MQMONA
Command Server Agent for IBM MQ
User Guide

Version 9.4.1

9th September 2026

Paul Clarke
Morag Hughson

MQGem Software Limited
support@mqgem.com

Take Note!

Before using this User's Guide and the product it supports, be sure to read the general information under "Notices”

Fourth Edition, September 2026

This edition applies to Version 9.4.1 of the MQMONA Command Server Agent for IBM MQ and to all subsequent releases and modifications until otherwise indicated in new editions.

(c) Copyright MQGem Software Limited 2023, 2026. All rights reserved.

Notices

The following paragraph does not apply in any country where such provisions are inconsistent with local law.

MQGEM SOFTWARE LIMITED PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.

Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

The information contained in this document has not be submitted to any formal test and is distributed AS IS. The use of the information or the implementation of any of these techniques is a customer responsibility and depends on the customer's ability to evaluate and integrate them into the customer's operational environment. While each item has been reviewed by MQGem Software for accuracy in a specific situation, there is no guarantee that the same or similar results will be obtained elsewhere. Customers attempting to adapt these techniques to their own environments do so at their own risk.

The following terms are trademarks of the International Business Machines Corporation in the United States and/or other countries:
IBM MQ
IBM
z/OS

The following terms are trademarks of the Microsoft Corporation in the United States and/or other countries:
Windows 95,98,Me
Windows NT, 2000,XP

Table of Contents

Chapter 1. MQMONA Command Server Agent for IBM MQ....1

1.1.Overview....1
1.2.Changes from previous version....1
1.3.Installation....2

Chapter 2. Licensing....3

2.1.Licence File Location....3
2.2.Multiple licences....3
2.3.Reporting found licences....3

Chapter 3. Getting Started....4

3.1.Configuring MO71 to use MQMONA....5
3.2.Configuring MQSCX to use MQMONA....6

Chapter 4. Consolidation....7

4.1.IBM MQ Commands and Responses....7
4.2.Consolidated Message Size....7

Chapter 5. Compression....9

5.1.IBM MQ string padding....9
5.2.MQMONA compression....9
5.3.String padding in MQSC....9

Chapter 6. Message Manipulation....10

6.1.Browsing a Queue....10
6.2.Copying or Moving messages....11
6.3.Deleting messages....12
6.4.Security....12

Chapter 7. Certificate Events....13

7.1.Certificates Nearing Expiry....13
7.2.Security....13
7.3.Inspection method....14

Chapter 8. Error Log Command....16

8.1.Configuration....16
8.2.Security....16
8.3.Error Log Location....17
8.4.Performance....17

Chapter 9. Command Reference....19

9.1.DISPLAY CERT....19
9.2.Inquire Error Log....25
9.3.Inquire Messages....26
9.4.Inquire Individual Message....30
9.5.Move/Copy Message....34
9.6.Delete Message....37
9.7.Close Queue....39
9.8.Abort Command....39
9.9.Progress Message (Response)....40
9.10.Inquire Queue Manager (Response) – Annotation....41

Chapter 10. Event Reference....42

10.1.Channel SSL Certificate Nearing Expiry....42

Chapter 11. Running MQMONA as a SERVICE....46

Chapter 12. Triggering **47**
12.1.Set up a trigger monitor 47
12.2.Describe MQMONA, the triggered application 47
Chapter 13. Security **49**
13.1.MQ Administrator using MQMONA 49
13.2.MQMONA Program 49
Chapter 14. Timing out commands **51**
Chapter 15. Status Reporting **52**
Chapter 16. Parameters **53**
16.1.Parameters Flags 53
Chapter 17. Simple Commands **54**
Chapter 18. Configuration File **55**
18.1.Example Configuration File 55
18.2.Configuration File keywords 55
18.3.Configuration inline in JCL on z/OS 59
Chapter 19. Worker Threads **60**
Chapter 20. Performance Comparisons **61**
20.1.IBM MQ Administrative Commands 61
20.2.Message Manipulation 62
20.3.Error Log Viewing 63
Chapter 21. Error reporting **64**
Chapter 22. Messages **65**
Chapter 23. Response Messages **85**
Chapter 24. Changes made in previous versions **89**
24.1.Changes made in version 9.4.0 89
24.2.Changes made in version 9.3.0 89
Chapter 25. Migration from previous versions **91**
25.1.Migrating from a version prior to V9.4.1 Build Date 09-09-2026 91

History

The MQMONA Command Server for IBM MQ agent was a simple 'C' program that was created in the same time frame as the MO71 IBM MQ GUI Administration SupportPac to mitigate problems with latency and bandwidth on slow networks. It was retired when the IBM MQ product added the client read-ahead feature which provided a different solution to the same problem, and it seemed that the agent was no longer necessary.

In recent times, we have been requested by some users to revive the agent as use of the read ahead feature does not always provide the necessary relief.

This product is loosely based on the original simple 'C' program, but has had a number improvements made from a serviceability and reporting perspective over and above what the original program did.

In addition, the ability to run commands that are only available to a locally logged on user, is another use of an agent program. The display of certificate data is the first of, we hope, many possibilities, along those lines.

I hope you find the program useful. As always I welcome your comments, both good and bad. Please feel free to e-mail me at support@mqgem.com with any bug reports or suggestions.

Chapter 1. MQMONA Command Server Agent for IBM MQ

1.1. Overview

The MQMONA Command Server Agent for IBM MQ is a program that can significantly improve the performance of MQ administrative tools in environments with low bandwidth, high latency, and/or large numbers of MQ objects. The agent achieves this by listening for MQSC and PCF commands issued by programs such as MO71 and MQSCX. It forwards them onto the IBM MQ Command Server and when it receives the reply messages, it compresses and consolidates them into a few, perhaps even only one, IBM MQ message, to be sent to the original application's reply queue, and retrieved by the application over the client network link.

When the IBM MQ Command Server responds to display commands, which result in multiple different objects being returned as the output, it does so with a single reply message for each object. By consolidating these many reply messages into one message, network latency becomes less of an issue, because there are fewer MQGET calls made by the original application. Read more about consolidation in Chapter 4. on page 7.

The IBM MQ Command Server tends to blank pad strings to their full length. A queue name, for example, is 48 characters long, and unless you have very long queue names, this tends to be a majority of blank characters. MQMONA strips those spaces to compress the data into a smaller message, and so bandwidth becomes less of an issue. Read more about compression in Chapter 5. on page 9.

1.2. Changes from previous version

The main changes from the previous version of the program are:

1. **Support for Browse and message manipulation**
   MQMONA now accepts various new commands that expedite browsing, copying, moving and deleting messages on queues.
   See Chapter 6. Message Manipulation on page 10 for full details.
2. **Implementation of the new MQCMD_INQUIRE_ERROR_LOG command**
   This has the advantage that an Administrator product such as MO71, can use the command on versions of IBM MQ before 9.4.5.
   For a description of this feature please refer to Chapter 8. Error Log Command on page 16.
3. **Name Mangling of error log path**
   For the default IBM MQ error log path, used by the `MQCMD_INQUIRE_ERROR_LOG` command, name mangling has been applied to the queue manager name part of the path. It has been clarified that **`mqerlPath`**, if used, must be the full path and not just the `MQ_DATA_PATH` portion. See 8.3.Error Log Location on page 17 for full details.
4. **Security Related fixes**
   MQMONA does all security by passing through the user ID from your admin tool for commands to be issued under, message manipulation to be done with, and replies to be returned to your ReplyToQ. Some fixes were made to ensure this is the case in all situations. The authority requirements are fully documented in a new chapter, Chapter 13. Security on page 49.

1.3. Installation

Installation has been made as simple as possible. Click the download button on the web site for the platform(s) you are interested in. This will download a zip file to your download location. Extract the files from the zip file using the appropriate tools for the platform listed below.

Platform	Unzip commands
AIX	gzip -d tar -xvf
Intel Linux	tar -xvzf
Power Linux	tar -xvzf
Linux on IBM Z	tar -xvzf
Windows	Extract using your favourite zip utility
z/OS	Extract using your favourite zip utility before transferring to a z/OS system. See 1.3.2 z/OS Installation Instructions below for more details

1.3.1. Linux, Unix and Windows

Once unzipped you should have the **mqmona** executable. Copy this file into the directory where you wish to run the program from. Normally this would be somewhere in your PATH. Remember that if you transfer the file between machines you should make sure you do so in binary.

