM`** (or it's equivalent PCF command **`MQG_CMD_RENAME_EV_STREAM`**) to change the name of a stream.

Streams are used to store the records (events, accounting and statistics) processed by **MQEV**.

A log file entry will be written by this command showing the stream that was changed, and the new name.

11.29.1 Syntax diagram for RENAME EVSTREAM

```
►►─ RENAME EVSTREAM( stream-name) ── NEWNAME( stream-name ) ──────►

►── ──────────────────────────────────────────────────────────────►◄
      └─── TYPE( ─┬─ EVENTS ──┬─ ) ──┘
                  ├─ STATQ ───┤
                  ├─ STATCHL ─┤
                  ├─ STATMQI ─┤
                  ├─ ACCTQ ───┤
                  └─ ACCTMQI ─┘
```

11.29.2 Parameter descriptions for RENAME EVSTREAM

(stream-name)

The name of the stream to be renamed. The maximum length of this string is MQG_STREAM_NAME_LENGTH.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_STREAM_NAME.

NEWNAME

The new name of the stream. The maximum length of this string is MQG_STREAM_NAME_LENGTH.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_NEW_STREAM_NAME.

TYPE

The type of data that is stored on this stream. This attribute is optional, and only required if the stream name is not a unique reference to the stream object being altered.

When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_STREAM_TYPE.

Possible values are:-

EVENTS

This stream contains event data.
The PCF value for this is MQG_STREAM_TYPE_EVENTS.

STATQ

This stream contains queue statistics data.
The PCF value for this is MQG_STREAM_TYPE_STAT_Q.

STATCHL

This stream contains channel statistics data.
The PCF value for this is MQG_STREAM_TYPE_STAT_CHL.

STATMQI

This stream contains MQI statistics data.
The PCF value for this is MQG_STREAM_TYPE_STAT_MQI.

ACCTQ

This stream contains queue accounting data.
The PCF value for this is MQG_STREAM_TYPE_ACCT_Q.

ACCTMQI

This stream contains MQI accounting data.
The PCF value for this is MQG_STREAM_TYPE_ACCT_MQI.

11.30 RESET EV

Use the MQSC command **`RESET EV`** (or it's equivalent PCF command **`MQG_CMD_RESET_EV`**) to reset the counts shown on the **MQEV** event processor.

A log file entry will be written by this command showing that a reset happened and recording the counts prior to the reset.

11.30.1 Syntax diagram for RESET EV

```
►►─ RESET EV ─── TYPE( COUNTS ) ──────►◄
```

11.30.2 Parameter descriptions for RESET EV

TYPE

The type of RESET being done. When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_RESET_TYPE.

COUNTS

All the event counts will be reset to zero. The `RESETTI` attribute will be updated to record when the reset command was issued. The counts prior to the reset are written to the MQEV log file. This value has the PCF value of MQG_RESET_TYPE_COUNTS.

11.31 RESUME EVQ

Use the MQSC command **`RESUME EVQ`** (or it's equivalent PCF command **`MQG_CMD_RESUME_EV_Q`**) to resume an event queue to be processed by the **MQEV** event processor.

A log file entry will be written by this command showing the queue name that was resumed.

11.31.1 Syntax diagram for RESUME EVQ

```
►►─ RESUME EVQ( queue-name ) ──────►◄
```

11.31.2 Parameter descriptions for RESUME EVQ

(queue-name)

The name of the IBM MQ event queue being resumed for MQEV processing.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_EVENT_Q_NAME.

11.32 STOP EV

Use the MQSC command **`STOP EV`** (or it's equivalent PCF command **`MQG_CMD_STOP_EV`**) to stop the **MQEV** program. The **`STOP EV`** command can only be issued by an **MQEV** administrator see Chapter.19 Security on page 199 for more information.

Alternatively, on z/OS, the MQEV program can be stopped using the MVS stop command. See 9.4 Stopping MQEV using the MVS STOP command on page 47.

There are two features of the **`STOP EV`** command which makes it somewhat special:

1. Firstly, the **`STOP EV`** command is special in that it will only be executed if the command was issued after the **MQEV** itself was started. This means that if there is an 'old' **`STOP EV`** command on the command queue when **MQEV** is started it will be ignored.
2. This command will not generate a response. The reason for this is that the command can be used, say from **MQSCX,** without concern for whether the **MQEV** is not running or in the process of ending. In other words, **MQSCX** will not hang waiting for a response since it will not expect one.

11.32.1 Syntax diagram for STOP EV

```
►►─ STOP EV ──────────────────────────►◄
```

11.32.2 Parameter descriptions for STOP EV

(None)

11.33 SUSPEND EVQ

Use the MQSC command **`SUSPEND EVQ`** (or it's equivalent PCF command **`MQG_CMD_SUSPEND_EV_Q`**) to suspend an event queue from being processed by the **MQEV** event processor.

A log file entry will be written by this command showing the queue name that was suspended.

11.33.1 Syntax diagram for SUSPEND EVQ

```
►►─ SUSPEND EVQ( queue-name ) ────────►◄
```

11.33.2 Parameter descriptions for SUSPEND EVQ

(queue-name)

The name of the IBM MQ event queue being suspended from MQEV processing.

When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_EVENT_Q_NAME.

12 Alerts

Alerts can be created in **MQEV** and used for a number of purposes. They can be created either by a user or a script.

12.1 Alert Definition

The following table briefly lists the main fields of an alert and their meaning.

Field	Meaning
`ALERTTI`	Alert time. The time the alert was raised.
`CATEGORY`	A free-form string allowing you to group together alerts
`EVENTID`	The id of the event associated with this alert (if any)
`EVOBJNAME`	The associated Object name (if any)
`EVOBJTYPE`	The type of the object given by the OBJNAME value.
`EVQMGR`	The associated Queue Manager name (if any)
`EXPIRETI`	Expire time. The time this alert will expire and be removed.
`RETINTVL`	How long, in seconds, that this event should be kept If necessary a large value, such as 999999999, will essentially make this non expiring
`SEVERITY`	A severity setting of how important this alert is. **Value** — **Suggested use** **TERM** — Serious error which prevents MQEV from processing **SEVERE** — Severe error detected **ERROR** — Used for general errors found in MQ event processing **WARN** — Used for situations which aren't necessarily errors but may require user attention. **INFO** — Low level information alerts. Normally users would not be expected to see these alerts. They are only shown if explicitly asked for.
`EVSTREAM`	The stream that the event associated with this alert is on (if any)
`TEXT`	A text description of the issue.

Alerts can created with a reference to an event, if necessary. However, this is not mandatory, and you may have a need to create alerts which do no refer to an event.

12.2 Alert Uses

Alerts can be used in a number of ways:

- To alert a user of a situation that requires attention
- As a reminder to a user that something ought to be done
- As a reminder to a script function to check the status of something

we will discuss each of these in turn in the sections that follow.

12.2.1 User Alert

Perhaps the most common use of an alert is just to bring a situation to the attention of a user. This is most often done because a criteria detected in the `MQEVEvent()` script function. The expectation is that an MQ Administrator periodically checks the status of alerts in the system. The simplest way of of doing this is just to issue the command:

```
DISPLAY EVALERT(*)
```

Or, perhaps more likely the Administrator has a command console which displays the status of events in the system. For example, an MO71 GUI display.

Of course in an ideal world there would be no alerts. However, if the system wishes to bring something to the user attention then the alert would defined. For example, suppose we wished the user to be alerted to any form of authorisation failure in the system. We might have the following script:

```
func MQEVEvent()
  if (event.evtype = AUTHOR)
    @text    = event.summary
    @objname = event.evobjname
    @objtype = event.evobjtype
    ADD EVALERT TEXT('<@text>')  +
        EVOBJNAME('<@objname>')  +
        EVOBJTYPE(<@objtype>)    +
        CATEGORY(AUTHOR)
  endif
endfunc
```

This simple script checks the type of event and if it is an authorisation event it will add a new alert with the text of the alert equal to the summary text of the event. By specifying a category we can easily reference or display just the authorisation alerts.

In addition, this alert will have the **`EVENTID`** filled in by the system, since it was created within the `MQEVEvent()` function.

12.2.2 User Reminder

Of course there is nothing to prevent a user from just manually adding their own alerts. This can be useful to set a reminder for themselves. For example, suppose we wish to remind ourselves that we ought to upgrade to the latest maintenance on a queue manager.

You could imagine issuing the following command:

```
ADD EVALERT TEXT('Upgrade MQ maintenance level') EVQMGR(MQG1)
SEVERITY(WARN) CATEGORY(REMINDERS)
```

Of course the severity of the alert would depend on the type of alert itself.

12.2.3 Script Reminder

There can be times where a script detects a condition but that it only becomes 'an error' if the condition is maintained for some period of time. Consider, for example, a 'QUEUE HIGH' event. Actually receiving a 'QUEUE HIGH' event may not be regarded as a problem. However, if a queue remains 'QUEUE HIGH' for, let's say, a minute then that might be considered a problem. How would we code this? Clearly we don't want to have our script processing to actually issue a wait since that would prevent subsequent events being processed which would defeat the point. Well, it will be no surprise that the way we can process this is to use alerts.

Situations of this nature follow a similar pattern. There is an event that raises the alert level and another which clears it. In this case 'QUEUE HIGH' and 'QUEUE LOW' respectively. All we need to do therefore is raise an alert when we see 'QUEUE HIGH' and remove it when we see 'QUEUE LOW'. If the alert we raise actually lives long enough to expire then that is a real problem and perhaps we should tell someone. In this case we'll just raise another alert.

The following script demonstrates the processing:

```
*********************************************************************
* Function for processing an event                                  *
*********************************************************************
func MQEVEvent()
  @objname = event.evobjname
  @objtype = event.evobjtype
  if (event.evreason = PERQDPHI)
    *****************************************************************
    * Add a temporary alert that will expire in 60 seconds
    *****************************************************************
    ADD EVALERT TEXT('Queue High') +
        CATEGORY(QHIGHTEMP)        +
        EVOBJNAME('<@objname>')    +
        EVOBJTYPE(<@objtype>)      +
        SEVERITY(INFO) RETINTVL(60) REPLACE
  endif
  if (event.evreason = PERQDPLO)
    *****************************************************************
    * Remove both alert types, temporary and final, wildcarded CATEGORY
    *****************************************************************
    REMOVE EVALERT TEXT('Queue High') +
           CATEGORY(QHIGH*)           +
           EVOBJNAME('<@objname>')    +
           EVOBJTYPE(<@objtype>)
  endif
endfunc

*********************************************************************
* Function called when an alert expires                             *
*********************************************************************
func MQEVAlertExpire()
  if (alert.category = 'QHIGHTEMP')
    @objname = alert.evobjname
    @objtype = alert.evobjtype
    ADD EVALERT TEXT('Queue High') +
        CATEGORY(QHIGH)            +
        EVOBJNAME('<@objname>')    +
        EVOBJTYPE(<@objtype>)      +
        SEVERITY(WARN) REPLACE
  endif
endfunc
```

This script shows just how easy it is to delay processing, let's work through the code.

All events will come in to the `MQEVEvent()` function. In this case we are interested in the 'QUEUE HIGH' and 'QUEUE LOW' events. So, the first thing we do is check the reason for the event i.e.

```
if (event.evreason = PERQDPHI)
```

If it is a 'QUEUE HIGH' event then all we want to do is add a temporary alert with a fairly short expiry time, let's say 60 seconds. We also set it at the lowest severity since this is not an error yet. We also choose a category name for these 'temporary queue high' events, let's say 'QHIGHTEMP'. This can be any name you find appropriate.

If, on the other hand, we are receiving a 'QUEUE LOW' event then we can remove any previously created alerts since all seems well. So, we remove any 'QHIGH' or 'QHIGHTEMP' alerts for this queue name. We do this in a single command by using a wildcarded category name of 'QHIGH*'.

However, if the 'QHIGHTEMP' alert actually expires (ie. a 'QUEUE LOW' event does not arrive within 60 seconds and delete it) `MQEVAlertExpire()` is called. Now we need to actually raise a “real” alert. In this case we choose just to add a higher severity alert but you could choose to send an email/SMS or whatever.

Of course this is just an example. You could change the code to suit your requirements. For example, you might only want to do this type of processing for certain queue names or even at only certain times of the day. You can make the conditions as specific as you like.

There are in fact other ways of checking for a 'QUEUE HIGH' event. Depending on your familiarity with MQ event reason code, you may prefer to use of these instead.

Instead of:

```
if (event.evreason = PERQDPHI)
```

we could have said

```
if (cfh.reason = 2224)
```

or even

```
if (cfh.reason = const.MQRC_Q_DEPTH_HIGH)
```

They all have the same effect.

12.3 Alert Retention

Given the various different uses for alerts, sometimes the lifespan of an alert is quite long, say several days or weeks, for example, until the new maintenance level has been applied; and sometimes it is much shorter, say a minute or two while you wait to see if a queue high state lasts long enough to worry about it.

Due to these two differing uses, the default alert retention interval that is defined on the EV object (see **`ALTER EV`** on page 57) is measured in days, but if you specify it on an **`ADD EVALERT`** command (see page 52) then it is measured in seconds.

It is expected that if you are adding an alert with a long lifespan, as a reminder for a human operator, you will likely just allow the default retention value to be used, or will be using a GUI Tool such as MO71 which makes the units obvious, and allows you to type in “5 days” if you need to provide a non-default value.

If you are writing a script as shown in the example above, you'll want a much shorter lifespan than could be specified in days, and so for all these cases, it is clear that a value in seconds is appropriate.

12.4 Maximum Number of Alerts

Alert definitions are saved to the data queue in a single message. This means that you cannot have more alerts than will fit into, say, a 4MB message. In practice this should not be a limitation since this would allow for 10's of thousands of alerts. The expectation is that the user will act on the alerts and delete them once they have been dealt with. However, if the user does not do that, and this limit is reached, **MQEV** will start pruning the list of alerts as required, starting with the oldest, lowest priority, first.

12.5 Alert Publication

MQEV supports a number of different objects and to discover their value you need to issue some form of `DISPLAY` command. For most objects this if fine however for `ALERT` objects this could lead to a considerable amount of polling. After all, you want to find out about new alerts as soon as possible. So, to solve this problem **MQEV** will publish out any changes to any alerts. This way you find out about changes very quickly and yet if there aren't any updates then the system does not burn unnecessary CPU and bandwidth.

MQEV will publish alerts on the topic `'MQGem/MQEV/Alerts'`.

Note that **MQEV** will not publish changes to alerts at severity `INFO.` Alerts at this level are considered internal and would not normally be displayed externally.

MO71 subscribes to this topic string when you have an alerts list dialog open, so it can automatically refresh when a new alert is added or one is removed, saving you from needing to repeatedly press the 'Refresh' button or use Auto Refresh on the dialog.

12.5.1 Publication Message Format

The format of the publication is a PCF message with the following values:

Field	Value
MQCFH.Command	**MQG_CMD_ADD_EV_ALERT** **MQG_CMD_REMOVE_EV_ALERT**
Message Fields	**MQG_ATTR_ALERT_ID**
	MQG_ATTR_ALERT_TEXT
	MQG_ATTR_ALERT_SEVERITY
	MQG_ATTR_ALERT_CATEGORY
	MQG_ATTR_RETENTION_INTERVAL
	MQG_ATTR_ALERT_TIME
	MQG_ATTR_ALERT_Q_MGR
	MQG_ATTR_OBJECT
	MQG_ATTR_EVENT_ID
	MQG_ATTR_STREAM_NAME
Message Properties	**“Severity”** This means that you can subscribe to only certain severities.

13 Event Storms

Event storms are occasions when **MQEV** is deluged with lots of very similar events. This can happen for a variety of reasons but perhaps the simplest to consider is a rogue application. Consider the case where you have a badly written application which tries to connect to a Queue Manager and, if it fails, immediately tries again. This can cause a huge number of authority events being raised. This storm of events can clearly clog up the works to say nothing of the wasted storage.

To mitigate against event storms **MQEV** will check whether an event has been seen more time than a threshold value within a certain time interval. These values are configured in the `EV` object as **`STORMTHR`** and **`STORMINT`**. By default they have values of 20 and 60. This means that if the same event is seen more than 20 times in 60 seconds then the processing of the event is delayed. At the end of the 60 seconds a single event is processed with a count of how many times the event was seen in that interval.

If required the values for **`STORMTHR`** and **`STORMINT`** can be modified according to your requirements.

So, if an event storm is detected then at most (`STORMTHR+1`) events within each `STORMINT` interval will be generated. That is `STORMTHR` 'normal' events plus one storm event. The storm event will be the same as the first event in the storm with the addition of an **`EVREPEAT`** field which gives the number of events detected within the interval. So, for example, you can display the event storms detected with the following command:

```
DISPLAY EVENTS(*) WHERE(EVREPEAT)
```

13.1 Storm Alert

If an event storm is detected then **MQEV** will raise an alert. The text of the alert will be a description of the event and how many times it was detected within the time interval. The category of this event will be “$STORM”.

The alert will also contain a reference to the event id of the above mentioned event that counts the number of repeats, and the object name and type, if the event is related to an object.

Here is an example of an event and alert from an event storm.

```
EVQMGR(MQG1)         EVENTS($EVENTS)      EVTIME(2019-10-11 16:48:54 (Local))
EVREASON(INHPUT)     EVTYPE(INHIBIT)      EVOBJNAME(NON.PUTABLE)
EVOBJTYPE(QUEUE)     EVENTID(00000073)    CFHCMD(44)           CFHREASON(2051)
SUMMARY(Inhibit Put - Queue:NON.PUTABLE)                       QUEUE(NON.PUTABLE)
APPLTYPE(WINDOWSNT) APPLNAME(D:\nttools\q.exe)                 EVREPEAT(13)
```

Here is the alert that refers to this event.

```
EVALERT(1)           EVQMGR(MQG1)         EVOBJNAME(NON.PUTABLE)
EVOBJTYPE(QUEUE)
TEXT(Event Storm Detected 'Inhibit Put - Queue:NON.PUTABLE' repeated 33 times.)
SEVERITY(WARN)       CATEGORY($STORM)     ALERTTI(2019-10-11 16:49:54 (Local))
EXPIRETI(2019-10-25 16:49:54 (Local))    RETINTVL(1209600)    EVENTID(00000073)
EVSTREAM($EVENTS)
```

You can see that the alert provides the **`STREAM`** name, the **`EVQMGR`** name and the **`EVENTID`** which when combined uniquely identify the event that caused the alert to be raised.