1.3.2. z/OS Installation Instructions

Once unzipped, transfer the ***MQMONA.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 MQMONA.SEQ
```

Once the ***MQMONA.SEQ*** file is successfully FTPed to your z/OS system, from TSO issue the following:-

```
receive inds(MQMONA.SEQ)
```

When prompted for a filename, reply

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

MQMONA can be run on z/OS in BATCH (including as a Started Task). Example pieces of JCL are provided in the zip file. MQMONA 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).

1.3.3. z/OS UNIX Installation Instructions

If you wish to run MQMONA in z/OS UNIX, you can copy the MVS executable module that you have installed in the previous section, to a z/OS UNIX executable file using the following command.

```
TSO OPUT 'GEMUSER.USER.LOAD(MQMONA)' '/u/gemuser/bin/mqmona' BIN
```

Chapter 2. Licensing

MQMONA is licensed as a result of your existing MO71, MQSCX and/or MQEV licenses. Whichever applications you want to use with MQMONA, ensure the MQMONA program can find the licences for those programs.

2.1. Licence File Location

When you bought an MQGem Software licence, you were sent an ***mqgem.lic*** file. All you need to do is place this licence file in the appropriate place for the MQMONA program to find it as detailed in the table.

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

Alternatively you can set environment variable **MQGEML**, or configuration file attribute `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.

2.2. Multiple licences

If you have multiple licences, for example one for MO71, MQSCX and one for MQEV, then they must be concatenated into a single **mqgem.lic** file for MQMONA to find. This can be done using simple OS commands such as **copy** or by using your favourite editor.

2.3. Reporting found licences

MQMONA will report which licenses it found, and thus which programs will be able to use the MQMONA agent. You will see output like the following when MQMONA starts up.

```
[13:38:42] MQGA221I MO71 Licence details: Valid - expires 09/09/26
[13:38:42] MQGA221I MQSCX Licence details: Valid - expires 09/09/26
[13:38:42] MQGA221I MQEV Licence details: Valid - expires 09/09/26
```

If no licence file is found, then instead you will see output like this when MQMONA starts up. In this situation any command messages that arrived on the MQMONA command queue will be forwarded on to the IBM MQ Command Server queue and all MQMONA functionality will be switched off.

```
[02:39:05] MQGA220E No valid licence found
```

Chapter 3. Getting Started

Running MQMONA is quite simple. It needs one new queue definition in order to operate. By default it will try to use a queue named **MQGEM.MQMONA.COMMAND.QUEUE** as its input queue. If you prefer to have a differently named queue, you can do that and provide its name with the `-q` parameter flag. For this chapter we will define the default queue.

```
DEFINE QLOCAL(MQGEM.MQMONA.COMMAND.QUEUE)
       DESCR('MQGem Command Server Agent queue')
```

Now issue the following command to run MQMONA against a queue manager called MQG1.

```
mqmona -m MQG1
```

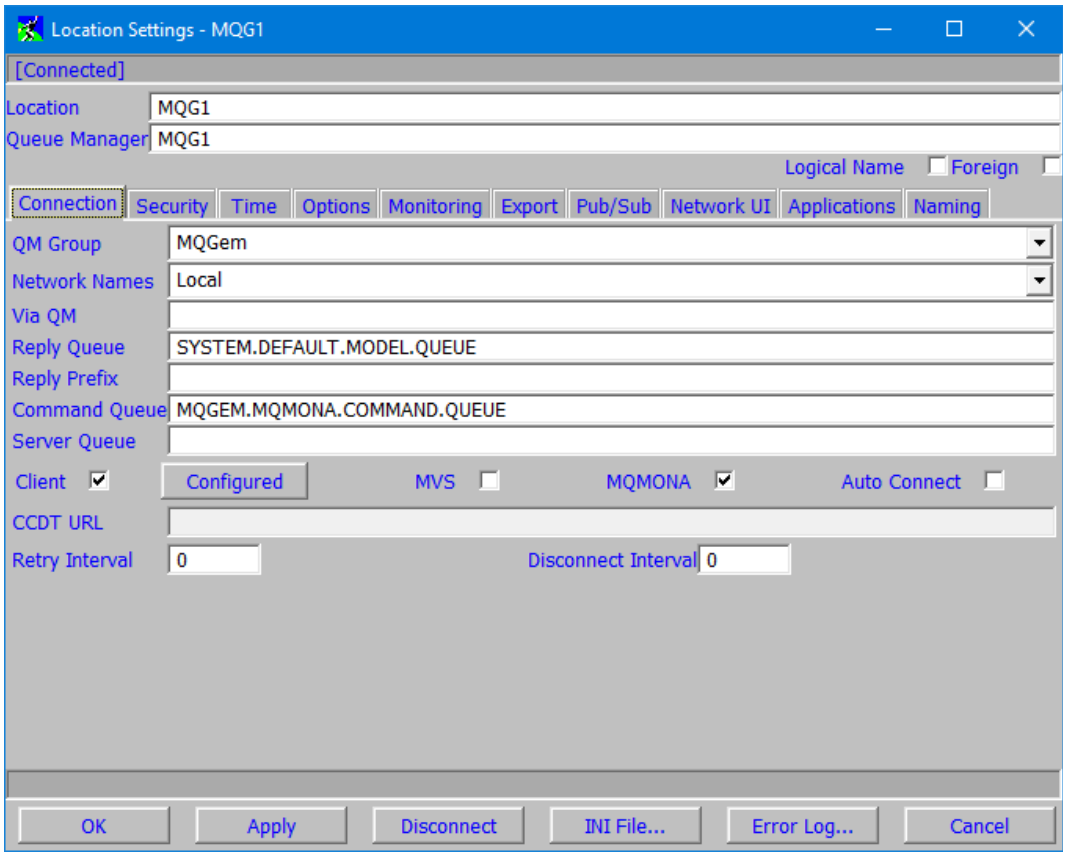
You will see a number of informational messages to show you that MQMONA has got up and running, and the names of the various queues it is using.

```
[13:38:42] MQGA001I MQGem Software MQMONA : Command Server Agent for IBM MQ
  Version V9.4.1 Build Date:Apr 30 2026
  (C) Copyright MQGem Software 2023,2026. (https://www.mqgem.com)
[13:38:42] MQGA221I MO71 Licence details: Valid - expires 09/09/26
[13:38:42] MQGA221I MQSCX Licence details: Valid - expires 09/09/26
[13:38:42] MQGA221I MQEV Licence details: Valid - expires 09/09/26
[13:38:42] MQGA004I Parameters in effect:
  Queue Manager         : MQG1
  Input Queue           : MQGEM.MQMONA.COMMAND.QUEUE
  Reply Queue           : SYSTEM.DEFAULT.MODEL.QUEUE
  Actual Reply Queue    : MQGEM.MQMONA.6301D47321206C01
  Verbose Level         : Low detail
```

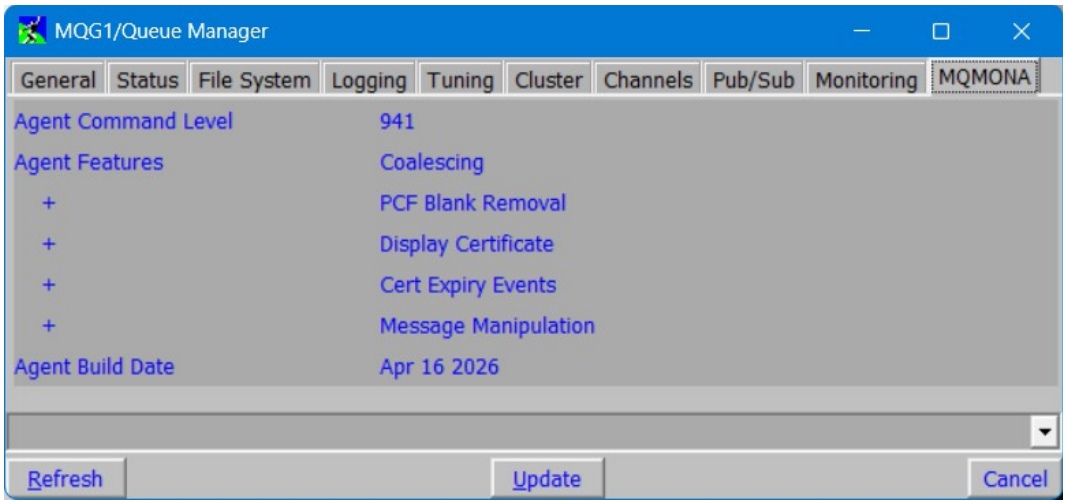
You can see the aforementioned input queue, `MQGEM.MQMONA.COMMAND.QUEUE`, and the reply queue it is using is a MODEL queue, the `SYSTEM.DEFAULT.MODEL.QUEUE`. The actual name of the dynamic queue created as its reply queue is also shown with a name that begins with `MQGEM.MQMONA`.

3.1. Configuring MO71 to use MQMONA

To use MQMONA with MO71, you must be using MO71 at V9.3.1 or higher. In order to set up MO71 to use MQMONA, you need to make one small change to the location. Open the location dialog for the queue manager, and on the Connection tab, check the “MQMONA” check-box which will change the “Command Queue” field to **`MQGEM.MQMONA.COMMAND.QUEUE`**. Now press Apply and issue your commands through MO71 as normal.



When you connect to a queue manager using the MQMONA agent, viewing the queue manager dialog will show an additional tab with some MQMONA details.



3.2. Configuring MQSCX to use MQMONA

To use MQMONA with MQSCX, you must be using MQSCX at V9.3.1 or higher.

In order to set up MQSCX to use MQMONA, you can either add an extra parameter flag when you start up the program as follows:-

```
mqscx -m MQG1 -q MQGEM.MQMONA.COMMAND.QUEUE
```

Or alternatively you can use an =conn command like this.

```
=conn qm(MQG1) cmdq(MQGEM.MQMONA.COMMAND.QUEUE)
```

This can be very useful to put into a synonym along with client connection details and then you can quickly use the synonym instead of typing all this in. This is our recommended way of using MQMONA with MQSCX connections.

```
=syn(MQG1) cmd(=conn qm(MQG1) channel(MQGEM.SVRCONN)
conname(gemmvs1.mqgem.com(1414)) cmdq(MQGEM.MQMONA.COMMAND.QUEUE) compmsg(RLE))
```

Use of message compression is advised for channels where MQSC traffic rather than PCF traffic is used. For example, MO71 connected to a z/OS queue manager, or channels for MQSCX. See 5.3. String padding in MQSC on page 9 for a full explanation.

When you connect to a queue manager using the MQMONA agent, viewing the MQSCX help screen (using F1 or =show scrn(help) will show some MQMONA details.

```
Administering Queue Manager
---------------------------
Qm            : MQG1
Command Level: 945
Platform      : Unix
MQMONA Level  : 941
```

Chapter 4. Consolidation

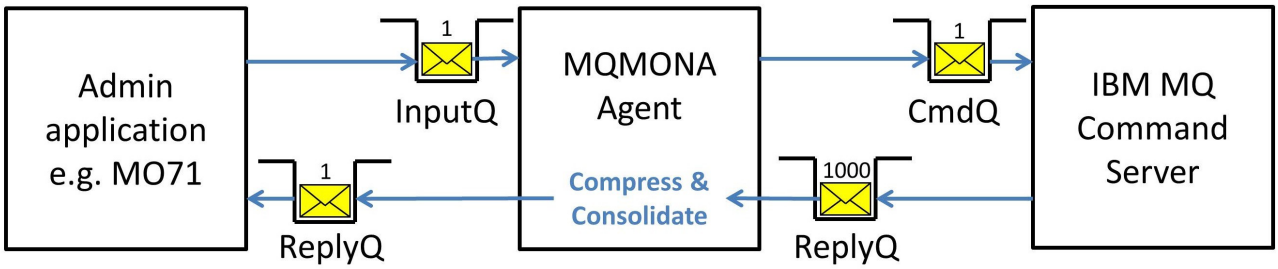
The MQMONA agent improves your IBM MQ administrative command experience when network latency is high, by consolidating the responses from the IBM MQ Command Server into fewer messages and thus reducing the number of MQGET calls an administrative application must make to retrieve all the answers.

4.1. IBM MQ Commands and Responses

When you issue a command to the IBM MQ Command Server, such as the MQSC command to display all queues, or the PCF command to Inquire Channel Status, you place a single command message on the `SYSTEM.ADMIN.COMMAND.QUEUE`[1], and provide the name of your ReplyToQ in the Message Descriptor. The IBM MQ Command Server sends the response messages that answer your command, to your nominated ReplyToQ.

The responses take the form of a single message for each item returned. This means if you displayed 1000 queues, your administrative application would be sent 1000 messages to retrieve from it's ReplyToQ.

Using MQMONA as an agent, means that your administrative application actually puts its command message onto the **`MQGEM.MQMONA.COMMAND.QUEUE`**, and MQMONA picks it up, changes the ReplyToQ and sends it on to the IBM MQ Command Server. This means that MQMONA receives the 1000 separate messages on its ReplyToQ, and when it does, it consolidates those reply messages together into several, or perhaps even just one larger reply message which it then puts to the original ReplyToQ for your administrative application.



By reducing the number of messages, we reduce the number of MQGET calls made over a client, which means we reduce the number of line turn-arounds, which are the expensive part of the system when on a network with high latency.

4.2. Consolidated Message Size

When MQMONA compresses and consolidates the responses from the IBM MQ Command Server into fewer, larger messages, it will keep the total size below a configured maximum. This is 1MB by default. You can change this by using configuration file setting **`maxmsgsize`** (see Chapter 18. Configuration File on page 55 for more details on how to set it).

For client connected MQ administrative applications, MQMONA will inquire the reply queue to discover its Maximum Message Length and if it is discovered to be smaller than the currently set **`maxmsgsize`** it will reduce the size used for this specific reply queue. Message **MQGA304I** will be written to the error log when this happens as this suggests that this specific MQ administrative application is not getting the full benefit of MQMONA consolidation.

[1] Or the SYSTEM.COMMAND.INPUT on z/OS

For MQ administrative applications that are connecting in “via” mode, the above check is not possible, and you must ensure that all queues along the route between MQMONA and the MQ administrative application have a Maximum Message Length that can accommodate messages up to the size specified by **`maxmsgsize`**.

Chapter 5. Compression

The MQMONA agent improves your IBM MQ administrative command experience when bandwidth is low, by removing unnecessary padding from your PCF response messages. The process of consolidation described in the previous chapter, also has a small compressive effect because the MQCFH header is not repeated for every response when they are consolidated into a single message. The MQCFH header is only 36 bytes long, but that's an additional saving of 30K when displaying 1000 queues.

5.1. IBM MQ string padding

The string fields in the response messages to an IBM MQ command are generally blank padded to their full length. Exceptions to this, thankfully, are the following 10K long fields.

- MQCA_TOPIC_STRING
- MQCACF_SUBSCRIPTION_NAME
- MQCACF_SUBSCRIPTION_USER_DATA
- MQCACF_SUB_SELECTOR

Object name fields are mostly 48 characters in length, but most people have queue names that are far shorter than that. The response message from the IBM MQ Command Server will always contain 48 characters even if over half of them are blanks.

5.2. MQMONA compression

MQMONA removes all these blanks at the end of strings, and adjusts the lengths in PCF messages accordingly. This has a fairly major compression impact. Consider the length of the response messages to just display the SYSTEM queues on a V9.3.0 queue manager.

The command server responds with 55 reply messages with a total size of 11.2 KB, which MQMONA consolidates and compresses into a single response message to the original application, with a total size of 6.86 KB. This gives a compression rate of 38%.

Compression rates will be higher when the responses have a high percentage of string fields. So, as an example, if you display all the SYSTEM queues and ask for the Backout Requeue Name; Cluster; Description; Process Name; and Stream Queue Name, then you get 55 reply messages with a total size of 26.27 KB, which MQMONA consolidates and compresses into a single response message with a total size of 12.9KB, a compression rate of 50%.

5.3. String padding in MQSC

MQSC response messages from z/OS are fully blank padded too. However, because the responses to MQSC commands are not quoted, these blanks are actually important to parsing the output. For this reason MQMONA does not strip off the blanks from MQSC responses messages, and we advise the use of message compression on client connections to reduce the client payload further.