14 MQEV Scripting

MQEV must always have a script file which tells it what to do as certain events occur. A default script file, ***mqev.mqx***, is provided in the installation. The provided installation file essentially does nothing. If it is not a requirement for **MQEV** to respond to events then this file can be left as it is, but the minimum structure shown in the provided file is mandatory even on z/OS where the PCF Accounting and Statistics are not available. In other words it is not necessary to have any actual scripting at all. However, if you wish to take actions upon the arrival of certain events then you can add some instructions to the script file. Don't worry though, it is very simple.

The basic principal is that there are some standard functions which are called at certain points in the processing. For example the function `MQEVEvent()` is called whenever **MQEV** receives an event of any type. This function is probably the most important function and the most likely one that you will want to add to. However, there are other functions as follows:

Function Name	Purpose
`MQEVEnding`	Called when **MQEV** is ending
`MQEVConnected`	Called when **MQEV** has just connected to a Queue Manager
`MQEVDisconnected`	Called when **MQEV** has disconnected from a Queue Manager
`MQEVRetryConnect`	Called when **MQEV** is retrying to connect to a Queue Manager
`MQAlertExpire`	Called when any alert expires
`MQEVEvent`	Called when **MQEV** receives an event of any kind
`MQEVAcctMQI`	Called when **MQEV** receives an MQI accounting message.
`MQEVAcctQ`	Called when **MQEV** receives a Queue accounting message.
`MQEVStatChl`	Called when **MQEV** receives an Channel statistics message.
`MQEVStatMQI`	Called when **MQEV** receives an MQI statistics message.
`MQEVStatQ`	Called when **MQEV** receives a Queue statistics message.

What you choose to do in the script is entirely up to you. For example, you could can write to a file, issue an MQ command, or raise an **MQEV** alert or perhaps issue an OS command (for example to send an email or SMS text alert).

The way you issue a script is very simple. Consider this simple code:

```
func MQEVEvent()
if (event.evtype = AUTHOR)
  ADD EVALERT TEXT(event.summary)
end
```

This code will check the type of event and if it is some type of authorisation failure then it will add a new alert. Of course we could have made it more specific. We could have checked the object name or type or perhaps the user causing the problem. Or perhaps we might want to check the time of day etc.

There is no practical limit to what you can check for nor what you can choose to do should that situation arise.

14.1 Invoking other programs from your script

There will be times in your scripts that you wish to invoke a program or command/shell script to do something outside of **MQEV**, such as email or text someone, or run a program, for example you could use the **Q** program to send messages to a queue that might trigger other processes. To do this, you can use the MQSCX control language function **`system()`**. For more details of the syntax of this and other MQSCX control language functions, see Appendix B. Expression Functions on page 210.

Here's an example of using the **Q** program to put a message to a logging queue when an event has arrived in **MQEV**. The command string to invoke is built up using the `_qmgr` system variable rather than hard-coding the queue manager name, and using the event association variable `event.summary` for the current event being processed. Read more about Association Variables and System Variables in 15.2 Variables on page 163.

```
func MQEVEvent()
  @CmdStr = "q -m" + _qmgr + ' -oLOG.Q -M"' + event.summary + '"'
  system(@CmdStr, const.SYNC)
endfunc
```

Invocations using the `system()` function can be either synchronous or asynchronous, specified in the optional second parameter. If the second parameter is omitted, the default value is to run asynchronously in order to maintain behaviour from prior releases. Depending on the command you are issuing, it may be that synchronous mode is more appropriate.

14.1.1 Synchronously

Invocations made using the `system()` function in synchronous mode only return control to **MQEV** once the command completes, so be careful not to use long running commands in this way.

Commands are invoked by the command processor (**`CMD`**) on Windows, and through the shell **`/bin/sh`** on other platforms, so redirection of output, say to a text file, is possible. Otherwise the output from the command may show up in the **MQEV** foreground window in some environments.

Using synchronous mode may be useful when testing the script to ensure you have the correct command string since output is easier to see.

We could imagine extending our earlier example like this using the _os system variable to choose a temporary file to redirect our command output to. Note on z/OS, running a program using the **`sh`** shell, means redirection of output can only use an HFS file, not an MVS dataset.

```
func MQEVEvent()
  @CmdStr = "q -m" + _qmgr + ' -oLOG.Q -M"' + event.summary + '"'
  if (_os = "WINDOWS")
    @CmdStr = @CmdStr + "> c:\temp\Q.out 2>&1"
  else
    @CmdStr = @CmdStr + "> /u/gemuser/Q.out 2>&1"
  endif
  system(@CmdStr, const.SYNC)
endfunc
```

14.1.2 Asynchronously

Invocations made using the `system()` function in asynchronous mode, return control to **MQEV** immediately, so if you have a long running command to invoke, you should perhaps use this mode.

```
  system(@CmdStr, const.ASYNC)
```

Redirection of output written to `stdout` or `stderr` is not possible in this mode.

15 Script Control Language

The **MQEV** script language is heavily based on the script language found in our **MQSCX** product. There are some minor difference with regard to naming of variables and some system variables but in general if you are at all familiar with the **MQSCX** product then you will have no problem writing **MQEV** scripts. And even if you have never used **MQSCX** you will find that the script language is very easy to pick up.

15.1 Getting started with the control language

If you are not familiar with the script language at all then it may be easier to play with the language using **MQSCX** rather than **MQEV.** The reason for this is that **MQSCX** is a user driven program whereas **MQEV** is driven by the arrival of event messages. In other words we can just type in some commands in **MQSCX** and see their effect whereas in **MQEV** we have to modify one of the functions mentioned in the previous section and then arrange for that function to be invoked. So, since your **MQEV** also allows you to run **MQSCX** let's use that program for a while until we get a feel for the language.

So, let's start simple, run the **MQSCX** program. Suppose I just want to issue something to the screen. Well I have the print command. So, type in the following command and press enter.

```
print "Hello World"
```

We see that **MQSCX** responds, not surprisingly with what we asked it to print. Suppose we give it something a little more complicated.

```
print 4 * 5
```

We can see that **MQSCX** treating what it has been given as as expression to be evaluated.

So, can we print out something we have received from the command server. Suppose we want to print out some events, how could we do that ? Well, try typing in the following but make sure you type it just as shown (better yet use copy/paste[26]).

```
foreach(DISPLAY EVENTS(*)) print SUMMARY; endfor
```

So, we run this command and we see that MQSCX does indeed write out all the event summary text but we also get lots of other stuff. Well, these are the commands that **MQSCX** is executing under the covers. Normally these wouldn't be shown but they can be very handy to see it going on. We can suppress them using the command

```
=echo langcon(no)
```

This command basically says don't echo control language lines even when entered from the console. So, if we issue the foreach command again.

```
foreach(DISPLAY EVENTS(*)) print SUMMARY; endfor
```

We now see that just the summary text displayed. However, suppose we also want to see the time the event happened. How would we do that ? Well, use command recall and add 'evtime' to the print statement.

26 You have to be a little careful with copy and paste though, especially when the text contains double quote characters. Unfortunately there are different flavours of double quote characters used in documents, you need to ensure that you use the standard one for programming.

```
foreach(DISPLAY EVENTS(*)) print EVTIME,SUMMARY; endfor
```

Issue this command and we see that we now have the event time and the summary text displayed. In **MQEV** we probably won't be printing things to the screen that often but it can be one of the simplest and easiest debugging aids so it is a useful thing to learn first. The print statement has a number of options which control how the data is displayed but for now let's concern ourselves with what else is going on in this command.

The first thing you would have noticed is the '`foreach`' command. This is a very simple command which takes some form of `DISPLAY` command as a parameter and will execute the sequence of instructions up to it's corresponding '`endfor`' statement for each response it receives from the command server. The responses themselves are not echoed to the screen but can be shown by issuing a `=set noecho(yes)` command.

So here we have a very simple way of find out the current state of the queue manager, or of **MQEV.** We can issue any `DISPLAY` command and then parse the results. So, how do we control whether we are sending the command to IBM MQ or to MQEV ? Well, this is very simple and is controlled by the last =`mqev` or =`mqsc` command issued.

So, that gives you an idea of how to issue an MQ command. But of course in **MQEV** we are normally dealing with data that has been sent to us in an event message rather than by us issuing a command. How do we access the event data? This brings us on to variables.

15.2 Variables

Perhaps the mainstay of any programming language, variables allow us store, retrieve, calculate and compare values. **MQEV** supports four types of variables, association, user, system and response. We will consider each of these in turn.

15.2.1 Association variables

Association variables are sets of variables that are associated with a certain aspect of the state of the program. They all use a prefix to identify which aspect of the current state they are associated with. There are a number of different types of associated variable:

Prefix	Meaning	Examples
`event.`	Event variables allow you to access the 'current' event message. These variables are only available in the `MQEVEvent()` and `MQEVAlertExpire()` functions. The names of the event variables match the names of the MQSC keywords for the output fields in a **`DISPLAY EVENTS`** command.	`event.evtype` `event.summary`
`data.`	Data variables allow you to access the 'current' accounting or statistics record. These variables are only available in the accounting and statistics functions.[27] The names of the data variables match the names of the MQSC keywords for the output fields in the equivalent display command. For example, for `MQEVAcctMQI()` look in **`DISPLAY ACCTMQI`**. There are several constructed fields in the DISPLAY commands – these cannot be accessed from the function. The description of the DISPLAY commands indicate which these are and how to obtain the same data. For example `ALLPUT` is constructed from the sum of `PUTNP`, `PUTP`, `PUT1NP` and `PUT1P`.	`data.rversion` `data.expired`
`mqmd.`	MQMD variables allow you to access the MQMD fields associated with an event message or accounting and statistics record. These variables are only available in the `MQEVEvent`() function and the accounting and statistics functions.[27] The names of the MQMD variables match the names as shown in the 'C' structure definition in `cmqc.h`.	`mqmd.UserIdentifier` `mqmd.ReplyToQMgr`
`cfh.`	CFH variables allow you to access the MQCFH fields associated with an event message or accounting and statistics record. These variables are available in the `MQEVEvent`() function and the accounting and statistics functions.[27] The names of the CFH variables match the names as shown in the `MQCFH` 'C' structure definition in `cmqcfc.h`.	`cfh.reason` `cfh.command`

27 The functions `MQEVAcctMQI()`, `MQEVAcctQ()`, `MQEVStatChl()`, `MQEVStatMQI()`, and `MQEVStatQ()`.

Prefix	Meaning	Examples
before.	This form of association variable allows to you to particularly target the 'before' version of a variable. As you might expect then this only applies to change configuration events that are notifying MQEV that something has changed. By comparing the 'before' variable with the 'after' variable you could check whether someone is changing a particular field of an object.	`before.maxmsgl`
after.	See 'before.' above	`after.maxmsgl`
cmddata.	Some events contain groups of fields. This prefix allows you to specifically target fields in the 'command data' group.	`cmddata.queue`
cmdctx.	Some events contain groups of fields. This prefix allows you to specifically target fields in the 'command context' group.	`cmdctx.evorigin`

alert.	Alert variables allow you to access the fields of an expiring alert. These variables are only available in the `MQEVAlertExpire()` function.	`alert.text`
		`alert.objname`
		`alert.objtype`

const.	Constant variables allow you to use a text string to refer to standard MQ constants. These variables are available at any time.	`const.MQCA_Q_NAME` `const.MQPER_PERSISTENT`

In addition to these prefixes, there are some fields which have multiple constituent parts. For these fields you can use a suffix to identify which part you want to refer to.

Suffix	Meaning	Examples
.long	This suffix is applicable only to indicator event variables and it allows you to access specifically the long term average value. This suffix only applies to event variables.	`event.nettime.long`
.short	This suffix is applicable only to indicator event variables and it allows you to access specifically the short term average value. This suffix only applies to event variables.	`event.nettime.short`

15.2.2 User Variables

User variables are variables which have, for the most part, been defined by control program itself. They are the only variables for which you can change their value. The key thing about user variables is that they always start with an '@' characters. The variables can contain any type of data, strings, integers, lists or real numbers. All of the following are valid.

```
@a = 1
@a = "This is a test string"
@a = 3.1415
```

It is not necessary to define a variable before use you just use them whenever you wish. An easy way to check the value of a variable is to just print it to the screen.

```
print @a
```

Note that if we try and print a variable that does not exists yet we get an error message

```
print @xxx
Error Message: Variable '@xxx' is not defined in the current scope.
```

There are exceptions to this and those are environment variables. If a variable has not been defined by the program then **MQEV** will look to see whether there is an environment variable of that name, Try this:

```
print @temp
```

The chances are that you have a 'temp' environment variable so the print statement will have printed it's value. Now you can override the value will something like this.

```
@temp = "My Value"
```

If you do this then you have created an in program version of a `@temp` variable. You have not changed the temp environment variable itself. So, the program need not worry about accidentally using a name that has an environment variable. However, environment variables can be a useful way to effectively pass parameters into your command files. For example, you could specify the path to a file or perhaps switch on a debugging flag.

There are some rules about naming of user variables

- The name is limited to 30 characters, which includes the '@' character.
- The character following the '@' must be alphabetic or the underscore '_' character.
  It is recommended that you avoid calling your user variables starting with @_ since these may be used by the **MQEV** program itself.
- Following characters can be alphanumeric, '.' or '_'
- User variables are case sensitive so @a and @A are different variables.

When a variable is defined, or first used, then it is defined in the current stack frame. The variable is available in the current stack frame or any higher stack frames. Any variables defined at the lowest stack frame will remain until the program ends unless it is explicitly deleted using the **delvar()** function, see 'Expression Functions' for a description of available functions. However, variables created in functions, i.e. Higher stack frames, will be deleted when the function returns. Please see Variable Scope and Stack Frames on page 170 for more information on this topic.

15.2.3 Arrays

There are times when you want to store large amounts of data. For example, suppose you wish to read all the queue definitions and store the values. It would not be feasible to define a different variable for each value. So, **MQEV** allows you to use arrays. An array is essentially a user variable with a subscript. The following are all valid uses of an array:

```
@var[5]                - the 5th element of array @var
@queue[6,10,2]         - multiple subscripts can be used
@mult[3,5,6,1]         - up to four subscripts can be used
```

It is not necessary to define the array before use. Arrays are sparse. This means that just because element @var[5] is defined it doesn't necessarily mean that element @var[4] is defined. Arrays are indexed from 1. Array element @var[0] is not valid. Each element of the array can contain a different data type. For example the following sequence is perfectly valid.

```
@val[1] = "Value"
@val[2] = 24
@val[3] = "Average"
@val[4] = 7.8
```

Once a variable is defined it remains until the program ends unless it is explicitly deleted using the **delvar()** function, see 'Expression Functions' for a description of available functions. The **delvar()** can be used to delete the entire array or just a single element.

15.2.4 System Variables

System variable are special variables which are provided by **MQEV** itself. Most system variables can not be changed by an assignment. System variables all start with the underscore '_' character.

The following system variables are defined:

Name	Value
_ccdt	The name of the current CCDT file.
_ccdtmode	Boolean indicating whether **MQEV** is currently in CCDT mode.
_client	Boolean indicating whether **MQEV** is currently connected as a client or not.
_cmdok	Boolean indicating whether the previous MQ command was successful.
_cmdlevel	The integer command level of the Queue Manager **MQEV** is administering. A value of -1 is returned if MQSCX is not currently connected.
_connqmgr	The name of the Queue Manager **MQEV** is currently connected to. This will differ from system variable _qmgr if you are connected in via mode.
_dspcmd	Boolean indicating whether the current event is a DISPLAY command event.
_emit	The emit object to use to emit this event. This value can be changed as required. Setting it to the value "$null" will cause the event not to be emitted. A value of "" will cause the event to sent to the emit object defined on the stream the event is written to.
_errno	The current system errno.
_errnostr	A brief description of the current system errno.
_idxEach	Only meaningful inside a foreach(...) loop. It contains the index of the object which is being processed. Outside of a foreach(...) loop the variable has the value 0.
_idxItem	Only meaningful inside a foritem(...) loop. It contains the index of the item which is being processed. Outside of a foritem(...) loop the variable has the value 0.
_idxWhile	Only meaningful inside a while(...) loop. It contains the iteration number of the loop. Outside of a while(...) loop the variable has the value 0.
_item	Used in a foritem(...) endfor loop. It contains the current list item value.
_lastrc	The last MQ reason code. For example from an =conn command If **MQEV** is not currently connected it will return MQRC_NOT_CONNECTED (6124)
_lastrcstr	A brief text description of the last MQ reason code above.
_lastresp	The last response from an `DISPLAY` command For example, “`QUEUE(Q1) TYPE(QLOCAL) CRDATE(25122013).....`”
_lic_cn	The licence contact name.
_lic_em	The licence email address.
_lic_lc	The licence licensee value.
_lic_rm	The number of days remaining on the licence.
_nl	Newline character. When printed this string will cause a new line.
_numEach	After processing a foreach(...) loop this variable contains the number of iterations of the loop.
_numItem	After processing a foritem(...) loop this variable contains the number of iterations of the loop.
_numWhile	After processing a while(...) loop this variable contains the number of iterations of the loop.

_os	The Operating System **MQEV** is running on. Possible values are: “AIX”,”LINUX”,”PLINUX”, “WINDOWS” or “MVS”
_platform	A string containing the platform of the Queue Manager **MQEV** is administering. A value of “NOTCONNECTED” is returned if **MQEV** is not currently connected.
_qmgr	The name of the Queue Manager **MQEV** is currently administering
_responses	The number of responses from the command server for the last `DISPLAY` command
_sep	Separator. When printed this string will cause a separator line to be written.
_stream	The stream this event will be written to. This value can be changed as required. Setting it to the value `"$null"` will cause the event to be discarded. A value of `""` will cause the event to sent to the default stream.
_time	The current time in seconds from January 1st 1970.

15.2.5 Response Variables

Response variables are the simplest way of processing responses from a `DISPLAY` command. A response from a `DISPLAY` command might look like this:

```
QUEUE(SYSTEM.DEFAULT.LOCAL.QUEUE)    TYPE(QLOCAL)   CURDEPTH(0)
```

This response will automatically define three response variables, QUEUE, TYPE and CURDEPTH. The values of the response variables will be values contained between the brackets. A response variable will always exist but they may have zero length if not returned by the command itself. This is helpful when writing scripts which are to be MQ version independent. You can specify any name, for example one that hasn't been defined yet on this Queue Manager, and **MQEV** will just return the empty string if it is not returned on the command. It does however mean that you have to be careful about how you spell the variable since spelling mistakes will not be reported.