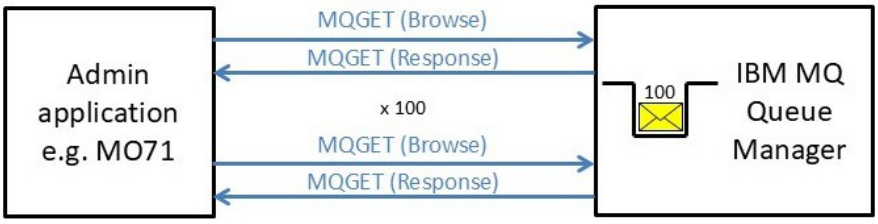
```
ALTER CHANNEL(MQGEM.SVRCONN) CHLTYPE(SVRCONN) COMPMSG(RLE)
```

Chapter 6. Message Manipulation

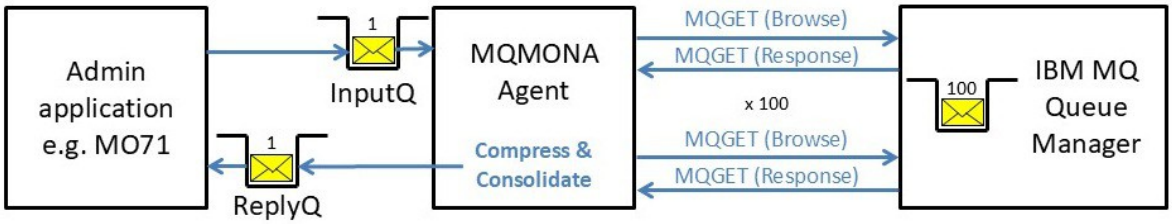
The MQMONA agent improves your IBM MQ administrative command experience when network latency is high and bandwidth is low, by providing an interface to browse and manipulate messages on queues without the overhead of many separate requests to the queue manager.

6.1. Browsing a Queue

When you browse a queue using an IBM MQ administrative tool, a number of MQGET calls, using the MQGMO_BROWSE options are made between the program and the Queue Manager. The more messages that are on the queue, the more MQGET calls must be made.



However, when using MQMONA to do this, a single command message is put on the command queue. The MQMONA Agent, which is running locally with the queue manager, issues the many MQGET calls that are needed, and then packages up the answers into a few or maybe even only one response message.

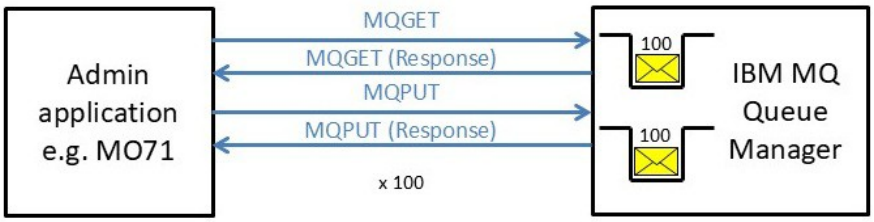


By reducing the number of MQGET calls that are done across the network, the whole process is sped up. Additionally, only the fields of the message the application is interested in seeing are returned in the response message. This reduces the number of bytes of data sent back to the application. After all a full MQMD is well over 400 bytes.

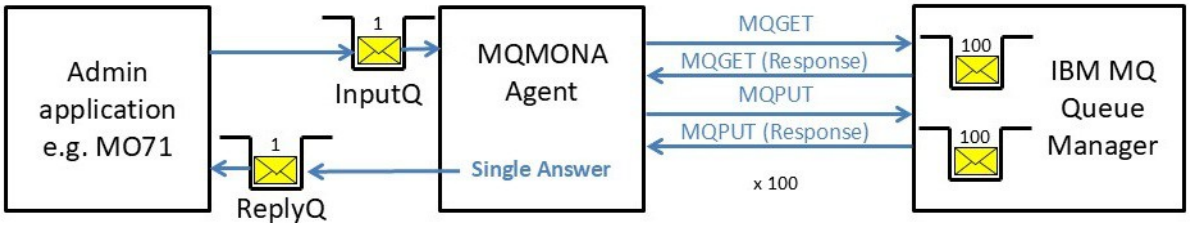
The number of messages returned to the application can further be reduced by having some filtering applied with a search string. When a search string is applied at the admin application, it reduces what is displayed, but does not reduce what is sent to the admin application. When a search string is applied using MQMONA, it reduces the data sent back to the admin application and thus speeds up the task.

6.2. Copying or Moving messages

When you need to copy or move messages from one queue to another, using standard IBM MQ administrative tools, all the messages are sent across the network to the program and then sent back across from the program to the queue manager to be put onto the target queue.



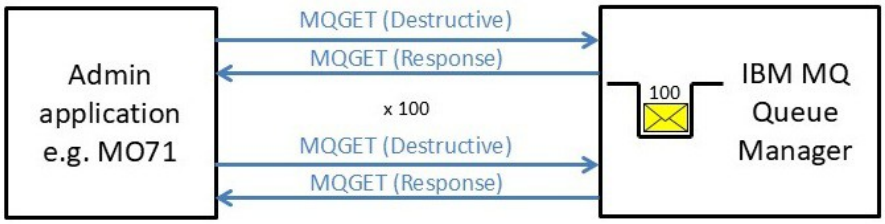
However, when using MQMONA to do this, a single command message is put on the command queue. The MQMONA Agent, which is running locally with the queue manager, issues the many MQGET and MQPUT calls to copy or move the messages from one queue to the other, before sending a single short response message back to the application.



It is quite clear that this massively reduces the amount of data sent across the client link, and therefore also speeds up the task.

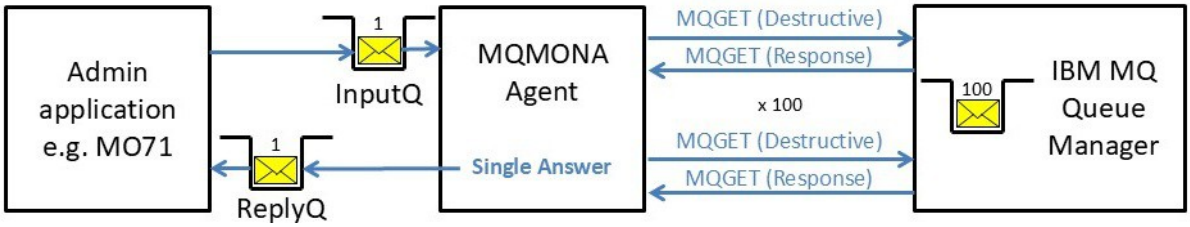
6.3. Deleting messages

When you delete messages from a queue using an IBM MQ administrative tool, a number of MQGET calls, are made between the program and the Queue Manager. The more messages that are on the queue, the more MQGET calls must be made. While many tools mitigate the network bandwidth issue by requesting zero bytes of the message when they do this, there are still as many MQGET calls as there are messages to delete.



Sometimes it is possible to use the `CLEAR QLOCAL` command, if you need to delete all the messages, and no other application has the queue open, but when that is not possible, using MQMONA can improve this situation too.

When using MQMONA to do this, a single command message is put on the command queue. The MQMONA Agent, which is running locally with the queue manager, issues the many MQGET calls to delete the messages from the queue, before sending a single short response message back to the application.



It is also worth considering the task of deleting only certain messages off a queue. This can be done with a single **Delete Messages** command, unlike the `CLEAR QLOCAL` command which removes all the messages.

6.4. Security

All MQGETs and MQPUTs involved are done under the authority of the admin application requesting the operation, and not under the authority of the MQMONA agent. MQMONA makes use of `MQOO_ALTERNATE_USER_AUTHORITY` when doing the tasks.

Chapter 7. Certificate Events

MQMONA allows you to issue a DISPLAY CERT command which will display the certificates in your repository. However, it is also useful for MQMONA to notify you of their expiry dates automatically and that's where Certificate Events come in. The MQMONA agent can detect conditions relating to the certificates in your queue manager key repository and raise events to notify you of these conditions.

7.1. Certificates Nearing Expiry

When a certificate expires, it will result in an error and a channel may fail to start. It is therefore desirable to be able to detect this condition before it happens, and ensure a new certificate is obtained and put into place.

The MQMONA agent can be configured to regularly inspect the contents of your key repository to see if any of the certificates are going to expire soon.

This feature requires the MQMONA agent to be able to detect an MQEV licence file on your system. See Chapter 2. Licensing on page 3 for how to ensure MQMONA can find your MQGem licences.

Assuming that an MQEV licence is found, this feature is on by default using the following settings. You can change these settings in your configuration file. See Chapter 18. Configuration File on page 55 for full details on using a configuration file.

Configuration Attribute	Default Value	Notes
certevent	asqmgr	The creation of certificate event messages is dependant on the queue manager attribute SSLEV being set to ENABLED.
certfreq	24 (hours)	The MQMONA agent will inspect the contents of the queue manager's key repository every 24 hours, with the initial inspection taking place each time the agent starts up.
certthresh	30 (days)	Any certificates found in the queue manager's key repository with a lifespan less than 30 days will have an event message written about them.

The MQMONA agent will write out message **MQGA224I** each time it checks the contents of the key repository, and will write out message **MQGA228I** if it finds a certificate to write an event message about. See 10.1.Channel SSL Certificate Nearing Expiry on page 42 for full details of the event message written.