Response variable are case insensitive, so although the command server returns the field in upper case it quite all right to refer to them in lower case.

For example if we printed out the values we would see the following.

```
print QUEUE,TYPE,CURDEPTH
SYSTEM.DEFAULT.LOCAL.QUEUE QLOCAL 0
```

Response variables are over written at the next MQ command. Therefore if you need to save the value of a response variable you should assign it to a user variable. For example:

```
@queue = QUEUE
@depth = CURDEPTH
```

As in the =`WHERE` clause indicator fields can be referenced using the suffixes .short' and '.long'.

For example:

```
print qtime.short, qtime.long
```

There are some responses which can not be accessed via response variables.

Consider the following:

```
DISPLAY QUEUE(SYSTEM.DEFAULT.LOCAL.QUEUE) TRIGGER
QUEUE(SYSTEM.DEFAULT.LOCAL.QUEUE) TYPE(QLOCAL) NOTRIGGER
```

Unfortunately MQSC does not respond with something like TRIGGER(YES) or TRIGGER(NO). Instead it returns the single string 'NOTRIGGER'. If you really need to know whether the queue returned TRIGGER or NOTRIGGER you can use processing like this.

```
if (findstr(_lastresp,"NOTRIGGER")) @trig = "NO"; else @trig = "YES"; endif
print @trig
```

This code fragment uses a system variable that we mentioned in the previous section, '_lastresp'. _lastresp contains the whole of the response from the last DISPLAY command[28]. So, if we search for the word "NOTRIGGER" in that response we can determine whether triggering is enabled or not and set our user variable accordingly.

Perhaps the most common use of response variables are in a foreach() clause. We have already seen a few examples in the introduction, we just didn't know they were called response variables.

Consider the following.

```
foreach(DISPLAY QUEUE(*))
  print queue,curdepth
endfor
```

In a foreach() clause the sequence of commands contained in the foreach() block is invoked for every response to the DISPLAY command. So, naturally any response variables contained in the response are available during the loop execution.

28 Provided that no other MQ commands have been issued since the DISPLAY response

15.3 Variable Scope and Stack Frames

All user variables belong to a particular stack frame. When a user variable is defined or first used it is created in the current stack frame. The current stack frame is merely the current level of the stack. For example, when the program starts you are at stack frame 0. If the program now invokes a function you are now at stack frame 1. If that function invokes another function you would be at stack frame 2 and so on. Each of these stack frames can have user variables defined. When execution reached the end of a stack frame then any variables owned by that stack frame will be deleted. Consider the following example:

```
func bar(a)
  print @a
endfunc

func foo(a)
  bar(@a+1)
  print @a
endfunc

@a = 1;
foo(@a+1)
print @a
```

This results in the output:

```
3
2
1
```

So, even though the program only ever prints out the value of @a we get three different values. This demonstrates that we can have multiple variables each with the same name at the different stack levels. As you can see, this is achieved because each of the functions specified that 'a' is a parameter. By definition parameters are variables local to the current stack frame. However, suppose we didn't define a variable in the functions, would it be visible ?

Let's try the following:

```
func bar(a)
  print @a,@b
endfunc

func foo(a)
  bar(@a+1)
  print @a,@b
endfunc

@a = 1;
@b = 2;
foo(@a+1)
print @a,@b
```

We've just added a new variable, @b, and given it the value of 2 in the main program. The functions make no declaration of @b, they just print out the value.

As suspected the output is now:

```
3 2
2 2
1 2
```

When a function accesses a variable it finds the definition with the nearest stack frame. So, if we updated the value of @b in function foo() would the change be reflected in the main program? Let's try it:

```
func bar(a)
  print @a,@b
endfunc

func foo(a)
  @b = 3
  bar(@a+1)
  print @a,@b
endfunc

@a = 1;
@b = 2;
foo(@a+1)
print @a,@b
```

The output is:

```
3 3
2 3
1 3
```

Yes!, the updated value is seen by everyone. Perhaps this is not a surprise since there is only one actual variable @b. However, suppose now I wanted function foo() to have it's own variable @b, to ensure that it couldn't mess up its callers value. Well we can do that by adding a var.

```
func bar(a)
  print @a,@b
endfunc

func foo(a)
  var b
  @b = 3
  bar(@a+1)
  print @a,@b
endfunc

@a = 1;
@b = 2;
foo(@a+1)
print @a,@b
```

The output is now:

```
3 3
2 3
1 2
```

So, now we see that both foo() and bar() see @b as having the value 3 but when we drop back to the main program it sees @b as having the original value 2.

The notion of stack variables is very common among programming languages and you should find it fairly intuitive. The only aspect that is somewhat unusual is that variables can be found anywhere up the stack. Most languages have the concept of local and global variables. By defining variables in a function **MQEV** allows you to define values which are partially global, i.e. only visible to functions further up the stack.

15.4 Expressions

Expression in **MQEV** follow the normal convention in term of operator precedence. A full list of the operators are given in 'Expression Operators' on page 209.

15.4.1 Data Types

MQEV understands the following data types

- String
  A string can be treated as a list using for foritem(...) clause.
- Integer
- Real

15.4.2 Coercion

In general data types are coerced automatically so that an expression can contain operands of different types without problem. Perhaps the most surprising and yet useful coercion is when a string is used as a number. Consider the following expression.

```
Print +"Hello World"
```

Here we have an arithmetic operator '+' and a string operand. **MQEV** could have disallowed this combination but instead it coerces the string into a number, and that number is the string length. So, the expression about will print out 11. This is more useful than you might think. Consider the following example:

```
if (channel) print "Client connection"; endif
```

Here we are using a string value as the expression in a boolean field. The `if`() clause wants a `TRUE` or `FALSE` result and yet we are passing it a string. However, since a string is coerced to a numeric value by virtue of it's length we get the desired result. In other words, the if expression will be `TRUE` if the channel variable has a value and `FALSE` if it has no length.

However, there are some combinations of operand and operators which are not allowed. For example consider the following expression.

```
print "Hello" / "World"
```

This really doesn't make any sense. Trying to divide one string by another has no real meaning so **MQEV** will report an error.

15.4.3 String Concatenation

One common operation you may wish to perform is concatenating strings. This can be achieve very simply using the '+' operator. For example:

```
@str = "Hello" + "World"
print @str
HelloWorld
```

Note that **MQEV** concatenates the string exactly, no spacing character is added. You need to add that to the original strings if required.

However, now consider the following:

```
@depth = 50
@str   = "CURDEPTH(" + @depth + ")"
print @str
61
```

61! This may have seemed a very strange result until you remember our discussion previously about coercion. Remember that if you use a string and a number in the same expression that the string length will be used in the calculation. What we need is a way of converting the number into a string. And luckily there is a very simple function `str`(). So, let's try this:

```
@depth = 50
@str   = "CURDEPTH(" + str(@depth) + ")"
print @str
CURDEPTH(50)
```

Great, that's just what we wanted. The `str`() function can be used on any data type.

15.5 Inserting code fragments

It is possible to add extra code into the command stream. Consider the following:

```
@cmd = "DISPLAY QUEUE(SYSTEM.DEFAULT.LOCAL.QUEUE)"
@cmd
```

The @cmd statement will actually run the command that is contained in the variable. This may not be too surprising but it is actually more powerful than it first appears. Now consider the following:

```
@cmd = "if (curdepth) print 'Queue is not empty'; endif"
@cmd
```

This demonstrates two things.

1. **There is no limit on the commands which can be inserted in this manner.**
   The control program can construct the command at run time and then execute them; a self modifying program if that is useful.

2. **This examples shows how to insert quotes characters inside a string**
   To identify a string in an expression you can use either ' or “ characters. The string is ended when that character next appears in the string. So, to write a string inside another string all you need do is use the other quote character.

15.6 Substitution commands

Substitution commands are a simple way of inserting a variable into command. Suppose we wish to display a queue but want the queue name to come from a variable. We could do the following:

```
@q = "SYSTEM.DEFAULT.LOCAL.QUEUE"
@cmd = "DISPLAY QUEUE(" + @q + ")"
@cmd
```

This works just fine and could well be the way to do it in some cases. However, using substitutions we can use a short cut. Look at the following:

```
@q = "SYSTEM.DEFAULT.LOCAL.QUEUE"
DISPLAY QUEUE(<@q>)
```

This will generate the same command. Essentially **MQEV** looks for a variable between <> characters. If it finds either a user or system variable between the <> characters then the variable is replaced by the value of the variable. Note that this substitution will only happen when the command is about to be run. It will not happen in an assignment as is demonstrated by the following:

```
@q   = "SYSTEM.DEFAULT.LOCAL.QUEUE"
@cmd = "DISPLAY QUEUE(<@q>)"
print @cmd
DISPLAY QUEUE(<@q>)
```

This means that you can easily insert code fragments containing substitutions.

15.6.1 Functions

There are a number of functions defined which can be used in expressions. For example functions are available for calculating dates, opening files, finding substrings and many more.

Please refer to 'Expression Functions' on page 210 for the complete list.

In addition you can write your own function. Please refer to Functions on page 182 for more information.

15.7 General syntax

Command can be entered anywhere on the line and white space is ignored. So, the following are all valid and equivalent:

```
@a=3
@a  =   3
    @a  =   3;
```

The last examples shows that an end of statement character is optional. This is really just a matter of style whether you like to add the semicolons or not. These are end of statement markers. Where they are really useful is if you want to enter multiple commands on the same line like this.

```
@a = 3; @b = 4; @c = 5
```

Here we enter three statements on the same line. In the last one we can omit the end of statement indicator since the end of the line will essentially do that for us.

15.7.1 Continuation

There are times when we might want to have single statement span a single line. To do this we use the same syntax as MQSC commands and have a character at the end of the line which indicates 'continuation'. There are two ways of signifying continuation.

1. **If the line ends in a '+'**
   The following line is concatenated starting at the first non-blank characters

2. **If the line ends in a '-'**
   The following line is concatenated starting at the first character

15.7.2 Comments

Any line where the first non-blank character is an asterisk '*' is considered a comment.

15.8 Statements

15.8.1 break

The break statement can only be issued from within a loop. The loop can be a while(...) clause, a foritem(....) clause or a foreach(....) clause. It causes execution to continue at the statement following the end of the current loop.

15.8.2 continue

The continue statement can only be issued from within a loop. The loop can be a while(...) clause, a foritem(....) clause or a foreach(....) clause. It causes execution to continue at the end of the current loop. Essentially this is a short-cut way of processing the next iteration of the loop.

15.8.3 foreach(....) clause

The foreach(..) clause is the way the responses to a `DISPLAY` or `RESET QSTATS` command are processed. The syntax is as follows:

```
foreach( <DISPLAY COMMAND> )
   <control statements>
endfor
```

You can specify any MQSC `DISPLAY` command or `RESET QSTATS` command in the brackets immediately following the foreach word. The command must be a single MQ command. The set of statements between the foreach(....) clause and the endfor statement will be run for each response from the command server. The statements will not start processing until all responses have been received from the command server. foreach(....) clauses can be nested which means that you can query other MQ objects based on the responses you receive from the first foreach clause.

The MQ command and the responses are not echoed to the screen. However, if you wish to see the commands, say for debugging purposes, you can display non-echoed lines using the command =show noecho(yes).

There are two other statements, break and continue, which can control the flow of the loop. In addition, if the while loop is in a function you can use return and goto to break out of the loop. Note that you can use goto to break out of a loop but you can not use goto to jump in to a loop.

15.8.4 foritem(....) clause

The foritem(....) clause provides a simple way to process a list. A list it just a simple comma separated string.

The syntax is as follows:

```
foritem( <String> )
   <control statements>
endfor
```

As an example consider the following:

```
foritem("a,b,c")
  print _item
endfor
```

When you run this set of commands **MQEV** will print out the values “a”, “b” and then “c”. It will run the sequence for each item in the list and put the current item in the system variable **_item**. The item will be stripped of any whitespace characters, such as spaces. For example the statements:

```
foritem("   a  ,  b ,  c  ")
  print _item
endfor
```

will yield the same results.

The foritem(...) clause is particularly useful when processing lists in MQ objects. For example, take a look at the namelist example namelists.mqx.

```
foreach(DISPLAY NAMELIST(*) NAMES)
  foritem(NAMES)
    print _item
  endfor
endfor
```

This very simple sequence of commands will print out the names contain in all of the defined namelists. Of course it isn't just namelists that use lists. Channel exits, group names, connection names, channel auth user lists and more all use lists of values.

There are two other statements, break and continue, which can control the flow of the loop. In addition, if the while loop is in a function you can use return and goto to break out of the loop. Note that you can use goto to break out of a loop but you can not use goto to jump in to a loop.

15.8.5 fprint statement

The fprint statement allows data to be written to a file. The statement is the same as the print statement we are already familiar with except it has an initial parameter which identifies the file to write to. So, suppose we wish to take the namelists example in the previous section and write the namelist names to a file rather than to the screen. We could modify the code as follows:

```
@hf = fopen("c:\temp\namelist.txt","w")
foreach(DISPLAY NAMELIST(*) NAMES)
  foritem(names)
    fprint @hf,_item
  endfor
endfor
fclose(@hf)
```

You can see that very easily we can generate a file containing whatever MQ objects or values we choose. Essentially the only changes required are to add a call to the function **fopen()** to open a file to write to. The fopen function returns a file identifier which is then passed to all subsequent fprint statements. Once writing to the file is complete the file is closed with a call to **fclose()**. Any file handles created will remain open until explicitly closed by a call to fclose() or until the **MQEV** program is ended. For a description of the formatting that is available please see the print statement description.

15.8.6 goto

The goto statement allows you to jump execution to a predefined label. Goto statements can only be used in functions.The syntax is:

```
goto <label name>
```

The label name can be up to 30 characters. It must start with an alphabetic character but this can followed by any alphanumeric character plus the '.' and '_' characters.

Many proponents of structured programming question the purity of using goto statements since they can make programs difficult to read and maintain. However, when used sparingly that can greatly simplify coding and actually improve readability. For example, one of the common uses it have a `goto` statement which branches to the end of a function in case of failure.

For example:

```
func foo(x)
  connect()
  if (_lastrc != 0) goto MOD_EXIT; endif
  … …
  … …
label MOD_EXIT;
  ;
endfunc
```

With a goto statement you can jump either forward or backward in a function. However you can not jump in to loops. You can jump out of a loop, but not jump in to one.

15.8.7 if(....) clause

The if(...) clause is the control language way of providing conditional execution.

There are two forms of the if(....) clause depending on whether an 'else' clause is required.

```
if( <Boolean expression> )
   <control statements>
endif


if( <Boolean expression> )
   <control statements>
else
   <control statements>
endif
```

The boolean expression can be any expression which results in a TRUE or FALSE result. The result is considered TRUE if the result is non-zero. The following are all valid if clauses:

```
if (curdepth > 100)              Depth of queue greater than some value
if (conname)                     Connection name is non-blank
if (!(queue == "SYSTEM.*"))      Not a system queue
if (exists(@option))             User Variable @option exists
if (findstr(descr,"test"))       Has "test" in the description
```

15.8.8 label

The label statement allows you to define a point that can be jumped to from elsewhere in the function. Labels can only be defined in functions.

The syntax is:

```
label <label name>
```

The label name can be up to 30 characters. It must start with an alphabetic character but this can followed by any alphanumeric character plus the '.' and '_' characters.

You can have as many labels in a function as you like but you should not define the same label in multiple places.

15.8.9 leave

The leave statement causes execution to leave the current file. Control is passed back to the calling code. So, imagine that you had a control file A which imported another control file B. If a leave statement was executed in control file B then the next statement executed would be the statement immediately following the the =import file(B) command in file A.

The leave statement is useful when detecting error conditions and can prevent a large build up of if-then-else type processing.

The leave statement can not be used in functions. If you wish to return prematurely from a function use the return statement.

15.8.10 print statement

The print statement is the general way in which you can output data.

The syntax is as follows:

```
print [:<format string>:] <print item> {,[:<format string>:] <print item> }
```

Essentially it is a comma separated list of items to be displayed. The items can be anything including strings, expressions or variables. Before each item you have the option to provide a format string which can provide some finer control of what is displayed.

The format string can contain:

Value	Meaning
Initial Number	A number immediately following the first ':' gives the minimum width of this item
'p'<number>	A 'p' followed by a number gives the precision of the real number
'l'	Item should be left aligned By default items are aligned according to their type, numerical values are right aligned, string values are left aligned.
'n'	No space before this item
'r'	Item should be right aligned By default items are aligned according to their type, numerical values are right aligned, string values are left aligned.
's'	If added to the last item on the print statement it will suppress the newline. Nothing will appear on the screen until a print statement which writes a newline is entered.

So, in the tradition that examples are the best way of learning, here are a few:

```
print "a","b","c"                - print out 3 items
print :10:"a",:5:"b",:20:"c"     - print out 3 items in columns
print :10:"a",:5r:"b",:20:"c"    - as above but right align second column
print "a","b",:n:"c"             - no spacing on last item
print 1/7                        - print 1/7 th
print :p6:1/7                    - print 1/7 th to 6 decimal places
```

By default each print statement will result in a new line of output. However, the new line can be suppressed using the 's' formatting flag. Equally a new line can be forced in the print statement by printing the system variable _nl. In addition a separator line can be printed by printing the system variable _sep.

All of the formatting available for the print statement is also available in the fprint statement.

15.8.11 return

The return statement allows either a value to be returned from a function or return from a function to be executed before reaching its end. The return keyword can be optionally followed by the value that should be returned.

The syntax of the statement is:

```
return [<expression>]
```

The expression can resolve to data type. For example, consider the following function:

```
func plus(a,b)
  return @a+@b;
endfunc
```

This function, useful really for only demonstration purposes, will return the sum of the two parameters. For example if both parameters are integers then it would return the integer sum. If both parameters are real numbers then it would return the real sum. Further if both parameters are strings then it will return the concatenation of the two strings. This behaviour demonstrates that the type of the parameters and return value are not defined but handled at run time. Each invocation of the function could use different data types.

A function will **always** return a value. If function execution reaches the end of the function without reaching a return statement then the integer 0 will be returned.

15.8.12 var

The var statement allows the programmer to force variables to be defined in the current stack frame. It can only be used in a function and, traditionally, would be defined at the start of the function although that isn't enforced.

The syntax of the statement is:

```
var <variable>{,<variable>}
```

You can therefore define as many variables as you need at the start of your function. For example,

```
func foo()
  var x, y, z
  … …
  … …
endfunc
```

Note that arrays are not supported by the var statement.

Although the variables are defined they have no value. Any attempt to use the variables without assigning a value would result in a runtime 'variable not defined' error. If required you can use the function ***exists()*** to check whether a variable has a value.

Please see Variable Scope and Stack Frames for further information.

15.8.13 wait() statement

The purpose of the wait() statement is merely to add a delay in processing.

The syntax of the command is:

```
wait(<Delay in seconds>)
```

You must be very careful issuing any form of wait in an MQEV function since it will hold up processing of other events. If you wish to wait for a certain amount of time for something to happen then you are usually better to use the mechanism described in Chapter 12.2.3 Script Reminder on page 154.

15.8.14 while(...) clause

The while(...) clause is the statement which is used to run a sequence of commands again and again.

```
while( <Boolean expression> )
   <control statements>
endwhile
```

The while(...) clause will loop continuously while ever the Boolean expression evaluates to `TRUE.` Clearly you need to be careful with the while() clause to ensure that the program doesn't loop forever unless that is what is required.

There are two other statements, break and continue, which can control the flow of the loop. In addition, if the while loop is in a function you can use return and goto to break out of the loop. Note that you can use goto to break out of a loop but you can not use goto to jump in to a loop.

15.9 Functions

Like most other programming languages **MQEV** supports the definition of user functions. Functions allow the definition of a group of statements which can be called from anywhere in the program by referring to the function name. Parameters can be passed to the function if required. The syntax of a function is as follows:

```
func FunctionName([Parameter [, Parameter])
   … …
   … …
endfunc
```

15.9.1 Function Basics

There is no restriction on how long a function can be or what statements it contains. Functions can call other functions and even call themselves (known as recursion). They also always return a value either explicitly using the **return** statement or an implicit zero. So an example might be:

```
func greeting(name)
  print "Hello",@name
endfunc

greeting("Mary");
```

Here we have a very simple function called 'greeting' being defined and we see a call to it from the main line program. The function takes a single parameter which it calls 'name'. The values passed from the function invocation are assigned to these parameter variables when the function is run. This would result in the output:

```
Hello Mary
```

If we try to pass too many parameters **MQEV** will report a syntax error explaining how many parameters are defined. However, **MQEV** will always allow us to pass less than the defined number of parameters. This can be useful if you want the function to accept different numbers of parameters, perhaps for qualification of some kind. If the function tries to reference a parameter which has not been passed in then it will generate a runtime error. If you want to check whether a parameter has been passed in you can use the **exists()** function.

For example, we could modify our function like this:

```
func greeting(name)
  if (exists(@name))
    print "Hello",@name;
  else
    print "Who are you?"
  endif
endfunc
```

Now if we call the function without passing in a parameter the function responds with:

```
Who are you?
```

Functions also allow you to return a value. So, in this example, we could decide to return the greeting and have the caller decide how to print it out. So, we would have something like this:

```
func greeting(name)
  if (exists(@name))
    return "Hello " + @name;
  else
    return "Who are you?"
  endif
endfunc

print ">",greeting("Mary")
```

Now if we call the function we get given a string back and we can print it out however we wish:

```
> Hello Mary
```

15.9.2 Function Invocation

Functions can be invoked either from an expression or from other functions or an imported command stream.

The only rule that should be followed is that the definition of the function must be registered before a call is made to the functions.

Calling a function from another function is perfectly straight forward however you must remember the rule above that the called function should be defined before the function that makes the call. However, there are a couple of special cases which we will discuss now.

Functions can be defined with up to 20 parameters. These parameters can accept any data type except arrays. An array can be defined in a function or it can be defined outside the function and referenced within it but you can't pass an array as an actual parameter.

15.9.2.1 Recursion

Occasionally it can be useful for a function to call itself, this is known as recursion. The classic example that is often used is to calculate factorial. (I'm not sure why since calculating a factorial would be far more efficient if done in a while loop.) However, it does demonstrate the principal fairly nicely.....

```
func fact(n)
  if (@n <= 1) return 1;
          else return @n * fact(@n-1)
  endif
endfunc
```

So, here we have a very simple function which returns the factorial of the number passed. It achieves this by calling itself if the number passed is greater than one. So, we can printout out the factorial of a number like this.

```
print fact(6)
```

Clearly one of the key things here is to ensure that the stack does unwind sooner or later. It is very easy to create infinitely recursive loops. When this happens, of course, the program will run out of stack and abnormally terminate.

15.9.2.2 Mutual Recursion

Mutual recursion is very similar to normal recursion but in this case we have two functions that each want to call each other. So, at first glance you might think that this would do the trick:

```
func flip(a)
  print "flip",@a
  if (@a > 0) flop(@a-1); endif
endfunc

func flop(a)
  print "flop",@a
  if (@a > 0) flip(@a-1); endif
endfunc
```

However, this breaks the golden rule that a function needs to be defined before it is used since flip() tries to call flop() before it has been defined. So, it would seem we are at an impasse. Well, not quite. Remember that you can define a function as many times as you like and each definition will overwrite the previous one. So, all we need to do is to make a minimal definition of flop() before we define flip(). It would look something like this :

```
func flop(a); endfunc

func flip(a)
  print "flip",@a
  if (@a > 0) flop(@a-1); endif
endfunc

func flop(a)
  print "flop",@a
  if (@a > 0) flip(@a-1); endif
endfunc
```

Now everything works just fine. You can consider the dummy definition of flop() as an indication to **MQEV** that there will be a function definition coming later. Some languages refer to this type of thing as a *forward definition*.

15.9.3 Dynamic Execution

Sometimes it is useful to define a function that operates largely the same on each invocation but its behaviour can be modified dynamically depending on the input parameters. For example in C one might pass in a function pointer that has the desired dynamic behaviour. **MQEV** does not have function pointers, it doesn't have pointers at all, However it can achieve a very similar effect using the '**eval**' function. The **eval** function is a very useful function, it will evaluate any given expression. This expression can be anything, including an invocation of a function.

Consider the following function:

```
**********************************************************************
*                                                                    *
* FUNCTION: WaitForState                                             *
*           Wait for a specific state to occur                       *
*                                                                    *
* - If you don't supply the number of iterations to loop round waiting*
*   for the channel to be running, then it will try it up to 10 times.*
*                                                                    *
**********************************************************************
func WaitForState(Command, Expression, Iterations)
  var rc; @rc = 0

  if (!exists(@Iterations)) @Iterations = 10; endif

  while (1)
    if (_idxWhile > @Iterations) @rc = -1; goto MOD_EXIT; endif
    wait(2)
    <@Command>
    if (eval(@Expression)) goto MOD_EXIT; endif
  endwhile
label MOD_EXIT
  return @rc
endfunc
```

This function uses a number of concepts but there are two things to notice.

- The use of substitution command to issue a command that is passed in
- The use of the eval() function to check the result of the command.

These two features means that this function is very dynamic. So, what would we use this function for? Well suppose we need a script that started a channel. We might well want to wait until the channel was running before continuing. We could now write something like this:

```
 START CHANNEL(<@ChannelName>)

 @rc = WaitForState("DISPLAY CHSTATUS(<@ChannelName>)", "_matches > 0")
```

It should be clear that the exact behaviour of the function can be changed enormously just by changing the Command and Expression parameters. The function can now be used for all sorts of things where you need to wait for some state in MQ to change.

Of course the concept of dynamic execution doesn't have to involve an MQ command, that is just an example of its use. The key thing is that a parameter can be passed to the **eval()** function. Since the **eval()** function will evaluate the expression, including calling functions, we now have a way of effectively passing a function pointer into a function.

15.9.4 Comments

In most of the examples throughout this manual we have not included comments. Ironically this is for clarity so you can concentrate on the actual instructions rather than any additional descriptions we may have provided. After all, the text in this document describes the code. However, when you are writing **MQEV** scripts we hope that you make liberal use of comments. In any programming language it is good practice to describe what is happening to make the code more maintainable. Think of the standard joke.....

Press Reporter: " What is the definition of bad programming? "
Programmer : " No comment "

However, consider the following:

```
* MQEV Example code written by Paul Clarke
*   taken from the MQEV manual

* Function : foo
* Purpose  : Demonstrates the passing of optional parameters
func foo(name)
  if (exists(@name))
    print "Hello",@name;
  else
    print "Who are you?"
  endif
endfunc
```

Clearly any comments defined between the **func** and **endfunc** statements belong to the function foo() but it is common coding practice to put the description of the function immediately before the function definition, rather than inside it. **MQEV** will associate immediately preceding comments as also part of the function.

The method used by **MQEV** is that all comments up to the second blank line will be include as part of the function definition.

```
  < Second blank line is function delimiter nothing prior to this included>
* Block comments included
* Block comments included

  < First set of optional blank lines included in function >

func foo(name)
  … …
endfunc
```

16 Debugging

As the script functions become more complicated it can be very useful to have some way of debugging your code. Perhaps the simplest form of debugging mechanism is trace. You could, for example, print out values to the screen or the log file when certain events happen. However, this can take a lot of time and is often unsatisfactory so **MQEV** also allows you to run your scripts in debug mode.

16.1 Debugger

There are two ways to go into debug mode, both of which require that you run **MQEV** in the foreground in a command window. If **MQEV** is already running then you can enter '!' on the status line. **MQEV** will then pause for instructions the next time any of the script is run. However, the normal way to enter debugging mode is to do it when **MQEV** is started. To run **MQEV** in debug mode you merely need to add a `-!` parameter such as the following:

```
mqev -!
```

This command will respond with something such as the following:

```
MQEV Version:9.1.0 (64 Bit) Build Date:Nov 22 2019

MQEV Initialising...
Loading MQSCX Script...
CMD:    1 > _error_func = "Error"
DBG>
```

There are two things to note here.

1. The line starting CMD: shows the current command we are about to execute
2. We have a prompt DBG> which is clearly waiting for us to do something

Of course your script might well have a different first line. You can ask to see a bit more of the script that is about to execute by entering the command 'l' (short for list). So, try entering 'l' at the prompt.

You should see that the program looks very similar to what we would see if we opened the ***mqev.mqx*** file in an editor. However, there can be differences so you should not assume that the format of the display is exactly the same as the input. We will discuss this soon but for now just see the list as a sequence of statements that **MQEV** is going to run. The keen eyed amongst you will have noticed two things:

1. **One of the lines has a > on it**
   This is the same line as we were shown initially. Showing, perhaps not surprisingly that this is still the current line and the next one that will be run.

2. **Only some of the lines are shown and the script file has many more lines**
   The debugger will only show the current code segment is it in. It will not, for example, show the contents of functions unless it is currently in that function. If you really wish to see the functions you can issue the command `l func` to see the functions defined and a command such as `l func MQEVEvent *` to display the contents of one of those functions.

So, I mentioned that the debug list format and the actual file format may not be the same. If you issued the 'l *' command you may already have seen it. The most noticeable example of this are multiple commands on a single lines. For example, suppose you had a line in your script such as the following:

```
if (_height)     @limit = _height-3; else @limit = 20; endif
```

This line actually contains multiple statements. If **MQEV** reported the line exactly as it is then it would make some things awkward[29]. For example consider the list output:

```
CMD:    9 > if (_height)     @limit = _height-3; else @limit = 20; endif
```

MQEV can tell us that it's about to run one of these statements but we wouldn't know which one. Later on we'll see that **MQEV** also allows us to set breakpoints and having multiple statements on a single line makes that awkward too. So, instead of listing the commands on a single line **MQEV** will split them into multiple lines like this:

```
CMD:    9   if (_height)
CMD:   10   @limit = _height-3
CMD:   11   else
CMD:   12   @limit = 20
CMD:   13   endif
```

This makes things much easier since each separate statement has a separate line. It is always clear which line **MQEV** is about to run and each statement can be identified uniquely for, say, setting breakpoints. In general you should not have to worry about this distinction, just bear in mind that the statement numbers you see are not line numbers in the file but rather just global statement number and serve only to allow you to easily identity a statement. Another clue to this behaviour is that comment and blank lines do not even get a line number. These lines are logically part of the following statement. In actual fact it is possible, since **MQEV** allows you to dynamically modify and include code, that the line number assigned to a particular statement could change during the running of the file.

However, the key thing you probably want to know is what commands you can issue at the debug command line. By all means try a few in the example we are currently running.

16.1.1 <Enter>

The the simplest 'command' you can enter is nothing, ie. Just press enter at the prompt. The debugger will run the current command, display any output and then pause on the next command. Repeatedly pressing enter will effectively step through the program one command at a time.

16.1.2 print

At any time you may be curious of the value of a variable or expression. You can issue a print command to have the values displayed. For example:

```
print @i, queue, _time, sqrt(100)*3
```

This example shows us printing a user variable, a response variable, a system variable and an expression. The debug print command will use the current debugger stack frame. For more information on this please refer to the 'sf' command on page 192.

29 Those of you familiar with other source level debuggers will be familiar with this problem. How many times have we had to change the program and recompile just so we can set a breakpoint on a multi-line statement ?

16.1.3 eval

The are times when you just want to see what the current command will evaluate to before you run it. If the current statement is a while() statement, an if () statement or an assignment then entering the command 'eval' will evaluate the current statement and display the result. For example:

```
CMD:   32     @value = 3+5
DBG>eval
Result:'8'
```

There may be times where you may wish to evaluate another expression altogether. In this case you can just follow the keyword 'eval' with the expression you wish to evaluate.

```
DBG>eval 6*7
Result:'42'
```

The expression can contain user variables, system variables and function calls if you wish. For example:

```
DBG>eval @a + @b
Result:'134'
```

Of course it is entirely possible that you have more than one variable @a in the program. If you are inside a function it is possible that there is both a local variable @a that belongs to the function and another variable @a that is in the main program. In fact, if you have a stack of function calls then there could be a myriad of @a variables. So, how does **MQEV** know which @a to use in the evaluation ?

Well, the debugger has a 'current' stack frame. A stack frame is what dictates the current scope of variables. Take a look at 'Variable Scope and Stack Frames' on page 170 for more information. By default the debugger will use the current stack frame for its evaluation. However, you can use the 'sf' command to change it to the stack frame you want.

16.1.4 Assignment

Sometimes it is useful to be able to change values of user variables. You can, therefore, enter an assignment directly at the debug line. You can either change existing variables or define new ones.

```
DBG>print @value
10
DBG>@value = 20
DBG>print @value
20
```

Assignment will use the current debugger stack frame. For more information on this please refer to the 'sf' command on page 192.

16.1.5 list (short-form 'l')

The list command will list the command source. There are a number of different variants of the command allowing the user to display different portions of the command list.

Command	Meaning
`list`	List a few source lines around the current position
`list 10`	List 10 lines around the current position
`list @3,10`	List 10 lines starting at statement 3
`list @3,@20`	List from statement 3 to statement 20
`list *`	List all source lines
`list +`	List the next few statements starting at the last list position
`list +,10`	List the next 10 lines starting at the last list position
`list -`	List the previous few statements starting at the last list position
`list -,10`	List the previous 10 lines starting at the last list position
`list func`	List out the defined functions
`list func test`	List out some of the function test
`list func test *`	List out all of function test
`list func test @3,@8`	List out function test from statement 3 to statement 8

Note that statements are not the same as lines. This is because blank lines and comments are not given statement numbers. The advantage of this is that the list output is far more readable since only actual statements are given numbers. It also means that when a statement line is output it's preceding comment line(s) are also output.

16.1.6 llist (short-form 'll')

The llist command accepts the same parameters as the list command, the only difference is that the llist command will output the file line numbers in addition to the global line number.

16.1.7 where

The where command will display the current command and it's line numbers and potentially some context about where in the set of command you are. For example, you could have the following response.

```
DBG>where
Line:     8 [ 0]+CMD:     3   foreach(DIS QL(*) CURDEPTH WHERE(CURDEPTH GT 0)
Line:    10 [ 0]+CMD:     5!>   @val[1,@i] = QUEUE
```

Notice that the file line number as well as the statement number is displayed. In this example you can see that we are currently on line 10 at statement 5. The `where` command also tells us the loops that we are currently in, so in this case we are processing the foreach loop on line 8. If we were inside an imported file the where command will also give us the name of the file and place at which the lines were imported.

The `where` command also shows us the stack frames associated with each statement (eg,**[0]**) and it shows us which stack frame is currently active. This is signified by the plus (+) sign immediately following the stack frame number.

16.1.8 Breakpoints

MQEV allows the user to set a breakpoint on any execution line. You may set up to 10 breakpoints at any one time. There are three commands which allow you to manage the breakpoints in the program.

16.1.8.1 bl

This command will list the current breakpoints. Lines with breakpoints will also be shown with an exclamation mark '!' next to the command when it is displayed using the list command.

16.1.8.2 bp

Set a break point. The command can be used a number of ways.

- To set a break point in the main line code
```
bp <Statement Number>
```
- To set a break point at start of a function
```
bp <function name>
```
- To set a break point at a statement in a function
```
bp <function name> <Statement Number>
```

If you attempt to set a breakpoint on an invalid line **MQEV** will try to set a breakpoint on the next execution line. At any one time there can be up to 10 breakpoints set in the program.

16.1.8.3 bc

Clear break point. The command can be used a number of ways.

- To clear a break point in the main line code
```
bc <Statement Number>
```
- To clear the break point at start of a function
```
bc <function name>
```
- To clear a break point at a statement in a function
```
bc <function name> <Statement Number>
```

Note that a command line may not have the same statement number for the entire life of the program. As files are imported or command inserts occur the exact position of a line may change. Consequently it is entirely possible that to clear a breakpoint you must enter a different statement number than was used to set the breakpoint.

16.1.9 end

The end command can be used to terminate the command file. The **MQEV** program will end without running any further commands.

16.1.10 run

The run command instructs **MQEV** to run the commands either until the next breakpoint or until the end. If there are no current breakpoints then debug mode is essentially ended and **MQEV** will go back to the status line display. You can re-enter debug mode by entering '!' on the status line. **MQEV** will then stop at the next script line that is about to be run.

16.1.11 runout

The runout command is essentially a shorthand way of setting a breakpoint after the end of the current loop, issuing the command run, and then clearing the breakpoint. It is useful if you have confirmed the processing of one iteration of the loop and want to continue debugging on the statements following the loop.