```
MQGA224I Checking Certificates in '/var/mqm/qmgrs/MQG1/ssl/key.kdb' for Expiry
MQGA228I Certificate event written for DN 'CN=MQG1,O=MQGem' with 29 days left
```

7.2. Security

In order to put the event message, MQMONA will need authority to put to the event queue.

7.2.1. Example security commands for Distributed Platforms

Suppose you have a user ID, in the group MQMONA, which will run the MQMONA program. Below is the additional authority needed if you use this feature.

```
SET AUTHREC PROFILE(SYSTEM.ADMIN.CHANNEL.EVENT) GROUP('MQMONA') AUTHADD(PUT)
```

7.2.2. Example security commands for z/OS

Suppose you have a user ID, in the group MQMONA, which will run the MQMONA program. Below is the additional authority needed if you use this feature. It assumes 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.

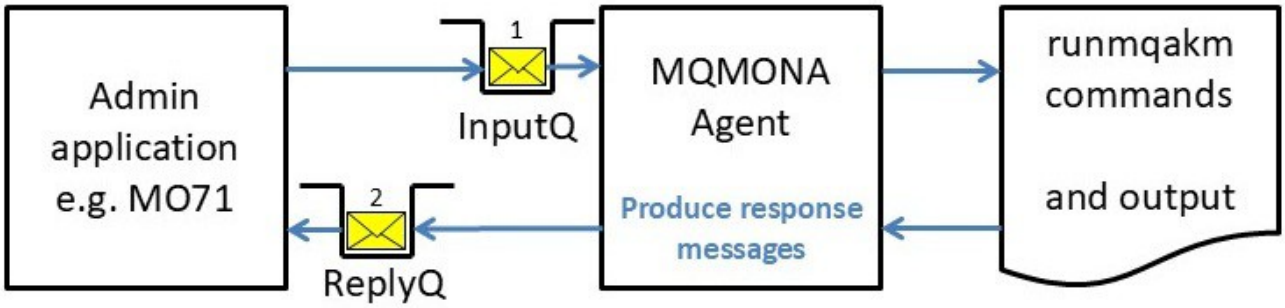
```
PERMIT qmgr.SYSTEM.ADMIN.CHANNEL.EVENT) CLASS(MQQUEUE) ID(MQMONA) ACCESS(UPDATE)
```

7.3. Inspection method

The method used by MQMONA to inspect the queue manager's key repository depends on the platform where it is running. However, regardless of which platform you are using, it is essential that MQMONA is running local to the queue manager in order for this to be successful. If it is not running locally, error message **MQGA122I** will be issued, and this feature will be disabled.

```
MQGA112I A client-connection was made using channel 'MQGEM.SVRCONN'. Certificate
Event checking disabled
```

7.3.1. Distributed platforms – runmqakm command



On distributed platforms, MQMONA issues `runmqakm` commands and parses the output. As a result the user ID that MQMONA runs under will need to have access to the key repository being queried. It is likely that MQMONA is running as an MQ Service object (see Chapter 11. Running MQMONA as a SERVICE on page 46) and so will be running as an mqm group user id which will give it the access it needs. However, if you run it under some other user id, then ensure it is provided access to run a `runmqakm` command, like the following two examples.

```
runmqakm -cert -list all -db queue-managers-key-repository -stashed -locale en
```

```
runmqakm -cert -details -label label-from-previous-command -db queue-managers-
key-repository -stashed -locale en
```

The `-locale en` parameter on this command ensures that, regardless of the language your IBM MQ installation is using, the output from these commands that MQMONA has to parse, is in English.

The date and time output returned from `runmqakm`, sometimes shows a time zone in a format of `GMT+nn:nn` and sometimes shows a time zone abbreviation like `EDT` or `NZDT`. If the latter is found, and it is a time zone abbreviation that MQMONA does not know about, it will compare it to the local time zone where it is running, to see if it can find the time zone bias that way.

Error messages **MQGA231I** or **MQGA232E** will be reported for any unknown time zone encountered (once per found time zone). Please report these to MQGem Software support if you see them.

```
MQGA232E Time zone in runmqakm output date string not recognised
Output: 'November 28, 2024 12:40:47 PM NZDT'
```

7.3.2. z/OS – list keyring callable service

On z/OS, MQMONA calls the `R_datalib` callable service (`IRRSDL64`) using the `READ` functions, `DataGetFirst` and `DataGetNext`. To be able to query the contents of the queue manager's keyring, the user ID MQMONA is running under needs access to the keyring owned by the CHINIT user id. Examples of the access needed are shown below, and full details of how these calls are protected can be read in z/OS Security Server RACF Callable Services > R_datalib > RACF authorization.

In the following examples, the CHINIT user ID is `MQG1USR` and the keyring is `MQG1RING`. The MQMONA agent is running under a different user ID called `MQG1AGT`.

Ring-specific profile checking

If there is a matching profile called `MQG1USR.MQG1RING.LST` in the `RDATALIB` class then `MQG1AGT` requires `READ` access.

```
PERMIT MQG1USR.MQG1RING.LST CLASS(RDATALIB) ID(MQG1AGT) ACCESS(READ)
```

Global profile checking

If there is no such matching profile then global profile checking will be used instead. This would then mean `MQG1AGT` requires `UPDATE` access to the `IRR.DICTCERT.LISTRING` profile in the `FACILITY` class. It needs `UPDATE` rather than `READ` here because it is not the owner of the keyring.

```
PERMIT IRR.DICTCERT.LISTRING CLASS(FACILITY) ID(MQG1AGT) ACCESS(READ)
```

Chapter 8. Error Log Command

MQMONA implements a form of the **MQCMD_INQUIRE_ERROR_LOG** command which allows a remote Administration program, such as MO71, to display the error log. Of course you might well ask why we went to the bother of implementing our own version of the command. Essentially there are three reasons:

1. **IBM MQ Version**
   The command was introduced in IBM MQ 9.4.5 which means that anyone on an earlier Queue Manager is missing out. Having said that MQMONA does require the 'new' version of Error Log format and therefore it does require the Queue Manager to be at least IBM MQ Version 9.0.3
2. **Performance**
   In our experience the command does not perform particularly well and can take many seconds to return a response.
3. **National Language Support**
   The Queue Manager Command Server version of this command assumes that the error log is in the same code page as the Queue Manager. Often, of course, this will be the case. However, for extra flexibility the MQMONA implementation allows you to specify the code page. The other command change is that the message text is sent to the request unconverted. This gives greater control and flexibility to the requester about how and when the data is converted.

There are other reasons but that is looking further down the line. For example, if this is a commonly used feature then we may well enhance the command to allow for greater choice in message selection.

Of course, in general you don't need to know about the command, what it does or how to use it. All you need decide is whether you want MQMONA, rather than the Queue Manager, to process this particular command. And that comes down to MQ Version and whether you are having any performance issues.

8.1. Configuration

By default MQMONA **will not** process the **MQCMD_INQUIRE_ERROR_LOG**. Instead the command will be passed through to the queue manager where it will either be processed or rejected depending on the MQ version. In order to switch it on in MQMONA you need a line in your configuration file such as the following:

```
mqerlCmd = yes
```

There are other configuration parameters which can be specified to modify the behaviour and the full list of parameters can be found listed in Chapter 18. Configuration File on page 55. All the parameters start with the prefix '**mqerl**'.

8.2. Security

If MQMONA processes the **MQCMD_INQUIRE_ERROR_LOG** command, the user issuing it requires **+set** authority on the MQMONA command queue. See 9.2. Inquire Error Log on page 25.

8.3. Error Log Location

By default MQMONA will look for the Error Logs in the following location:

```
<MQ_DATA_PATH>\qmgrs\<Queue Manager Name>\errors\AMQERR01.LOG etc
```

The <Queue Manager Name> portion is adjusted as per the rules in IBM Docs for queue manager name mangling. The most common adjustment is to change dots into exclamation points.

The <MQ_DATA_PATH> portion is resolved by either looking in the registry (on Windows) or default location "/var/mqm" (on Unix). This can be overridden, if you like, by setting the environment variable **MQ_DATA_PATH**.

Alternatively, if you want to be totally explicit about the location, you can set the value of **mqerlPath** in the configuration file, to the path where the error log files can be found. To be completely clear, you must provide the full path up to but not including the AMQERR0x.LOG file name. In this case, if any name mangling is required to the queue manager name portion of the location, you must handle that yourself.

8.4. Performance

As with any software its performance can significantly affect the user experience. Slow performance can lead to frustration and dissatisfaction. With that in mind considerable effort has been made to try and ensure that the feature is responsive as it can be. However, there are a few things to bear in mind.

8.4.1. Error Log Size

IBM MQ Error Logs can get pretty big. Very big in fact. It is possible to have a number of Gigabytes in your Errors Logs, storing millions of error messages. As a consequence, a single **MQCMD_INQUIRE_ERROR_LOG** could result in Gigabytes of data being returned. In general this is not a good thing!

Of course normally Error Logs aren't normally this big but there are still a couple of things to consider:

1. **Be sensible in what you ask for.**
   It sounds trite but asking for years worth of messages without a good reason is just a waste of time and CPU. Be selective it what you ask for and work back from the newest message rather than forwards from the oldest.
2. **Set the value of mqerlRespLimit to a sensible value**
   If using MQMONA at the back end, you can configure a limit to how much data can be returned in a single command.
   By default this is set to 100MB so for the majority of cases the command should complete just fine. One could make a strong case that even this is too large and should be reduced. However, even at the default value, it does protect you against accidental requests for huge amounts of data.

8.4.2. Command Speed

If, like us, you find the delay of even a few seconds between clicking the button and seeing the result quite frustrating then you may have your patience tested by the Error Log command. In our experience it can regularly take many seconds to return and, in extreme cases, can take multiple minutes if the Error Log is large. We talk about this in a little more detail in the 'Error Log Viewing' section of our Performance Comparisons chapter.

Clearly your mileage may vary and any experience we may have had with the GA version of IBM MQ 9.4.5 may well change in the future. However, so far our experience has been a little counter intuitive. Imagine that you have a large error log file spread out evenly throughout a year. What we found is that requests for messages early in the year responded far faster than for messages later in the year. The further through the file you request the slower it is. This is somewhat unfortunate, since a common query might be something like "*show me the messages created in the last two hours*". It is very rare that you want to see an entire years worth of messages, what you normally want to see is what just happened. So, the common use pattern has the worst performance.

For this reason we have written our own implementation of the command which is more suited to usage with MO71. We strongly recommend that you enable the Error Log support in MQMONA if you want to use this command from MO71.

Of course we should add at this point that there is nothing magic here. Even with MQMONA responding to the command, a request to see hundreds of megabytes of data will still take a long time and use a lot of bandwidth. We'd be interested to hear your experience though, and whether we could improve things further.

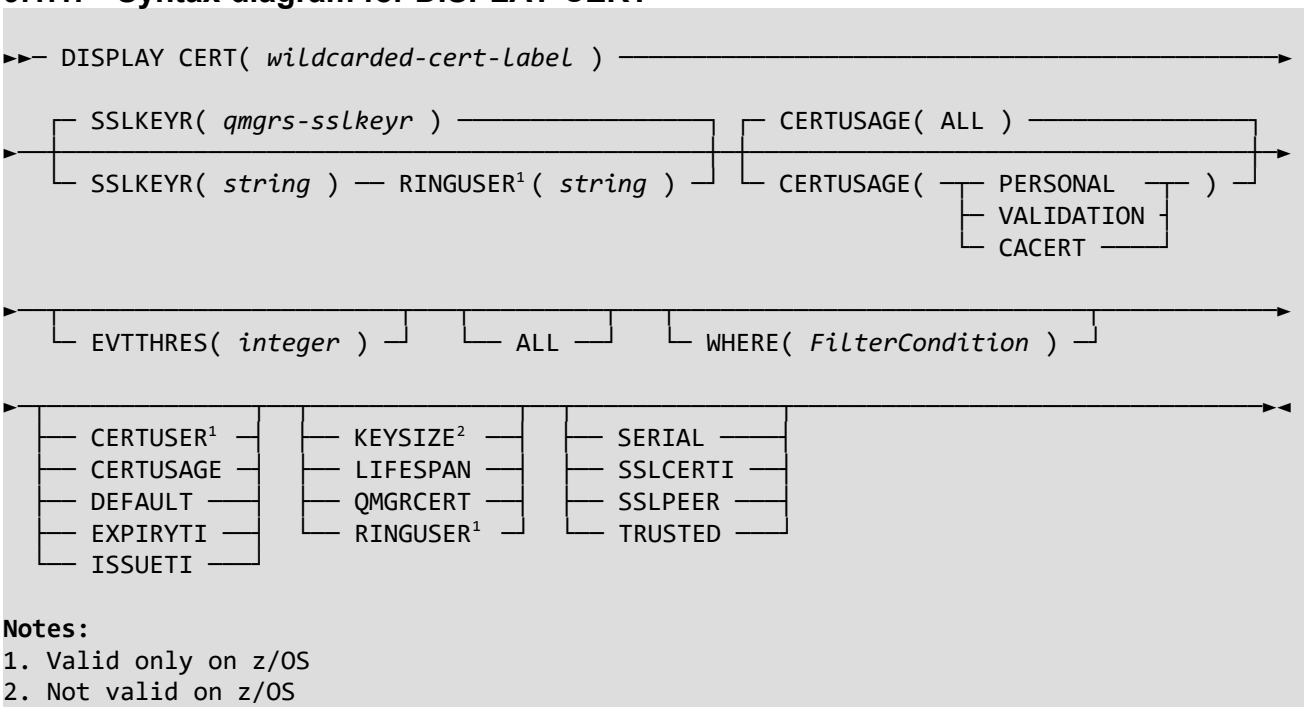
Chapter 9. Command Reference

In addition to forwarding commands to the IBM MQ Command Server, the MQMONA agent accepts some commands of its own. This chapter describes the MQSC and PCF commands that can be issued to MQMONA.

9.1. DISPLAY CERT

Use the MQSC command **DISPLAY CERT** (or it's equivalent PCF command **MQG_CMD_DISPLAY_CERTIFICATE**) to display the details of certificates in a distributed queue manager key repository or a z/OS queue manager key ring. The availability of this command can be turned off using the `certdisplay` switch in the configuration file. See Chapter 18. Configuration File on page 55 for details.

9.1.1. Syntax diagram for DISPLAY CERT



Notes:
1. Valid only on z/OS
2. Not valid on z/OS

9.1.2. Parameter descriptions for DISPLAY CERT

(wildcarded-cert-label)

The certificate labels to be displayed. This can be a wildcarded string.

The maximum length of this string is `MQ_CERT_LABEL_LENGTH` (64).

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

SSLKEYR

This parameter is used to specify which key repository you wish to see data from.

On z/OS this is a keyring name. On distributed platforms this can be a key database file (KDB) or a PKCS#12 file.

The maximum length of this string is `MQ_SSL_KEY_REPOSITORY_LENGTH` (256).

If it is not specified, the output shows certificates from the key repository named in the queue manager attribute SSLKEYR.

See 9.1.8. Default Key Repository file extension on page 25 for behaviour when the file extension is not supplied either in this attribute or in the queue manager SSLKEYR attribute.

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

RINGUSER

If specifying the name of a keyring on z/OS, rather than displaying certificates from the keyring named in the queue manager SSLKEYR attribute, then this field must also be supplied to provide the owner user id for the keyring. This attribute is only applicable on z/OS.

The maximum length of this string is MQG_ZOS_USER_ID_LENGTH (8).

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

CERTUSAGE

Which certificates to return. When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_MQMONA_CERT_USAGE.

Possible values are shown below. For fuller descriptions of the meaning of these values, see the output description further down.

ALL

All certificates will be returned.

The PCF value for this is MQG_CERT_USAGE_ALL.

PERSONAL

Only Personal Certificates will be returned. These will be certificates that have a private key, and are used to identify entities such as your queue manager.

The PCF value for this is MQG_CERT_USAGE_PERSONAL.

VALIDATION

Only Validation Certificates will be returned. These are certificates that are neither a personal certificate nor a CA certificate. These will include ones that have been added to the Key Repository to validate an inbound remote certificate. Self-signed certificates of other queue managers will show up in this category.

The PCF value for this is MQG_CERT_USAGE_VALIDATION.

CACERT

Only Certificate Authority (CA) Certificates will be returned.

The PCF value for this is MQG_CERT_USAGE_CACERT.

EVTTHRES

This is an optional parameter which allows you to generate certificate events at the time the DISPLAY command is issued.

The lifespan threshold below which an event will be generated for any displayed certificates with a remaining lifespan less than the number of days specified. The value must be between 0 and 9999 days.