16.1.12 sf

By default the current stack frame is used for evaluations but this command can be used to override it temporarily. Please see 'Variable Scope and Stack Frames' on page 170 for a description of stack frames.

The syntax of the command is:

```
sf <stack frame number>
```

Stack frames are numbered consecutively from 0 depending on how many levels of stack you are in. For example, if we were debugging the factorial function described in the section on 'Recursion' on page 183 we could issue the where command and be shown something like this.

```
DBG>where
Line:    7 [ 0] CMD:    1   print fact(6)
Line:    3 [ 1] FNC:    5             return @n * fact(@n-1)
Line:    3 [ 2] FNC:    5             return @n * fact(@n-1)
Line:    3 [ 3] FNC:    5             return @n * fact(@n-1)
Line:    3 [ 4] FNC:    5             return @n * fact(@n-1)
Line:    1 [ 5]+FNC:    1 > func fact(n)
```

The where command shows where program execution is right now. It essentially shows the stack. In the example above you can see that we are nested a few levels deep in the calls to function fact(). The where command output shows the stack frame after the line number as something like **[2]**. You can see that as each function call is made the stack frame number increases. The stack frame the debugger is currently using is signified by it being followed by a plus '**+**' sign. So, if we issued a debug command like **print @a** we would see the value of @a in stack frame 5.

```
DBG>print @n
2
```

However, suppose we are interested in the value of @a at the first invocation. That is where the **sf** command comes in. Now we enter the sequence.

```
DBG>sf 1
DBG>print @n
6
```

If we now issued the where command we would see

```
DBG>where
Line:    7 [ 0] CMD:    1   print fact(6)
Line:    3 [ 1]+FNC:    5             return @n * fact(@n-1)
Line:    3 [ 2] FNC:    5             return @n * fact(@n-1)
Line:    3 [ 3] FNC:    5             return @n * fact(@n-1)
Line:    3 [ 4] FNC:    5             return @n * fact(@n-1)
Line:    1 [ 5] FNC:    1 > func fact(n)
```

You can see that the plus (+) sign now indicates that the debugger stack frame is now set the 1. Note that the current command is still the same as it always was and execution of the program is not affected by the **sf** command. The **sf** command only affects commands entered in the debugger such as eval, print and assigning values to variables. As soon as you execute any of the actual program lines the stack frames reverts back to the actual stack frame in use by the program.

16.1.13 Help (short-form ?)

The help command provide a quick memory jogger of the debug commands which are available. There are two forms of the command, either just on it's own or with the command you want help on.

```
help
```

The command issued on its own will just show a simple list of the commands available. If you optionally follow the word 'help' (or indeed ?) with the name a command then a description of that command will be shown. For example:

```
help list
```

16.1.14 Command alteration

The debugger does not currently allow you either insert or delete command in the command stream. However, there is an occasion where you can change the commands run. Consider the following short program:

```
CMD:    1   @mycmds = "print 3*4"
CMD:    2   @mycmds
```

In this case **MQEV** will run whatever commands are contained in the user variable @mycmds. It follows therefore that you set a breakpoint on line 2 and then change the values of @mycmds you could essentially insert commands into the command stream.

17 Data Management

As **MQEV** receives the event data it will compress the data (please see 'Compression' on page 7) but the data still has to go somewhere and so we need to consider that and have ways of dealing with it. First of all, how much data are we talking about? Well, this is a hard question to answer since the size and frequency of event data will vary enormously by installation. A reasonable average message size to assume, once compressed, is 300 bytes. Let's say a queue manager generates about 5,000 events a day[30]. That would mean we need to store about 1.5 MB per day. Let's say we wish to store this data for six months that would mean we need to store around 270 MB of information. By modern standards that is a rather small amount of data and it is likely our estimates are on the high side. However, this is only a rough guide, there could be many reasons why the actual number is different:

- **What events are we choosing to store?**
  5,000 events a day may seem like a lot but it depends on what events you are choosing to store. For example, suppose you choose to store command events. Furthermore suppose you choose to store `DISPLAY` commands. All of a sudden 5,000 is not looking quite so big, especially if you have some of those monitoring products who are regularly issuing various display commands. By default `DISPLAY` commands are not stored since they are pretty innocuous. However, if you do choose to store them then consider whether you really need to store them for very long. It may be worth considering sending `DISPLAY` commands to their own stream with small retention interval.

- **We might want to store the data for multiple queue managers**
  MQEV is quite capable of receiving events from any number of queue managers. All you have to do is 'point' your event queues at a queue that **MQEV** is reading and it will store the events, neatly separated by queue manager name.

  Generally speaking this is not recommended since it means that in order for **MQEV** to receive events you need to have networks and channels running etc. This could impinge on the reliability of your event processing. However, in some circumstances it may be preferable to run one central **MQEV** rather than have a set of separate processes. However, there is a limit to this behaviour. You could not, for example, expect one central **MQEV** process to handle all the events from, say, a thousand different queue managers. Like anything else, you need to be sensible in your topology.

- **We might decide to store the data for a lot longer**
  The default retention interval for streams is 90 days. This means that events will be discarded after about 3 months. In our example above we have extended this to 6 months but suppose you decided to keep your event data for 5 years. Well, this can have a significant effect on the amount of data you are storing. If you really choose to do this then you may wish to consider which events are really important. For example, as we discussed above, storing `DISPLAY` commands for 5 years seems unnecessary.

- **We may get occasional floods of events**
  It is possible to get a flood of events, for example when a network goes down. Or perhaps if you have a misbehaved application. Hopefully these are rare events but some installations have to cope with dodgy networks and many more have to contend with dodgy programmers! So how do we minimise this impact. The answer is 'Event Storms'. You can configure **MQEV** to combine lots of similar events received in a short interval into a single event.

- **What types of data do you wish to store**
  So far we have just considered MQ Events. However, **MQEV** can also store your statistics and accounting messages. Statistics messages are fairly well controlled in that you get a fairly predictable arrival rate of the messages and you know how many objects you have so you can fairly easily work out the maximum number of messages of this type you might get. However, Accounting messages are more tricky. If you have very short lived Applications then you may get a high rate of accounting messages – at least one message for each short lived connection. Again it comes down to how long is this data relevant. It can be extremely useful to be able to look at this data over the period of say a week or even a month but you need to consider whether it is really worth keeping the data much longer than that.

30 We are trying to estimate on the high side

If all else fails we may have to manually discard events. This could be, perhaps, because we have received some erroneous data which is has created a stream which we shouldn't have. The command we would issue would be:

```
PURGE EVSTRM(BADSTREAM) [ EVQMGR(MYQM) ]
```

This will discard all data from a stream. If required you can qualify the stream by queue manager so that only data from certain queue manager(s) are discarded.

Remember that the script functions allow you to separate the data into as many streams as you wish. Each stream can be given a different retention interval. So, it should be fairly straightforward to configure **MQEV** to store only the relevant data for long periods of time.

18 Operational Characteristics

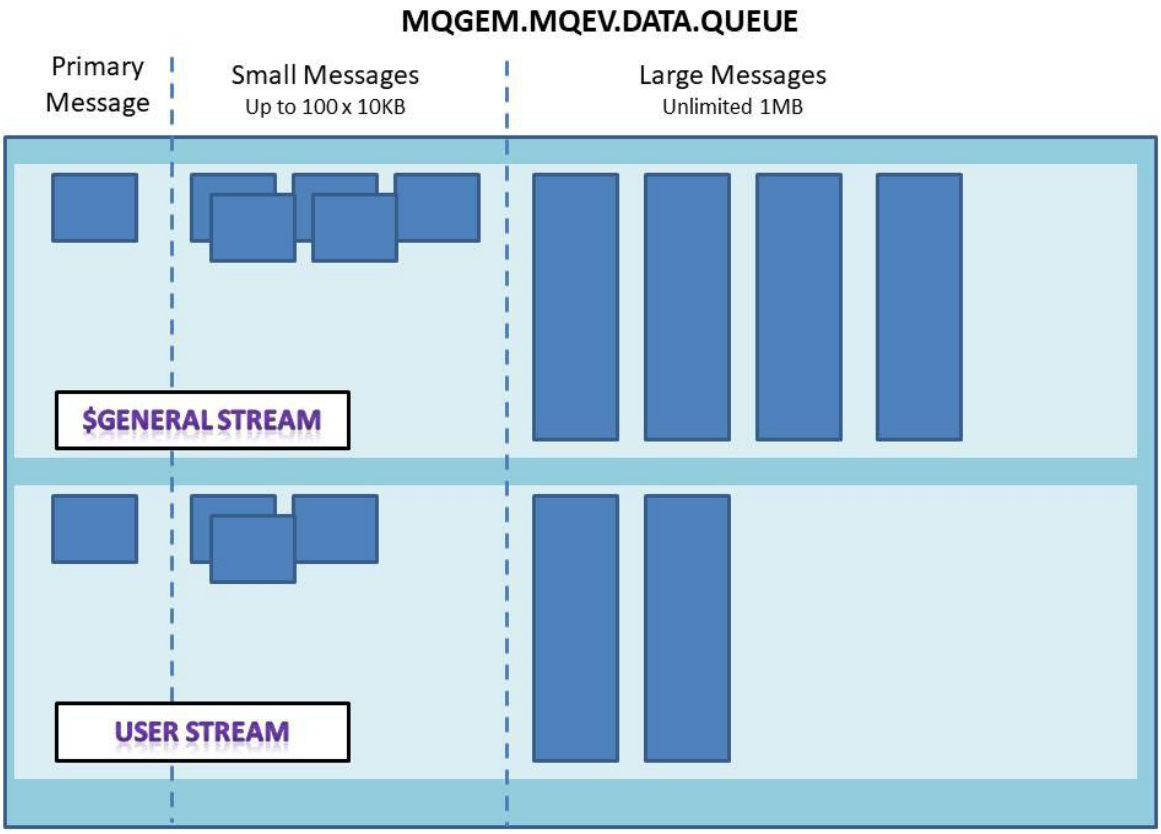
18.1 Message Consolidation

So far we have mentioned in general terms that the event data is stored on the `MQGEM.MQEV.DATA.QUEUE` but very little information about how that is done. Generally speaking, of course, the user of **MQEV** does not need to know. However, the curious amongst you may be looking at the queue and wondering what is going on so here is a brief explanation of how the data is stored.

Event data is stored in unique streams. Each stream has a key which is 'Queue Manager Name' and 'Stream Name' and Stream Type. Events for different queue managers, different streams and/or different stream types will therefore never exist in the same MQ message. When events are placed into MQ messages it is done in three stages. There are, therefore, three types of MQ message for each stream.

- **Primary Message**
  The primary message of each stream contains the stream header and the latest (most recent) set of event data. The primary message is up to 10K bytes. There can be at most one of these.
- **Small Message**
  When the primary message becomes full it is copied to a 'small message'. A small message is up to 10K bytes. There can be up to 100 of these.
- **Large Message**
  When we have 100 small messages they are consolidated into a single large message. Large messages can be up to 1MB. There is no limit on the number of possible large messages.

We could view this pictorially like this:



18.2 Message Retention

The event messages are stored in 'streams'. Each stream has a configurable retention interval which defaults to either 90 or 45 days depending on the type of record stored on the stream (events, accounting or statistics). This can be increased or reduced as required. However, the intention is that all streams should have a finite retention period. Beyond this point the data is considered irrelevant or at least there is little point storing it in a direct access program like **MQEV.** Instead, if you really do wish to keep this data indefinitely then you should consider archiving the event data to another queue, or possibly offloaded to a file. If you plan to do this then you may wish to store the original data as explained below.

18.3 Time zones

Accounting and Statistics messages contain data within them which records dates and times. Specifically, every accounting and statistics message contains four fields, ***Interval Start Date/Time*** and ***Interval End Date/Time***. These fields are reported in the queue manager's local time. This means that if you move these messages to a machine in a different time zone, the context of the time zone of these fields will be lost. It is therefore inadvisable to funnel accounting and statistics to a central collection queue manager. This funnelling pattern is useful and common with event messages, but not suitable for accounting and statistics.

As a result of this it is also essential that the MQEV program runs on a machine in the same time- one as the queue manager if it is processing accounting and statistics messages. If MQEV is running with a bindings connection to the queue manager then all is well. However, if it is running as a client connecting to the queue manager on another machine, then the client machine and the queue manager machine need to be on the same time zone in order for the dates and times inside accounting and statistics messages to be correctly interpreted.

18.4 Original Event Data

Because of the mechanisms used in the message compression employed by **MQEV** it is not possible to get back to exactly the same data that was received from IBM MQ. **MQEV** will store the relevant fields but will not store everything. For example, no attempt is made to store the **MQMD** which came along with the event data. If you have need to store the event data long term then you may wish to consider using the daisy chaining mechanism in the `EVQ` objects to forward the event data to another queue which can then either be archived or stored to a file. This way you know that you have exactly the data that was issued by IBM MQ.

18.4.1 Daisy chaining

Daisy chaining is a pattern whereby messages consumed by an application are first forwarded to another application's input queue, unchanged, prior to being processed. It is a common pattern when consuming event messages, since originally IBM MQ only produced one event message for any particular notification. In modern versions of IBM MQ it is of course possible to have your event messages published to multiple queues, but none-the-less, daisy chaining is still a useful feature.

To use daisy chaining for any particular queue that is being processed by **MQEV**, set the **`FWDQ`** attribute to the name of the queue where a copy of the message should be sent. Event messages are created by the queue manager using default persistence, and you can therefore control the persistence of event messages by altering the **`DEFPSIST`** attribute of an IBM MQ event queue definition. You can also independently control the persistence of the forwarded copies of these messages using the **`FWDPSIST`** attribute. The **`FWDQ`** and **`FWDPSIST`** attributes can be found on the **`ADD EVQ`** and **MQEV** commands.

18.5 IBM MQ Configuration

Clearly, as an IBM MQ application, there are certain MQ settings which are needed in order for **MQEV** to operate successfully. We have already discussed that **MQEV** needs the two queues, `MQGEM.MQEV.DATA.QUEUE`[31] and `MQGEM.MQEV.COMMAND.QUEUE` and considered their attributes so we will not repeat ourselves.

Here we concern ourselves more with the queue manager properties of IBM MQ.

Attribute	Description	Recommended	Minimum
`MAXUMSGS`	Queue Manager attribute which controls the maximum size of a transaction.	10,000	200
`MAXHANDS`	Queue Manager attribute which controls how many queues can be opened by a process. Must be sufficient to open all the events queues plus the two standard queues.	256	As required
`MAXMSGL`	Queue Manager attribute controlling how large messages can be.	4,194,304	2,097,152
`*EV`	Event attributes enabled accordingly	ENABLED	N/A

31 Or a Persistence Queue of the name of the monitored queue manager if you are using a State Queue Manager.

19 Security

MQEV is an MQI application, it is written in 'C' and uses the standard MQI to interface with IBM MQ. It does not use any private interfaces into IBM MQ. As such it is subject to exactly the same security mechanisms as any other MQ program.

19.1 Authorities needed by the MQEV program

One mode of running **MQEV** is to run as a privileged user, for example if you run it as a `SERVICE.` However you you may choose to run **MQEV** under a non-mqm user id, in which case at least the following authorities will be required.

In addition to the authorities required on the various queues, we recommend creating a topic object as follows in order to set authorisations for the specific topic string used by **MQEV**.

```
DEFINE TOPIC('MQGEM.MQEV.ALERTS') TOPICSTR('MQGem/MQEV/Alerts')
DESCR('Topic Object for MQEV Alert publications')
```

In addition to the commands shown below, if you intend to issue IBM MQ commands from inside the `mqev.mqx` script functions, authorities to allow **MQEV** to issue those commands will be needed. Also, if you allow **MQEV** to use a Dead-letter queue, authorisation to put to that queue will be required. This can be the queue manager's central DLQ or an **MQEV** specific DLQ - see ALTER EV on page 57 for details.

19.1.1 Example security commands for Distributed Platforms

It is assumed in the following examples that the user id under which the **MQEV** program is running is in a group named `mqevapp`.

```
SET AUTHREC                                      OBJTYPE(QMGR)  GROUP('mqevapp') AUTHADD(CONNECT,INQ,DSP,ALTUSR)
SET AUTHREC PROFILE('MQGEM.MQEV.COMMAND.QUEUE')  OBJTYPE(QUEUE) GROUP('mqevapp') AUTHADD(GET,INQ,PUT)
SET AUTHREC PROFILE('MQGEM.MQEV.DATA.QUEUE')     OBJTYPE(QUEUE) GROUP('mqevapp') AUTHADD(GET,BROWSE,PUT,INQ)
SET AUTHREC PROFILE('SYSTEM.ADMIN.COMMAND.QUEUE') OBJTYPE(QUEUE) GROUP('mqevapp') AUTHADD(PUT)
SET AUTHREC PROFILE('SYSTEM.DEFAULT.MODEL.QUEUE') OBJTYPE(QUEUE) GROUP('mqevapp') AUTHADD(GET,DSP,PUT)
SET AUTHREC PROFILE('MQGEM.MQEV.ALERTS')         OBJTYPE(TOPIC) GROUP('mqevapp') AUTHADD(PUB)
```

In addition to the above, we also need to grant **MQEV** access to each of the event, accounting and statistics queues. One example is shown below, but all queues to be monitored will need to have the same access.

```
SET AUTHREC PROFILE('SYSTEM.ADMIN.QMGR.EVENT') OBJTYPE(QUEUE) GROUP('mqevapp') AUTHADD(GET,INQ)
```

MQEV needs authority to put the responses to commands it is asked by applications such as **MQSCX** and **MO71**. If you use a different `MODEL` queue for these tools, i.e. not `SYSTEM.DEFAULT.MODEL.QUEUE`, then the following authorities will be required.

```
SET AUTHREC PROFILE('MQSCX.**') OBJTYPE(QUEUE) GROUP('mqevapp') AUTHADD(PUT)
SET AUTHREC PROFILE('MQMON.**') OBJTYPE(QUEUE) GROUP('mqevapp') AUTHADD(PUT)
```

19.1.2 Example security commands for z/OS (using RACF)

It is assumed in the following examples, that the user id under which the **MQEV** program is running is in a group named `MQEVAPP`. It is also assumed that all the classes in use below are active on the system through the necessary switch profiles, and that the profiles used have already been **`RDEFINE`**-d.

```
PERMIT qmgr.BATCH                        CLASS(MQCONN)  ID(MQEVAPP) ACCESS(READ)
PERMIT qmgr.ALTERNATE.USER.**            CLASS(MQADMIN) ID(MQEVAPP) ACCESS(UPDATE)
PERMIT qmgr.MQGEM.MQEV.COMMAND.QUEUE     CLASS(MQQUEUE) ID(MQEVAPP) ACCESS(UPDATE)
PERMIT qmgr.MQGEM.MQEV.DATA.QUEUE        CLASS(MQQUEUE) ID(MQEVAPP) ACCESS(UPDATE)
PERMIT qmgr.SYSTEM.ADMIN.COMMAND.QUEUE   CLASS(MQQUEUE) ID(MQEVAPP) ACCESS(UPDATE)
PERMIT qmgr.SYSTEM.DEFAULT.MODEL.QUEUE   CLASS(MQQUEUE) ID(MQEVAPP) ACCESS(UPDATE)
PERMIT qmgr.MQEV.**                      CLASS(MQQUEUE) ID(MQEVAPP) ACCESS(UPDATE)
PERMIT qmgr.PUBLISH.MQGEM.MQEV.ALERTS    CLASS(MXTOPIC) ID(MQEVAPP) ACCESS(UPDATE)
```

In addition to the above, we also need to grant **MQEV** access to each of the event queues. One example is shown below, but all queues to be monitored will need to have the same access.

```
PERMIT qmgr.SYSTEM.ADMIN.QMGR.EVENT CLASS(MQQUEUE) ID(MQEVAPP) ACCESS(UPDATE)
```

MQEV needs authority to put the responses to commands it is asked by applications such as **MQSCX** and **MO71**. The following authorities will be required.

```
PERMIT qmgr.MQSCX.**                CLASS(MQQUEUE) ID(MQEVAPP) ACCESS(UPDATE)
PERMIT qmgr.MQMON.**                CLASS(MQQUEUE) ID(MQEVAPP) ACCESS(UPDATE)
```

19.2 Authorities needed by users of MQEV

For issuing **MQEV** commands, there are two classes of user.

MQEV Users	These users can issue any **MQEV** `DISPLAY` command. This is implicit. If the user has the ability to put to the `MQGEM.MQEV.COMMAND.QUEUE` command queue it is assumed, by **MQEV**, that they are entitled to issue a `DISPLAY` command.
MQEV Administrators	These users can issue any **MQEV** command. If a `non-DISPLAY` command is issued then **MQEV** will check that the user has `MQSET` authority on the `MQGEM.MQEV.COMMAND.QUEUE.` If this check fails then the command will fail with a security error.

Users who need to issue **MQEV** commands can use tools such as **MQSCX** or **MO71**. These tools require some minimum authorities in order to connect, put a command and get a reply. See the specific user guide for full details.

Users who need to issue **MQEV** commands should, in addition to the above, be given `PUT` access to the `MQGEM.MQEV.COMMAND.QUEUE` queue as shown in the examples that follow.

For full access to all MQEV commands, not just DISPLAY commands, users who have need to use these commands, for example to configure MQEV, will need additional authority as shown in the examples that follow.

19.2.1 Example security commands for Distributed Platforms

In the command that follows, it is assumed that the user ids which require access to issue **MQEV** `DISPLAY` commands are in a group named `mqevusers`.

```
SET AUTHREC PROFILE('MQGEM.MQEV.COMMAND.QUEUE') OBJTYPE(QUEUE) GROUP('mqevusers') AUTHADD(PUT)
```

In the command that follows, it is assumed that the user ids which require access to configure **MQEV** with non-`DISPLAY` commands are in the `mqevusers` group above and additionally in the `mqevadmins` group used below.

```
SET AUTHREC PROFILE('MQGEM.MQEV.COMMAND.QUEUE') OBJTYPE(QUEUE) GROUP('mqevadmins') AUTHADD(SET)
```

19.2.2 Example security commands for 19.1.2 z/OS (using RACF)

In the command that follows, it is assumed that the user ids which require access to issue **MQEV** `DISPLAY` commands are in a group named `MQEVUSR`.

```
PERMIT qmgr.MQGEM.MQEV.COMMAND.QUEUE   CLASS(MQQUEUE) ID(MQEVUSR) ACCESS(UPDATE)
```

In the command that follows, it is assumed that the user ids which require access to configure **MQEV** with non-`DISPLAY` commands are in the `MQEVUSR` group above and additionally in the `MQEVADM` group used below.

```
PERMIT qmgr.MQGEM.MQEV.COMMAND.QUEUE   CLASS(MQQUEUE) ID(MQEVADM) ACCESS(ALTER)
```

20 Trouble Shooting

It is never easy to write a trouble shooting section since there are clearly many things that can go wrong. In our experience the vast majority of the time it is a configuration error. The tricky bit is finding it. If you find that **MQEV** is not collecting or displaying your events then re-check the configuration and parameters you are passing. There are some simple debugging flags that may help. Starting MQEV with something like:

```
mqev -vpP
```

will cause it print out a summary of the event messages it is receiving. Other verbose options can help in other circumstances.

If your situation is not covered in the FAQ then please do raise a question with Support, details of how to do this are given in 20.2 Support on page 204.

20.1 Frequently Asked Questions

20.1.1 My MQEV runs in the background, how do I know what it's doing?

MQEV is designed to run in the background. There are a number of status commands to show how many messages it has processed, and the resources (memory and data queue messages) are in use. It will also output its activity to a log file which by default is found in the same directory as the program, but which can be directed to an appropriate location by using the `-L` flag when running the **MQEV** program. See Chapter 4 Parameters on page 30.

20.1.2 Why does MQEV not delete all my log files?

MQEV can be configured with a retention interval for the log files generated during the life of a particular **MQEV** process. However, **MQEV** will not delete the files generated by other **MQEV** processes. This is to allow the user to inspect the contents of those files, for example in the case where the program ends unexpectedly.

It is recommended that you start **MQEV** using a script file that initially deletes all the ***MQEVLog*.txt*** files in the log directory, so that you start with a clean slate on each run (except in the case where you have something to look into of course!).

20.1.3 Why does MQSCX complain that my commands are invalid?

Remember that you need to be in `=mqev` mode in order to send commands to the **MQEV** command server, and in `=mqsc` mode to send commands to IBM MQ. If you regularly forget which mode you are in, consider updating your **MQSCX** prompt to make it clear.

20.1.4 Why do the MQEV menus not appear in MO71?

This would be most likely because **MO71** has not yet seen that your queue manager has a queue called `MQGEM.MQEV.COMMAND.QUEUE`. Refreshing the objects for the queue manager in question should resolve this problem. Alternatively, explicitly tick the “MQEV commands” check box in the Options menu on the location dialog.

20.1.5 Why does my alert disappear?

When creating an alert you can specify a retention interval. If you do not specify one, the default value from the EV object will be used. If you do specify one, remember that the value you provide is measured in seconds. This is described further in 12.3 Alert Retention on page 156.

20.1.6 Why can't I see the events I know have been generated?

Check the following:-

- Is the event queue being processed by **MQEV**? Use the **`RESUME EVQ`** command to ensure it is.
- Was the event more than 24 hours ago? By default the **`DISPLAY`** commands will only show the last 24 hours worth of activity. To display older data than this, add a parameter such as **`FROM(-5days)`** to the command.
- Ensure that **MQEV** is running in the same time zone as the queue manager generating the events.
- If the event you can't see is a command event recording a **`DISPLAY`** command, check the value of the **`DISPCMDS`** attribute on the **`EV`** definition.
- Ensure that the event in question is not being discarded by your scripts - by setting `_stream = "$null"`

20.1.7 Why don't my scripts work?

Check the following:-

- Accessing accounting and statistics data require an association variable, for example `data.PUTNP`, rather than just `PUTNP.` If you specify just `PUTNP` alone, this will be interpreted as a response variable and will therefore have the value null.
- Remember that you can run in debug mode by running the **MQEV** program with the **`-!`** flag. This allows you to step through the code and display variables at each line of your script. See Chapter 16 Debugging on page 187.

20.1.8 Why does MQEV complain that there are functions missing from my script?

While it is not necessary to have any code in all the functions, **MQEV** does require you to have all the functions present in your script file. This mechanism ensures that you have not accidentally misspelt the name of the function and are unaware that it isn't being called. The provided sample ***mqev.mqx*** is a good place to start when writing your scripts.

20.1.9 Why does my MQEV on z/OS complain that no licence is found

An **MQEV** licence file for the distributed platforms will not enable an **MQEV** on z/OS to run. There is a separate licence for **MQEV** on z/OS. It will have a product name of “MQEVz”.

20.1.10 Why can't I view my MQEV on z/OS log files in their PDSE while MQEV is running

If you specify the log path using the **`-L`** program parameter and directly specify the PDSE name rather than using a DD name, there is nowhere to supply a disposition of shared, so the underlying file operations run without it. Use the **`MQEVLOG`** DD name and ensure to code `DISP=SHR` and you will be able to view old log files in the PDSE while the **MQEV** program is running.

20.1.11 Why doesn't my script wait until my system call is finished before continuing

The `system()` call can be run in two modes, synchronous and asynchronous. If you omit the second parameter on the `system()` call, it will run in asynchronous mode (to retain compatibility with prior versions). If you want your script to wait until the `system()` call is finished before continuing, ensure you code `const.SYNC` in the second parameter of the `system()` call.

20.1.12 What does “Responses limited as requested” mean ?

A number of commands for **MQEV**, such as `DISPLAY` EVENTS, will limit the number of responses they will return to avoid flooding the requester with answers. By default each command will return up to 100 responses. This is done for a number of reasons not least of which that returning large numbers of answers is both costly in terms of processing power but it also makes it hard for the command issuer to 'see the wood from the trees'. It encourages the user to use filtering such as date range, object name, userid etc to limit the number of responses. Of course there are times when you want all possible answers, for example you want a history of all events or you are doing some form of export. In those cases the response limit can be overridden using the `MAXRESP` attribute on the command.

20.1.13 What does “Source records limited as requested” mean ?

Some of the commands for **MQEV**, such as the display commands for Events, Accounting and Statistics, will limit the number of records used from the repository to generate the response. This is done for a variety of reasons. First of all it prevents accidental consumption of large amounts of CPU. An **MQEV** repository can contain millions of records and you might not want to process all of them. Secondly it allows **MQEV** to process the data more efficiently since it knows the limit of what is required.

Note that `MAXRESP` and `MAXRECS` are related but nonetheless quite different. It is entirely possible, for example, for a command to use a large `MAXRECS` value and yet only return a single answer. Consider the command:

```
DISPLAY ACCT(*) SUM(TOTAL) ALL
```

This command may well have thousands of source records from which to draw the response (`MAXRECS`) but will only actually send a single reply (`MAXRESP`) which is the sum total of all the records it found.

The default `MAXRECS` value is 1,000,000 which should be sufficient for the vast majority of commands. It is entirely possible that, even in fairly large installations, the repository never actually contains as many as 1,000,00 records. Clearly it depends on the frequency of data collection and the stream retention interval. Consider an installation which is creating 5,000 events a day (a considerable number) and retains the events themselves for 6 months (a considerable retention period) this would still be 'only' 900,000 records – below the 1,000,000 default. However, this is one of those 'one size probably doesn't fit all' type situations and, as such, you can change the default if you wish by changing `DEFMAXRECS` in the `EV` objects.

20.1.14 I saw “IBM MQ QMgr is generating bad data in STATCHL messages” in my MQEV log.

A defect introduced in IBM MQ V9.2.5 resulted in the queue manager field of a Channel Statistics message containing rubbish data instead of the queue manager name. This rubbish data was also often not even printable characters. MQEV checks for this bad data and if it is found that the queue manager name is not correct, reports the message “IBM MQ QMgr is generating bad data in STATCHL messages” on the log once for each run of MQEV where it is seen. MQEV also corrects the queue manager name in the data it saves.

We hope that IBM will soon produce an APAR for this IBM MQ defect, but in the meantime, the data collected by MQEV should not contain this troublesome bad data.

This check and correction of the bad data received from IBM MQ can be turned off by setting the environment variable MQEV_NO_SCQM_FIX. We do not recommend doing this unless the APAR fixing the issue is applied.

20.2 Support

We am sorry you are having problems and need support. The first thing you should always do is just to check that you are using the latest version of the program. Please go to our web site (MQEV) and check the latest build date. If there really is a problem with the software then there is always a good chance that someone else has found the problem before you have. So, you can save yourself a lot of time and effort if you make sure you are always on the latest level of maintenance.

If you can still reproduce the problem on the latest version then please feel free to email us the details of the problem and we will do our best to help you. Remember that the more complete you make the description of the problem the better chance we have of solving it. The kind of information you should include in your email is:

- Your full name
- The exact version, including build date, of the **MQEV** (and **MQSCX** or **MO71** as well if applicable) you are using.
- The email address and issue date from within your licence file (if you have one)
- The OS platform and version you are using
- A complete description of the problem and how to recreate it. Please include as much detail as possible such as frequency of occurrence. For example, does the problem happen every time or just occasionally? Are there any error messages produced by **MQEV** or IBM MQ at the time of the problem ?

Once you have gathered this information please email it to support@mqgem.com. We shall reply as soon as possible. Note that priority will be given to customers based on the severity of the problem and the type of licence held.

21 Changes made in previous versions

This chapter will give you an idea of the changes that have been made if you have used a previous version.

21.1 Changes made in Version 9.3.0

1. **Support for IBM MQ Command Levels up to 932**
   This includes changes to command and configuration events as a result of object changes, and also the enhancement of Queue Accounting records to include the connection name of a client connected application. This allows `SUM(CONNAME)` to be used on `DISPLAY ACCTQ`.

2. **Protection against defective data in Channel Statistics records**
   A defect introduced into the IBM MQ Queue Manager in V9.2.5 means that Channel Statistics records have some rubbish data in the queue manager field of the record. **MQEV** detects this issue and works around it. See 20.1.14 I saw "IBM MQ QMgr is generating bad data in STATCHL messages" in my MQEV log. on page 203 for more details.

3. **Support for emit format NDJSON**
   Newline Delimited JSON (NDJSON) is suitable for situations where you have a downstream program reading the emitted file at the same time as MQEV is writing to it.

21.2 Changes made in Version 9.2.2

1. **IBM MQ Command Levels up to 924 supported.**

2. **Addition of emitters**
   You can request MQEV emit events, accounting and statistics messages as JSON, CSV or MQSC messages on a queue, or files. This can be useful if you wish to push events to a centralised store such as Elastic or Splunk. For more information please see Chapter 6 Emitters on page 35.

3. **`BUILD` added to `DISPLAY EV` command**
   MQEV will return the date of the program build.

4. **Performance improvements**

   - The speed of accessing Event, Accounting and Statistics data has been improved when accessing large numbers of records. In addition `MAXRECS` control added to the `DISPLAY` commands

   - Housekeeping of strings has been improved to reduce storage usage.

5. **Add `DEFMAXREC` to the EV object**
   Users can set a default 'maximum number of records to search' on the EV object.
   By default this has a value of 1,000,000.

6. **Minor changes to the PCF Groups contained in a response message to the PCF equivalent of a `DISPLAY EVENTS` command showing a configuration event.**
   See 22.1 Migrating from a version prior to Version 9.2.2 on page 207 for more details if you have your own PCF application issuing commands to **MQEV**.

7. **Support added for hexadecimal numbers in expressions and WHERE clause**
   MQSCX expression can now contain numbers such as 0xAB43D

8. **Support added for hexadecimal strings in expressions and WHERE clause.**
   For example:
   ```
   DISPLAY ACCTMQI(*) WHERE(CONNID EQ 0x'414D51434D5147322020202020202010CAE36001CCCD23') SUM(NONE) ALL
   ```

21.3 Changes made in Version 9.2.1

1. **IBM MQ Command Levels up to 921 supported.**
2. **Addition of CMDLEVEL to Accounting and Statistics commands**
   This will return the IBM MQ Command level of the Queue Manager at the time the record was written.
3. **Addition of CHANNEL to DISPLAY ACCTQ command**
   IBM MQ can now report **`CHANNEL`** as part of the Accounting Queue data. This is now stored and can be displayed in the normal way. Records can also be summed by channel name by using **`SUM(CHANNEL)`**

21.4 Changes made in Version 9.2.0

1. **The addition of a -k parameter to indicate MQEV is running as an IBM MQ service.**
   This parameter should be used when running as an IBM MQ service. This mode of execution essentially says that MQEV should start and end in line with the Queue Manager itself. So, when the Queue Manager ends MQEV will not attempted any retries. Please see Chapter.9 Running MQEV with your Queue Manager on page 45 for more information.
2. **A new command `'STOP EV'` has been added**
   An administrator can end MQEV at any time by issuing the `'STOP EV'` command. Please see Chapter 11.32 STOP EV on page 152 for a description of this command.
3. **New and changed expression functions**
   - `valueof()`
   - `power()`
   - `system()` function now has an optional second parameter to determine whether to run synchronously or asynchronously.
4. **z/OS Support**
   MQEV is now available to run locally on z/OS. To enable this you require a z/OS specific licence. A distributed platform MQEV licence will not enable MQEV on z/OS to run.

22 Migration from a previous version

We always try to ensure that, as each version is shipped, all the features that were working on the previous versions remain intact. Before installing a new version we always recommend you first backup the contents of your **MQGEM.MQEV.DATA.QUEUE**. Please read 3.2 Upgrade on page 14 for more information on backing up your data. If you do find a problem then please send us a problem report and we will try to fix your issue as soon as possible.

22.1 Migrating from a version prior to Version 9.2.2

The response to a **DISPLAY EVENTS** command showing a configuration event has changed slightly. The change affects the following set of attributes in the response.

Field Name	PCF constant	MQSC name
EventUserId	MQCACF_EVENT_USER_ID	EVENTUSER
EventSecurityId	MQBACF_EVENT_SECURITY_ID	EVSID
EventOrigin	MQIACF_EVENT_ORIGIN	EVORIGIN
EventAccountingToken	MQBACF_EVENT_ACCOUNTING_TOKEN	EVACCTTK
EventIdentityData	MQCACF_EVENT_APPL_IDENTITY	EVAPPLID
EventApplType	MQIACF_EVENT_APPL_TYPE	EVAPPLTYPE
EventApplName	MQCACF_EVENT_APPL_NAME	EVAPPLNAME
EventApplOrigin	MQCACF_EVENT_APPL_ORIGIN	EVAPPLORIG

If you were issuing a PCF command to do this, these fields were previously returned in the group **MQG_GROUP_BEFORE** if the event was a configuration change event, and not in any group for a create, delete or refresh configuration event.