For details of the event message, see 10.1. Channel SSL Certificate Nearing Expiry on page 42.

The event message, if needed, will only be generated if the MQMONA program can locate an MQEV licence. It will ignore any configuration for background certificate event generation including the queue manager SSLEV switch and will generate the event regardless. This allows you to test the event messages generated without enabling any other settings.

If a lifespan threshold is not specified, the display command will not cause any event messages to be generated.

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

WHERE

Specify a filter condition to only display records that satisfy the selection criterion of the filter condition.

When filtering on an integer field, some fields allow units as noted in their descriptions below. See the examples later on as well.

When using the PCF interface, this is an MQCFST parameter containing the MQSC style `WHERE` clause string with identifier `MQG_ATTR_WHERE`.

9.1.3. Always returned

The following parameters are always returned.

CERT

The certificate label. The maximum length of this string is `MQ_CERT_LABEL_LENGTH` (64).

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

CERTUSAGE

Whether the certificate is a Personal or Certificate Authority (CA) certificate. In some cases it is neither. When using the PCF interface, this is an MQCFIN parameter with identifier `MQG_ATTR_MQMONA_CERT_USAGE`.

Possible values are:

PERSONAL

This certificate is a Personal Certificate.

On distributed platforms, this is a certificate that `runmqakm` considers to be `personal`. It will be one that has a private key in the Key Repository.

On the z/OS platform, this is what the `R_datalib` output describes as a “Certificate_Usage” of “Personal”.

The PCF value for this is `MQG_CERT_USAGE_PERSONAL`.

VALIDATION

This certificate is neither a Personal Certificate, nor a CA Certificate.

It will be one that has been added to the Key Repository to validate an inbound remote certificate. Self-signed certificates of other queue managers will show up in this category.

On distributed platforms, this is a certificate that `runmqakm` shows when listing `ca` certificates rather than `personal` certificates, but does not have the “ca=true” `basicConstraints` extension.

On the z/OS platform, this is sometimes referred to as a SITE certificate. It is what the `R_datalib` output describes as a “Certificate_Usage” of “Other (site)”.

The PCF value for this is `MQG_CERT_USAGE_VALIDATION`.

CACERT

This certificate is a CA Certificate.

On distributed platforms, this is a certificate that has the “ca=true” `basicConstraints` extension.

On the z/OS platform, this is what the `R_datalib` output describes as a “Certificate_Usage” of “Certauth”.

The PCF value for this is `MQG_CERT_USAGE_CACERT`.

LIFESPAN

This is the remaining life span of the certificate measured in seconds.

When filtering on this field using a `WHERE` clause, time-based units can be used, for example, instead of providing a number of seconds, you can instead provide a number of days and follow the number with the word “days”. See the examples in 9.1.6. Example invocations on page 24.

When using the PCF interface, this is an MQCFIN64 parameter with identifier `MQG_ATTR_MQMONA_CERT_LIFESPAN`.

9.1.4. Requested parameters

In addition to the above listed parameters that are always returned, you can request the following information about a certificate.

CERTUSER

The owning user ID of the certificate. This attribute is only applicable on z/OS.

The maximum length of this string is `MQG_ZOS_USER_ID_LENGTH` (8).

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

DEFAULT

Whether the certificate is marked as the default certificate in this key repository. When using the PCF interface, this is an MQCFIN parameter with identifier `MQG_ATTR_MQMONA_CERT_DEFAULT`.

Possible values are:

YES

This certificate is the default. The PCF value for this is `MQG_DEFAULT_CERT_YES`.

NO

This certificate is not the default. The PCF value for this is `MQG_DEFAULT_CERT_NO`.

EXPIRYTI

This is the date and time the certificate will expire.

When using the MQSC interface, this is formatted as `YYYY-MM-DD HH:MM:SS (TZ)`.

When using the PCF interface, this is an MQCFIN64 parameter containing an integer representation[2] of the date and time with identifier MQG_ATTR_MQMONA_CERT_EXPIRY_TIME.

When filtering on this field using a WHERE clause, ensure comparison is done using this integer representation.

ISSUETI

This is date and time the certificate was issued.

When using the MQSC interface, this is formatted as YYYY-MM-DD HH:MM:SS (TZ).

When using the PCF interface, this is an MQCFIN64 parameter containing an integer representation[2] of the date and time with identifier MQG_ATTR_MQMONA_CERT_ISSUE_TIME.

When filtering on this field using a WHERE clause, ensure comparison is done using this integer representation.

KEYSIZE

This is the size of the certificate key. When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_MQMONA_CERT_KEYSIZE.

This attribute is not applicable on z/OS.

QMGRCERT

Whether the certificate is the queue manager's certificate. This is discovered by comparing the label of the certificate with the label supplied in the queue manager attribute CERTLABL.

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

Possible values are:

YES

This certificate is the queue manager's Certificate. The PCF value for this is MQG_QMGR_CERT_YES.

NO

This certificate is not the queue manager's Certificate. The PCF value for this is MQG_QMGR_CERT_NO.

SERIAL

The serial number of the certificate. When using the PCF interface, this is an MQCFST parameter with identifier MQG_ATTR_MQMONA_CERT_SERIAL.

The maximum length of this string is MQG_CERT_SERIAL_LENGTH (64).

On distributed platforms this is shown as a series of pairs of hexadecimal digits separated by colon characters ':' to match the output seen in IBM MQ DISPLAY CHSTATUS displays.

SSLCERTI

The Issuer's Distinguished Name of the certificate. When using the PCF interface, this is an MQCFST parameter with identifier MQCACH_SSL_CERT_ISSUER_NAME.

The maximum length of this string is MQ_SSL_PEER_NAME_LENGTH (1024).

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

SSLPEER

The Subject's Distinguished Name of the certificate. When using the PCF interface, this is an MQCFST parameter with identifier MQCACH_SSL_PEER_NAME.

The maximum length of this string is MQ_SSL_PEER_NAME_LENGTH (1024).

TRUSTED

Whether the certificate is trusted. When using the PCF interface, this is an MQCFIN parameter with identifier MQG_ATTR_MQMONA_CERT_TRUSTED.

Possible values are:

YES

This certificate is trusted. The PCF value for this is MQG_TRUSTED_CERT_YES.

NO

This certificate is not trusted. The PCF value for this is MQG_TRUSTED_CERT_NO.

HIGH

This certificate is highly trusted. The PCF value for this is MQG_TRUSTED_CERT_HIGH. This value is only applicable on z/OS.

9.1.5. Error Codes

When using the PCF interface, this command might return the following error codes in the response format header, in addition to those codes listed in Error codes applicable to all commands.

Reason (MQLONG)

The value can be any of the following values:

MQRC_KEY_REPOSITORY_ERROR

(2381, X'94D') SSL/TLS Key Repository error

Additional information will be recorded in the MQMONA error log file with more details about the issue.

9.1.6. Example invocations

Issue the following command to find all self-signed certificates in the queue manager's key repository.