In V9.2.2 these fields are now always returned in the group **MQGACF_COMMAND_CONTEXT** matching what happens with a command event. The MO71 product issues a PCF command and already handles this difference.

If you were issuing an MQSC command to display event details, you will now notice a **CMDCTX:** label before these fields and then the **BEFORE:** label after these fields as shown in the (trimmed) example below.

```
EVQMGR(MQG1)           EVENTS($EVENTS)        EVTIME(2021-08-17 18:08:40 (Local))
EVREASON(CFGCHGOBJ)    EVTYPE(CONFIG)         EVUSERID(mqgemusr)     EVOBJNAME(APPL.QLOCAL)
EVOBJTYPE(QUEUE)       EVENTID(00000002)      CFHCMD(43)             CFHREASON(2368)
SUMMARY(Config - Change Object - Queue:APPL.QLOCAL - CLWLUSEQ[QMGR -> LOCAL])
CMDCTX:
EVENTUSER(mqgemusr)    EVORIGIN(MSG)          EVACCTTK(160105150000001AFA5FFE70975006C3A1...)
EVAPPLID( )            EVAPPLTYPE(WINDOWSNT) EVAPPLNAME(Qchange)    EVAPPLORIG( )
BEFORE:
OBJTYPE(QUEUE)         QUEUE(APPL.QLOCAL)     DESCR( )               PROCESS( )
CLWLPRTY(0)            CLWLUSEQ(QMGR)         DEFPRESP(SYNC)         DEFREADA(NO)
DEFTYPE(PREDEFINED)    QTYPE(QLOCAL)
AFTER:
CLWLUSEQ(LOCAL)
```

Configuration create, delete and refresh events previously did not contain any groups, but now contain the **MQGACF_COMMAND_CONTEXT** and one of the **MQG_GROUP_BEFORE** or **MQG_GROUP_AFTER** groups. The Configuration change events contain all three groups.

	CMDCTX	BEFORE	AFTER
Configuration Create Event	✓		✓
Configuration Change Event	✓	✓	✓
Configuration Delete Event	✓	✓	
Configuration Refresh Event	✓		✓

In MQSCX scripts, you can still address all the response variables by name as before. For example, in V9.2.1 and in the new version, the following command still produces the same result.

```
print EVACCTTK
```

If you were using a WHERE clause to directly utilise these fields, the following syntax worked in V9.2.1 and still works in the new version.

```
WHERE(EVORIGIN EQ MSG)
```

However, in V9.2.1, the following syntax also worked to achieve the same thing as the above command, and this syntax now does not work because this field is no longer considered part of the **BEFORE** group.

```
WHERE(BEFORE.EVORIGIN EQ MSG)
```

Appendix A. Expression Operators

Here is a list of the available operators. The majority of them can be used in both the =WHERE() clause and in normal control language expressions. Where there are restrictions they will be noted against the operator itself.

Operator	Meaning	Synonyms
	Standard WHERE clause operators	
EQ	Equals	=
NE	Not Equals	<> !=
GE	Greater Than or equals	>=
LE	Less Than or equals	<=
GT	Greater Than	>
LT	Less Than	<
LK	Like – wildcard comparison. (see wildcard note below)	==
NL	Not Like – wildcard comparison. (see wildcard note below)	
CT	Contains	
EX	Does not contain	
CTG	Contains generic (see wildcard note below)	
EXG	Does not contain generic (see wildcard note below)	
	Additional Operators	
=	Equals	**EQ**
==	Wildcard comparison	**LIKE**
!=	Not equals	**NE** <>
<>	Not equals	**NE** !=
OR	Logical OR	|
|	Logical OR	**OR**
||	Bitwise OR	
AND	Logical AND	**&**
&	Logical AND	**AND**
&&	Bitwise AND	
>	Greater Than	**GT**
>=	Greater Than or Equals	**GE**
<	Less Than	**LT**
<=	Less Than or Equals	**LE**
-	Minus	
+	Plus	
*	Multiply	
/	Divide	
%	Modulus	
NOT	Logical NOT	**!**
!	Logical Not	**NOT**

Appendix B. Expression Functions

Here is a list of the available functions the majority of them can be used in both the =WHERE() clause and in normal control language expressions. Where there are restrictions they will be noted against the function itself.

<table>
<tr><th>Function</th><th>Meaning</th></tr>
<tr><td>ceil(<value>)</td><td>Returns the highest equivalent integer. The main use of this function is to convert real numbers to integers. For example ceil(7.3) has the value 8.</td></tr>
<tr><td>date()
date(<time>)
date(<time>,<format string>)</td><td>This function takes 0, 1 or 2 parameters.

The <time> parameter is the number of seconds since January 1st 1970.

If no parameters are given the current time is returned in the default format. If just the time parameter is passed then that time is returned in the default format. If both a time and format is given then the returned value is the time in a formatted string according to the following values for the format string:
<table>
<tr><th>Format</th><th>Meaning</th></tr>
<tr><td>H</td><td>Two digit hour (24 hour clock)</td></tr>
<tr><td>HH</td><td>Hour (24 hour clock)</td></tr>
<tr><td>h</td><td>Two digit hour (12 hour clock)</td></tr>
<tr><td>hh</td><td>Hour (12 hour clock)</td></tr>
<tr><td>M</td><td>Two digit minutes</td></tr>
<tr><td>S</td><td>Two digit seconds</td></tr>
<tr><td>d</td><td>Two digit day of month</td></tr>
<tr><td>dd</td><td>Day of month including suffix eg.1st, 2nd, 3rd</td></tr>
<tr><td>j</td><td>Julian day of year (zero based)</td></tr>
<tr><td>J</td><td>Julian day of year (one based)</td></tr>
<tr><td>m</td><td>Three character month name eg.Jan,Feb,Mar</td></tr>
<tr><td>mm</td><td>Two digit month</td></tr>
<tr><td>mmm</td><td>Full month name eg. January, February,March</td></tr>
<tr><td>P</td><td>AM/PM</td></tr>
<tr><td>p</td><td>am/pm</td></tr>
<tr><td>y</td><td>Four digit year</td></tr>
<tr><td>yy</td><td>Two digit year</td></tr>
<tr><td>D</td><td>Three character day of week eg.Mon,Tue,Wed</td></tr>
<tr><td>DD</td><td>Full character day of week eg. Monday, Tuesday, Wednesday</td></tr>
<tr><td>t</td><td>Simple time format eg. 18:14:03</td></tr>
<tr><td>\<char></td><td>Escape character sequence. eg. \m will print ‘m’</td></tr>
</table>
The default format is : H:M d/mm/y eg 17:05 02/04/2014

For example, to print out the current day of the week enter the command:

print date(_time,”DD”)</td></tr>
<tr><td>delvar(<variable></td><td>Will delete the given variable. The variable can be a normal user variable, an array name or an individual element of an array. For example delvar(@a) and delvar(@a[3,4]) are both valid.</td></tr>
</table>

eval(<expression>)	This function takes a single expression and returns its calculated value. The normal use of this function is by passing a string which should be evaluations. For example, **eval("curdepth > 0")**
exists(<variable>)	Returns TRUE if the given variable exists and has a defined value. The variable can be a user or system variable or a user variable array element. You can not use this function on a response variable.
fclose(<file identifier>)	Closes the file given by the file identifier. The file identifier must have been previously return by fopen()

fgets(<file identifier>,<variable>)	Read the next line in the file given by the file identifier.
	The function returns:
	• the length of the line which is read
	• -1 to indicate end of file
	• -2 to indicate a file read error
	If a line is read then its contents is placed in the given variable. For example:
	`if (fgets(@hf,@line)>=0)`
	`  print @line`
	`endif`

findstr(<string>,<search string>)	Returns the offset in the string of the given search string. The search is case sensitive. If the string is not found 0 is returned. A search for the NULL string ("") will always return 1
findstri(<string>,<search string>)	Returns the offset in the string of the given search string. The search is case insensitive. If the string is not found 0 is returned. A search for the NULL string ("") will always return 1
floor(<value>)	Returns the lowest equivalent integer. The main use of this function is to convert real numbers to integers. For example floor(7.3) has the value 7.
fopen(<filename>,<mode>)	Opens a file and returns the file identifier. A value of -1 is returned if the file is not opened. The function is essentially just a wrapper around the C runtime fopen() function so the same modes should work. Essentially though the modes are: • "r" - open file for read • "w"- open file for write - note than any current content will be destroyed • "a" - open file for append If the call returns -1 then the system variable _errno can be used to determine the cause of the failure. The returned file identifier can be passed to fprint, as the first parameter, to write to the file.
getkey(<interval>)	Returns the next key code typed or 0 if no key is typed within the given interval. The interval is specified in seconds. Note that it is the code, not the character, that is returned. For example 'a' =61, 'b'=62, 'c'=63 and so on. The escape key returns 27.
lower(<string>)	Return the lower case version of the given string
max(a,b,c,....)	Return the maximum of the parameters There may be any number of parameters or any type.
maxsub(<array variable>) **maxsub(<array variable>,<index>)**	Returns the maximum subscript use on the array for the given index. If an index is not specified the first index, 1, is assumed.
min(a,b,c,....)	Return the minimum of the parameters There may be any number of parameters or any type.
minsub(<array variable>) **minsub(<array variable>,<index>)**	Returns the minimum subscript use on the array for the given index. If an index is not specified the first index, 1, is assumed.

mqtime(<date>,<time>)	Takes the date and time as strings in the format of: “2004-06-17” and “20.28.08” or “20040617” and “202808” and returns the number of seconds since January 1st 1970
numsubs(<array variable>)	Returns the number of subscripts used by the given array. For example, suppose you define two variables @a[10] = 3 and @a[14,27] = “Hi” then nusubs(@a) will return 2 since the maximum number of subscripts used by the variable @a array is 2.
power(x,y)	Returns a real number with the value of x^y For example `power(2,3) = 8.00` and `power(16,0.5) = 4.00`
round(<value>)	Returns the nearest equivalent integer. The main use of this function is to convert real numbers to integers. For example round(7.3) has the value 7.
sort(<array>) **sort(<array>,<column>)**	The sort function will sort a one or two dimensional array into ascending order. With two dimensional arrays you can optionally pass a column parameter giving the column you want to sort by.
sortd(<array>) **sortd(<array>,<column>)**	A sort function just like 'sort()' above but it sorts in descending order.
str(<value>)	Returns the string representation of the value. The main use of this function is to be able to concatenate strings with numbers since without the function the strings are coerced into numbers. For example “Hi”+1 has the value 3 since the length the of string “Hi” is 2. However, “Hi” + str(1) is “Hi1”.
strlen(<string>)	Returns the length of the given string
strreplace(<string>,<old>,<new>)	Returns a string where all the occurrences of substring <old> are replaced with substring <new> in the string <string>. For example **strreplace(“Q.TST”,”.TST”,”.PRD”)** has the result of “Q.PRD”. Passing an empty <old> substring will return <string> as the result. Passing an empty <new> substring will effectively just delete occurrences of <old>.
substr(<string>,<offset>,<length>	Returns the substring for the given length at the given offset. The first character of a string has an offset of 1. If the length parameter is greater than the available number of characters then the returned string is shortened accordingly.
sqrt(<number>)	Return the square root of the given parameter as a real number
system(<command>) **system(<command>,<options>)**	This function takes 1 or 2 parameters. Will invoke the system with the given command string. This is used to start other programs. For example: `'q -oLOG -M"event.summary"'` could be used to send an MQ message containing the event summary string, to a logging queue. The options provided in the second parameter can be: Option — Meaning `const.SYNC` — The command is run synchronously and control does not return to the script until the command completes. `const.ASYNC` — The command is run asynchronously and control is immediately returned to the script, while the command continues. If the second parameter is not specified, the command is run asynchronously. For more information about using the `system()` function, see 14.1 Invoking other programs from your script on page 160.

<table>
<tr><td>upper(<string>)</td><td>Return the upper case version of the given string</td></tr>
<tr><td>valueof(<string>,<string>)</td><td>
A function which will parse a set of fields of the format “...field1(value)...” and return the value associated with that field. The returned value is always a string regardless of its contents. This can be useful for processing parameters to the program or indeed parsing the response from the queue manager such as:
<pre>valueof(“curdepth”, _lastresp)</pre>
If any value is itself contained in quotes then these quotes will be stripped from the returned value.

If you need to convert a returned string into an integer then you can use the function <code>eval()</code>. For example <code>eval("123")</code> has the value 123. Of course <code>eval()</code> can take any expression so you can pass any valid expression this way.

To determine whether a value has been returned at all you can use the function <code>exists()</code>.

For example, suppose you had the following string variable :
<pre>@s = "a(Hello) b(123) c() d('another')"</pre>
The the result of the of the function is as follows:
<table>
<tr><td><code>valueof("a",@s)</code></td><td>Has the value “Hello”</td></tr>
<tr><td><code>valueof("b",@s)</code></td><td>Has the value “123”
Note that this is a string, not a number.</td></tr>
<tr><td><code>valueof("c",@s)</code></td><td>Has the value “”</td></tr>
<tr><td><code>valueof("d”,@s)</code></td><td>Has the value “another”
Note that the single quotes are stripped.</td></tr>
<tr><td><code>valueof("x",@s)</code></td><td>Is <Not Set></td></tr>
</table>
</td></tr>
<tr><td>wait(<number>)</td><td>Waits the given number of seconds before returning. The function returns TRUE.</td></tr>
</table>

Appendix C. Variable Names

A variable name field must conform to the following rules.

- Names are case insensitive
- Names must not start with a numeric field
- Names may only valid characters, where valid characters are any of:
  - Alphanumeric characters
  - Underscore (_)
  - Period (.)

Appendix D. Event Reasons

Here is the full list of event reasons which is used by the `DISPLAY EVENTS()` command Those highlighted will return all the event reasons listed below them.

MQSC Value	Meaning	PCF Constant
AUTHOR	Any authority event	MQG_EVENT_REASON_AUTH
AUTCON	Connection not authorised	MQG_EVENT_REASON_AUTH_CONN_NOT_AUTH
AUTSYSCON	System connection not authorised	MQG_EVENT_REASON_AUTH_SYS_CONN_NOT_AUTH
AUTCSP	CSP not authorised	MQG_EVENT_REASON_AUTH_CSP_NOT_AUTH
AUTOPEN	Open not authorised	MQG_EVENT_REASON_AUTH_OPEN_NOT_AUTH
AUTCLOSE	Close not authorised	MQG_EVENT_REASON_AUTH_CLOSE_NOT_AUTH
AUTCMD	Command not authorised	MQG_EVENT_REASON_AUTH_CMD_NOT_AUTH
AUTSUB	Subscription not authorised	MQG_EVENT_REASON_AUTH_SUB_NOT_AUTH
AUTSUBDST	Subscription Destination not authorised	MQG_EVENT_REASON_AUTH_SUB_DEST_NOT_AUTH
CHANNEL	Any channel event	MQG_EVENT_REASON_CHANNEL
CHLACT	Channel activated	MQG_EVENT_REASON_CHANNEL_ACTIVATED
CHLADEFERR	Channel auto-define error	MQG_EVENT_REASON_CHANNEL_AUTO_DEF_ERROR
CHLADEFOK	Channel auto-defined OK	MQG_EVENT_REASON_CHANNEL_AUTO_DEF_OK
CHLBLK	Channel blocked	MQG_EVENT_REASON_CHANNEL_BLOCKED
CHLCONV	Channel conversion error	MQG_EVENT_REASON_CHANNEL_CONV_ERROR
CHLNOTACT	Channel not activated	MQG_EVENT_REASON_CHANNEL_NOT_ACTIVATED
CHLNOTAVL	Channel not available	MQG_EVENT_REASON_CHANNEL_NOT_AVAILABLE
CHLMAXACT	Channel Max Active	MQG_EVENT_REASON_CHANNEL_MAX_ACTIVE
CHLMAXCHL	Channel Max Channels	MQG_EVENT_REASON_CHANNEL_MAX_CHANNELS
CHLMAXINST	Channel Max Instances	MQG_EVENT_REASON_CHANNEL_MAX_INST
CHLMAXINSTC	Channel Max Instances Client	MQG_EVENT_REASON_CHANNEL_MAX_INSTC
CHLSTR	Channel started	MQG_EVENT_REASON_CHANNEL_STARTED
CHLSTP	Channel stopped	MQG_EVENT_REASON_CHANNEL_STOPPED
CHLSTPU	Channel stopped by user	MQG_EVENT_REASON_CHANNEL_STOPPED_BY_USER
CHLBLKWRN	Channel blocked warning	MQG_EVENT_REASON_CHANNEL_BLOCKED_WARNING
CHLBLKADDR	Channel Blocked Address	MQG_EVENT_REASON_CHANNEL_BLOCKED_ADDRESS
CHLBLKUSER	Channel Blocked User	MQG_EVENT_REASON_CHANNEL_BLOCKED_USER
CHLBLKNOACC	Channel Blocked No Access	MQG_EVENT_REASON_CHANNEL_BLOCKED_NOACCESS
CHLBKLWADDR	Channel Blocked Warning Address	MQG_EVENT_REASON_CHANNEL_WBLOCKED_ADDRESS
CHLBLKWUSER	Channel Blocked Warning User	MQG_EVENT_REASON_CHANNEL_WBLOCKED_USER
CHLBLKWNOACC	Channel Blocked Warning No Access	MQG_EVENT_REASON_CHANNEL_WBLOCKED_NOACCESS
COMMAND	Any command event	MQG_EVENT_REASON_CMD
CMDARCLOG	Archive Log	MQG_EVENT_REASON_CMD_ARCHIVE_LOG
CMDBCKCF	Backup CF Struc	MQG_EVENT_REASON_CMD_BACKUP_CF_STRUC
CMDCHGAI	Change Auth Info	MQG_EVENT_REASON_CMD_CHANGE_AUTH_INFO

MQSC Value	Meaning	PCF Constant
CMDCHGBP	Change Buffer Pool	MQG_EVENT_REASON_CMD_CHANGE_BUFFER_POOL
CMDCHGCF	Change CF Struc	MQG_EVENT_REASON_CMD_CHANGE_CF_STRUC
CMDCHGCHL	Change Channel	MQG_EVENT_REASON_CMD_CHANGE_CHANNEL
CMDCHGCI	Change Comm Info	MQG_EVENT_REASON_CMD_CHANGE_COMM_INFO
CMDCHGLSTR	Change Listener	MQG_EVENT_REASON_CMD_CHANGE_LISTENER
CMDCHGNL	Change Namelist	MQG_EVENT_REASON_CMD_CHANGE_NAMELIST
CMDCHGPS	Change Page Set	MQG_EVENT_REASON_CMD_CHANGE_PAGE_SET
CMDCHGPRC	Change Process	MQG_EVENT_REASON_CMD_CHANGE_PROCESS
CMDCHGQ	Change Queue	MQG_EVENT_REASON_CMD_CHANGE_Q
CMDCHGQMGR	Change Queue Manager	MQG_EVENT_REASON_CMD_CHANGE_Q_MGR
CMDCHGSEC	Change Security	MQG_EVENT_REASON_CMD_CHANGE_SECURITY
CMDCHGSVC	Change Service Object	MQG_EVENT_REASON_CMD_CHANGE_SERVICE
CMDCHGSTGC	Change Storage Class	MQG_EVENT_REASON_CMD_CHANGE_STG_CLASS
CMDCHGSUB	Change Subscription	MQG_EVENT_REASON_CMD_CHANGE_SUBSCRIPTION
CMDCHGTOP	Change Topic	MQG_EVENT_REASON_CMD_CHANGE_TOPIC
CMDCHGTRC	Change Trace	MQG_EVENT_REASON_CMD_CHANGE_TRACE
CMDCLRQ	Clear Queue	MQG_EVENT_REASON_CMD_CLEAR_Q
CMDCLRTOPS	Clear Topic String	MQG_EVENT_REASON_CMD_CLEAR_TOPIC_STRING
CMDCRTAI	Create Auth Info	MQG_EVENT_REASON_CMD_CREATE_AUTH_INFO
CMDCRTBP	Create Buffer Pool	MQG_EVENT_REASON_CMD_CREATE_BUFFER_POOL
CMDCRTCF	Create CF Struc	MQG_EVENT_REASON_CMD_CREATE_CF_STRUC
CMDCRTCHL	Create Channel	MQG_EVENT_REASON_CMD_CREATE_CHANNEL
CMDCRTCI	Create Comm Info	MQG_EVENT_REASON_CMD_CREATE_COMM_INFO
CMDCRTLSTR	Create Listener	MQG_EVENT_REASON_CMD_CREATE_LISTENER
CMDCRTNL	Create Namelist	MQG_EVENT_REASON_CMD_CREATE_NAMELIST
CMDCRTPS	Create Page Set	MQG_EVENT_REASON_CMD_CREATE_PAGE_SET
CMDCRTPRC	Create Process	MQG_EVENT_REASON_CMD_CREATE_PROCESS
CMDCRTQ	Create Queue	MQG_EVENT_REASON_CMD_CREATE_Q
CMDCRTSVC	Create Service Object	MQG_EVENT_REASON_CMD_CREATE_SERVICE
CMDCRTSTGC	Create Storage Class	MQG_EVENT_REASON_CMD_CREATE_STG_CLASS
CMDCRTSUB	Create Subscription	MQG_EVENT_REASON_CMD_CREATE_SUBSCRIPTION
CMDCRTTOP	Create Topic	MQG_EVENT_REASON_CMD_CREATE_TOPIC
CMDDLTAI	Delete Auth Info	MQG_EVENT_REASON_CMD_DELETE_AUTH_INFO
CMDDLTCF	Delete CF Struc	MQG_EVENT_REASON_CMD_DELETE_CF_STRUC
CMDDLTCHL	Delete Channel	MQG_EVENT_REASON_CMD_DELETE_CHANNEL
CMDDLTCI	Delete Comm Info	MQG_EVENT_REASON_CMD_DELETE_COMM_INFO
CMDDLTLSTR	Delete Listener	MQG_EVENT_REASON_CMD_DELETE_LISTENER
CMDDLTNL	Delete Namelist	MQG_EVENT_REASON_CMD_DELETE_NAMELIST
CMDDLTPS	Delete PageSet	MQG_EVENT_REASON_CMD_DELETE_PAGE_SET

MQSC Value	Meaning	PCF Constant
CMDDLTPRC	Delete Process	MQG_EVENT_REASON_CMD_DELETE_PROCESS
CMDDLTQ	Delete Queue	MQG_EVENT_REASON_CMD_DELETE_Q
CMDDLTSVC	Delete Service	MQG_EVENT_REASON_CMD_DELETE_SERVICE
CMDDLTSTGC	Delete Storage Class	MQG_EVENT_REASON_CMD_DELETE_STG_CLASS
CMDDLTSUB	Delete Subscription	MQG_EVENT_REASON_CMD_DELETE_SUBSCRIPTION
CMDDLTTOP	Delete Topic	MQG_EVENT_REASON_CMD_DELETE_TOPIC
CMDINQARC	Inquire Archive	MQG_EVENT_REASON_CMD_INQUIRE_ARCHIVE
CMDINQAI	Inquire Auth Info	MQG_EVENT_REASON_CMD_INQUIRE_AUTH_INFO
CMDINQCF	Inquire CF Struc	MQG_EVENT_REASON_CMD_INQUIRE_CF_STRUC
CMDINQCFS	Inquire CF Struc Status	MQG_EVENT_REASON_CMD_INQUIRE_CF_STRUC_STATUS
CMDINQCHL	Inquire Channel	MQG_EVENT_REASON_CMD_INQUIRE_CHANNEL
CMDINQCHLI	Inquire Channel Initiator	MQG_EVENT_REASON_CMD_INQUIRE_CHANNEL_INIT
CMDINQCHS	Inquire Channel Status	MQG_EVENT_REASON_CMD_INQUIRE_CHANNEL_STATUS
CMQINQCARC	Inquire Archive	MQG_EVENT_REASON_CMD_INQUIRE_CHLAUTH_RECS
CMDINQCLQM	Inquire Cluster Queue Manager	MQG_EVENT_REASON_CMD_INQUIRE_CLUSTER_Q_MGR
CMDINQCSVR	Inquire Command Server	MQG_EVENT_REASON_CMD_INQUIRE_CMD_SERVER
CMDINQCI	Inquire Comm Info	MQG_EVENT_REASON_CMD_INQUIRE_COMM_INFO
CMDINQCONN	Inquire Connection	MQG_EVENT_REASON_CMD_INQUIRE_CONNECTION
CMDINQLSTR	Inquire Listener	MQG_EVENT_REASON_CMD_INQUIRE_LISTENER
CMDINQLOG	Inquire Log	MQG_EVENT_REASON_CMD_INQUIRE_LOG
CMDINQNL	Inquire Namelist	MQG_EVENT_REASON_CMD_INQUIRE_NAMELIST
CMDINQPRC	Inquire Process	MQG_EVENT_REASON_CMD_INQUIRE_PROCESS
CMDINQPSST	Inquire Pub/Sub Status	MQG_EVENT_REASON_CMD_INQUIRE_PUBSUB_STATUS
CMDINQQ	Inquire Queue	MQG_EVENT_REASON_CMD_INQUIRE_Q
CMDINQQMGR	Inquire Queue Manager	MQG_EVENT_REASON_CMD_INQUIRE_Q_MGR
CMDINQQSG	Inquire QSG	MQG_EVENT_REASON_CMD_INQUIRE_QSG
CMDINQQSTS	Inquire Queue Status	MQG_EVENT_REASON_CMD_INQUIRE_Q_STATUS
CMDINQSEC	Inquire Security	MQG_EVENT_REASON_CMD_INQUIRE_SECURITY
CMDINQSVC	Inquire Service	MQG_EVENT_REASON_CMD_INQUIRE_SERVICE
CMDINQSTGC	Inquire Storage Class	MQG_EVENT_REASON_CMD_INQUIRE_STG_CLASS
CMQINQSUB	Inquire Subscription	MQG_EVENT_REASON_CMD_INQUIRE_SUBSCRIPTION
CMDINQSUSD	Inquire Subscription Status	MQG_EVENT_REASON_CMD_INQUIRE_SUB_STATUS
CMDINQSYS	Inquire System	MQG_EVENT_REASON_CMD_INQUIRE_SYSTEM
CMDINQTHD	Inquire Thread	MQG_EVENT_REASON_CMD_INQUIRE_THREAD
CMDINQTOP	Inquire Topic	MQG_EVENT_REASON_CMD_INQUIRE_TOPIC
CMDINQTOPS	Inquire Topic Status	MQG_EVENT_REASON_CMD_INQUIRE_TOPIC_STATUS
CMDINQTRC	Inquire Trace	MQG_EVENT_REASON_CMD_INQUIRE_TRACE
CMDINQUSE	Inquire Usage	MQG_EVENT_REASON_CMD_INQUIRE_USAGE
CMDMOVQ	Move Queue	MQG_EVENT_REASON_CMD_MOVE_Q

MQSC Value	Meaning	PCF Constant
CMDPNGCHL	Ping Channel	MQG_EVENT_REASON_CMD_PING_CHANNEL
CMDRECBSDS	Recover BSDS	MQG_EVENT_REASON_CMD_RECOVER_BSDS
CMDRECCF	Recover CF Struc	MQG_EVENT_REASON_CMD_RECOVER_CF_STRUC
CMDREFCLUS	Refresh Cluster	MQG_EVENT_REASON_CMD_REFRESH_CLUSTER
CMDREFQM	Refresh Queue Manager	MQG_EVENT_REASON_CMD_REFRESH_Q_MGR
CMDREFSEC	Refresh Security	MQG_EVENT_REASON_CMD_REFRESH_SECURITY
CMDRESCHL	Reset Channel	MQG_EVENT_REASON_CMD_RESET_CHANNEL
CMDRESCLUS	Reset Cluster	MQG_EVENT_REASON_CMD_RESET_CLUSTER
CMDRESQM	Reset Queue Manager	MQG_EVENT_REASON_CMD_RESET_Q_MGR
CMDRESQST	Reset Queue Statistics	MQG_EVENT_REASON_CMD_RESET_Q_STATS
CMDRESTPIP	Reset TPIPE	MQG_EVENT_REASON_CMD_RESET_TPIPE
CMDRLVCHL	Resolve Channel	MQG_EVENT_REASON_CMD_RESOLVE_CHANNEL
CMDRLVIND	Resolve Indoubt	MQG_EVENT_REASON_CMD_RESOLVE_INDOUBT
CMDRSMQM	Resume Queue Manager	MQG_EVENT_REASON_CMD_RESUME_Q_MGR
CMDRSMQMC	Resume Queue Manager Cluster	MQG_EVENT_REASON_CMD_RESUME_Q_MGR_CLUSTER
CMDREVSEC	Reverify Security	MQG_EVENT_REASON_CMD_REVERIFY_SECURITY
CMDSETARC	Set Archive	MQG_EVENT_REASON_CMD_SET_ARCHIVE
CMDSETCA	Set Channel Auth	MQG_EVENT_REASON_CMD_SET_CHLAUTH_REC
CMDSETLOG	Set Log	MQG_EVENT_REASON_CMD_SET_LOG
CMDSETSYS	Set System	MQG_EVENT_REASON_CMD_SET_SYSTEM
CMDSTRCHL	Start Channel	MQG_EVENT_REASON_CMD_START_CHANNEL
CMDSTRCHLI	Start Channel Initiator	MQG_EVENT_REASON_CMD_START_CHANNEL_INIT
CMDSTRLSTR	Start Listener	MQG_EVENT_REASON_CMD_START_CHANNEL_LISTENER
CMDSTRCSVR	Start Command Server	MQG_EVENT_REASON_CMD_START_CMD_SERVER
CMDSTRSVC	Start Service	MQG_EVENT_REASON_CMD_START_SERVICE
CMDSTRTRC	Start Trace	MQG_EVENT_REASON_CMD_START_TRACE
CMDSTPCHL	Stop Channel	MQG_EVENT_REASON_CMD_STOP_CHANNEL
CMDSTPCHLI	Stop Channel Initiator	MQG_EVENT_REASON_CMD_STOP_CHANNEL_INIT
CMDSTPLSTR	Stop Listener	MQG_EVENT_REASON_CMD_STOP_CHANNEL_LISTENER
CMDSTPCSVR	Stop Command Server	MQG_EVENT_REASON_CMD_STOP_CMD_SERVER
CMDSTPCONN	Stop Connection	MQG_EVENT_REASON_CMD_STOP_CONNECTION
CMDSTPSVC	Stop Service	MQG_EVENT_REASON_CMD_STOP_SERVICE
CMDSTPTRC	Stop Trace	MQG_EVENT_REASON_CMD_STOP_TRACE
CMSSUSQM	Suspend Queue Manager	MQG_EVENT_REASON_CMD_SUSPEND_Q_MGR
CMDSUSQMCL	Suspend Queue Manager Cluster	MQG_EVENT_REASON_CMD_SUSPEND_Q_MGR_CLUSTER
CMDUNKNOWN	Unknown Command	MQG_EVENT_REASON_CMD_UNKNOWN
CONFIG	Any configuration Event	MQG_EVENT_REASON_CONFIG
CFGCHGOBJ	Object change	MQG_EVENT_REASON_CONFIG_CHANGE_OBJECT
CFGCRTOBJ	Object creation	MQG_EVENT_REASON_CONFIG_CREATE_OBJECT

MQSC Value	Meaning	PCF Constant
CFGDLTOBJ	Object deletion	MQG_EVENT_REASON_CONFIG_DELETE_OBJECT
CFGREFOBJ	Object Refresh	MQG_EVENT_REASON_CONFIG_REFRESH_OBJECT
INHIBIT	Any Inhibit Event	MQG_EVENT_REASON_INHIBIT
INHGET	Inhibit Get	MQG_EVENT_REASON_INHIBIT_GET
INHPUT	Inhibit Put	MQG_EVENT_REASON_INHIBIT_PUT
LOCAL	Any Local Event	MQG_EVENT_REASON_LOCAL
LCLABASE	Alias Base Error	MQG_EVENT_REASON_LOCAL_ALIAS_BASE_ERROR
LCLUNKABS	Unknown Alias Base Queue	MQG_EVENT_REASON_LOCAL_UNKNOWN_ALIAS_BASE_Q
LCLUNKOBJ	Unknown Object Name	MQG_EVENT_REASON_LOCAL_UNKNOWN_OBJECT_NAME
LOGGER	Any Logger Event	MQG_EVENT_REASON_LOGGER
LOGSTS	Logger Status	MQG_EVENT_REASON_LOGGER_STATUS
PERFM	Any Performance Event	MQG_EVENT_REASON_PERFM
PERQDPHI	Queue High	MQG_EVENT_REASON_PERFM_Q_DEPTH_HIGH
PERQDPLO	Queue Low	MQG_EVENT_REASON_PERFM_Q_DEPTH_LOW
PERQDPFULL	Queue Full	MQG_EVENT_REASON_PERFM_Q_DEPTH_FULL
PERQSVCHI	Queue Service Interval High	MQG_EVENT_REASON_PERFM_Q_SVC_INTERVAL_HIGH
PERQSVCOK	Queue Service Interval OK	MQG_EVENT_REASON_PERFM_Q_SVC_INTERVAL_OK
REMOVE	Any Remote Event	MQG_EVENT_REASON_REMOTE
REMDXMQTYPE	Default Transmission Queue Type Error	MQG_EVENT_REASON_REMOTE_DEF_XMITQ_TYPE_ERROR
REMDXMQUSE	Default Transmission Queue Usage Error	MQG_EVENT_REASON_REMOTE_DEF_XMITQ_USAGE_ERROR
REMQTYPERR	Queue Type Error	MQG_EVENT_REASON_REMOTE_Q_TYPE_ERROR
REMQNAMERR	Remote Queue Name Error	MQG_EVENT_REASON_REMOTE_REMOTE_Q_NAME_ERROR
REMXMQTYPE	Transmission Queue Type Error	MQG_EVENT_REASON_REMOTE_XMIT_Q_TYPE_ERROR
REMXMQUSE	Transmission Queue Usage Error	MQG_EVENT_REASON_REMOTE_XMIT_Q_USAGE_ERROR
REMUNKDXMTQ	Unknown Default Transmission Queue	MQG_EVENT_REASON_REMOTE_UNKNOWN_DEF_XMITQ
REMUNKRQM	Unknown Remote Queue Manager	MQG_EVENT_REASON_REMOTE_UNKNOWN_REMOTE_QMGR
REMUNKXMQ	Unknown Transmission Queue	MQG_EVENT_REASON_REMOTE_UNKNOWN_XMIT_Q
SSL	Any SSL Event	MQG_EVENT_REASON_SSL
SSLERROR	SSL Error	MQG_EVENT_REASON_SSL_ERROR
SSLWARN	SSL Warning	MQG_EVENT_REASON_SSL_WARNING
SSLERRHSK	SSL Handshake Error	MQG_EVENT_REASON_SSL_HANDSHAKE_ERROR
SSLERRCIPH	SSL Cipher Spec Error	MQG_EVENT_REASON_SSL_CIPHER_SPEC_ERROR
SSLERRPEER	SSL Peer Name Error	MQG_EVENT_REASON_SSL_PEER_NAME_ERROR
SSLERRCAUT	SSL Client Auth Error	MQG_EVENT_REASON_SSL_CLIENT_AUTH_ERROR
STRSTP	Any Start/Stop Event	MQG_EVENT_REASON_STRSTP
SSTPQMA	Queue Manager Active	MQG_EVENT_REASON_STRSTP_QMGR_ACTIVE
SSTPQMNA	Queue Manager Not Active	MQG_EVENT_REASON_STRSTP_QMGR_NOT_ACTIVE
IMSBR	Any IMS Bridge Event	MQG_EVENT_REASON_BRIDGE
IMSBRSTR	Bridge started	MQG_EVENT_REASON_BRIDGE_STARTED

MQSC Value	Meaning	PCF Constant
IMSBRSTP	Bridge stopped	`MQG_EVENT_REASON_BRIDGE_STOPPED`
UNKNOWN	Event Unknown[32]	`MQG_EVENT_REASON_UNKNOWN`

32 This should not happen. If you see this response please ensure that you are using the latest version of MQEV. If you are using the latest version and you still get this response then please let us know and we will look into the issue.

End of document