```
DISPLAY CERT(*) WHERE(SSLPEER EQ SSLCERTI) ALL
```

Issue the following command to find all certificates that will expire in less than 30 days time.

```
DISPLAY CERT(*) WHERE(LIFESPAN LT 30d) ALL
```

Issue the following command to find all CA certificates that will expire in less than 90 days time.

```
DISPLAY CERT(*) CERTUSAGE(CACERT) WHERE(LIFESPAN LT 90d) ALL
```

Issue the following command to display the queue manager's certificate.

```
DISPLAY CERT(*) WHERE(QMGRCERT EQ YES) ALL
```

Find the CA certificate that validated a particular TLS protected running channel. MQSCX syntax for saving the value of a parameter from the output of one command and passing it in as input in another command is shown here.

```
DISPLAY CHSTATUS(MQGEM.SVRCONN) SSLCERTI
@signer = SSLCERTI
DISPLAY CERT(*) WHERE(SSLPEER EQ '<@signer>') ALL
```

9.1.7. Security Requirements

The user issuing the DISPLAY CERT command requires authority to issue a DISPLAY QMGR command.

- **On distributed this means**
  they require +dsp on the qmgr object.

- **On z/OS this means**
  they require READ access on the *hlq*.DISPLAY.QMGR MQCMDS profile.

9.1.8. Default Key Repository file extension

When the TLS Key Repository contains a file name, without an extension, the file used will depend on the queue manager's command level as shown in the table below.

This can either be in the queue manager's SSLKEYR attribute or in the DISPLAY CERT SSLKEYR attribute.

Command Level	Behaviour
Before 930	The file extension could not be supplied. It was always assumed to be .kdb
930 and later	If the file extension was not supplied, it was assumed to be .kdb. PKCS#12 files (.p12 file extension) could be used, but their file extension must be supplied.
944 and later	If the file extension was not supplied, it could be .p12 or .kdb. The queue manager first checks .p12 and if that doesn't exist, uses .kdb

9.2. Inquire Error Log

Use the PCF command MQCMD_INQUIRE_ERROR_LOG to retrieve the contents of the queue manager error log file. The availability of this command can be turned on using the mqerlCmd switch in the configuration file. See Chapter 18. Configuration File on page 55 for details.

This command is very similar in concept, although somewhat different, to the one documented for IBM MQ V9.4.5. See MQCMD_INQUIRE_ERROR_LOG (inquire error log) on Multiplatforms.

9.2.1. Security Requirements

The user issuing the Inquire Error Log (MQCMD_INQUIRE_ERROR_LOG) command requires +set access on the MQMONA command queue (normally MQGEM.MQMONA.COMMAND.QUEUE).

9.3. Inquire Messages

Use the PCF command MQG_CMD_INQUIRE_MESSAGES to list the details of the messages on the nominated queue. The availability of this command can be turned off using the messagecmds switch in the configuration file. See Chapter 18. Configuration File on page 55 for details.

9.3.1. Required Parameters

QueueName (MQCFST)

The name of the queue to browse messages on (parameter identifier: MQCA_Q_NAME).

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

9.3.2. Optional Parameters

LinkId (MQCFIN)

A unique ID to link together actions that might cause queues to be opened, so that they can all be closed together when work is completed (parameter identifier: MQG_ATTR_MQMONA_QUEUE_LINK_ID).

MaxMsgSize (MQCFIN)

The maximum size of message data to return on the command (parameter identifier: MQG_MAX_MSG_LENGTH).

MessageRange (MQCFST)

The numeric range of messages to return (parameter identifier: MQG_MSG_RANGE).

This is a character string with a maximum length of 16 characters, with the following format:

```
start,end
end
```

If only one number is provided, it is treated as the end number, and the range starts at the first message.

If two numbers are provided, they are used as the start and end numbers of the range of messages.

MessageOptions (MQCFIN)

Options to control the way messages are browsed on the queue (parameter identifier: MQG_MSG_OPTIONS). Possible values are shown below.

MQG_MO_NO_CONVERT

Do not data convert the messages. The default behaviour, if this option is not used, is to use MQGMO_CONVERT to convert the messages into the CCSID provided in the **ConvertCCSID** parameter.

MQG_MO_NO_PROPERTIES

Do not force messages properties into an MQRFH2 on the front of the message. The default behaviour, if this option is not used, is to use the MQGMO_PROPERTIES_FORCE_MQRFH2 option.

MQG_MO_NO_SEARCH_OFFSET

The default behaviour is to show the message summary starting at the found search string. If this option is used, the message summary will always show the start of the message. If no search string is in use, then this option has no effect.

ConvertCCSID (MQCFIN)

The CodedCharSetId to convert messages into (parameter identifier: `MQG_CONVERT_CCSID`).

If not specified, the value `MQCCSID_Q_MGR` (0) will be used.

ConvertEncoding (MQCFIN)

The Encoding to convert messages into (parameter identifier: `MQG_ATTR_MQMONA_CONVERT_ENCODING`).

If not specified, the value MQENC_NATIVE for the queue manager platform will be used.

MessageSearchString (MQCFST)

A string to search for in the message (parameter identifier: `MQG_MSG_SEARCH`).

Only messages that match the search string will be returned on the command.

MessageSelectorString (MQCFST)

An SQL92 expression to search for messages (parameter identifier: `MQG_MSG_SELECTOR`).

This string will be used in the `MQOD SelectionString` field. See IBM MQ documentation on this field to understand its contents.

MessageAttrs (MQCFIL)

Message Attributes (parameter identifier: `MQG_MSG_ATTRS`).

The parameter lists the attributes to be returned, which can be a combination of the following attributes. If this parameter is omitted, then all fields are returned. The table also shows the MQMD field that this represents.

Review the IBM MQ documentation on the field to understand the returned values.

Attribute	Description	MQMD Field
MQG_ATTR_REPORT	The report options set in the message	`Report`
MQG_ATTR_MSG_TYPE	The type of the message	`MsgType`
MQG_ATTR_EXPIRY	The remaining time to live of the message	`Expiry`
MQG_ATTR_FEEDBACK	The feedback code in the message	`Feedback`
MQG_ATTR_ENCODING	The encoding of the message	`Encoding`
MQG_ATTR_CODECHARSETID	The CCSID of the message	`CodedCharSetId`
MQG_ATTR_FORMAT	The message format	`Format`
MQG_ATTR_PRIORITY	The priority of the message	`Priority`
MQG_ATTR_PERSISTENCE	The persistence of the message	`Persistence`
MQG_ATTR_MSG_ID	The message ID of the message in hex	`MsgId`
MQG_ATTR_CHAR_MSG_ID	The message ID of the message as a string	`MsgId`
MQG_ATTR_CORREL_ID	The correlation ID of the message in hex	`CorrelId`
MQG_ATTR_CHAR_CORREL_ID	The correlation ID of the message as a string	`CorrelId`
MQG_ATTR_BACKOUT_COUNT	How often this message has been rolled back	`BackoutCount`

Attribute	Description	MQMD Field
MQG_ATTR_REPLYTOQ	The reply-to-queue in the message	ReplyToQ
MQG_ATTR_REPLYTOQMGR	The reply-to-queue manager in the message	ReplyToQMgr
MQG_ATTR_MSG_USER_ID	The user id in the message	UserIdentifier
MQG_ATTR_ACCOUNTING_TOKEN	The accounting token in the message	AccountingToken
MQG_ATTR_APPL_IDENTITY_DATA	Application data relating to identity	ApplIdentityData
MQG_ATTR_PUT_APPL_TYPE	The type of application that put the message	PutApplType
MQG_ATTR_PUT_APPL_NAME	The name of the application put the message	PutApplName
MQG_ATTR_PUT_DATE	The date on which the message was put	PutDate
MQG_ATTR_PUT_TIME	The time at which the message was put	PutTime
MQG_ATTR_APPL_ORIGIN_DATA	Application data relating to origin	ApplOriginData
MQG_ATTR_GROUP_ID	The Group ID of the message in hex	GroupID
MQG_ATTR_MSG_SEQ_NUMBER	Sequence number of logical message in group	MsgSeqNumber
MQG_ATTR_OFFSET	Offset of data in physical message from start of logical message	Offset
MQG_ATTR_MSG_FLAGS	Message flags	MsgFlags
MQG_ATTR_POSITION	Numeric position on the queue of the message	N/A
MQG_ATTR_MESSAGE	A small portion of the message data	N/A
MQG_ATTR_MSG_LENGTH	The length of the message	N/A
MQG_ATTR_MSG_TOKEN	The message token representing the message	N/A

9.3.3. Inquire Messages (Response)

The response to the Inquire Messages (MQG_CMD_INQUIRE_MESSAGES) command consists of the response header followed by attribute parameter structures for any requested attributes.

9.3.4. Response data

Review the table in the description of the **MessageAttrs** parameter to understand the output in the response message.

9.3.5. Progress Response Messages

If there are a lot of messages on the queue being browsed, this command could take some time, so progress responses messages will be sent back to update the requesting application on the progress of the command. This is controlled by the progdelay and progfreq attributes in the configuration file. See Chapter 18. Configuration File on page 55 for details.

For details of the contents of the response messages, see 9.9. Progress Message (Response) on page 40.

9.3.6. Error Codes

This command might return the following error codes in the response format header, in addition to those codes listed in Error codes applicable to all commands.

Reason (MQLONG)

The value can be any of the following values:

MQGRCCF_MAX_BROWSE_REACHED

(10101, X'2775') The maximum number of messages that can be browsed in one operation has been reached. This is controlled by the `maxbrwsmsgs` attribute in the configuration file. See Chapter 18. Configuration File on page 55 for details.

9.3.7. Security Requirements

The user issuing the Inquire Messages (`MQG_CMD_INQUIRE_MESSAGES`) command requires access to browse the queue.

- **On distributed this means**
  they require `+browse` on the `queue` being browsed.

- **On z/OS this means**
  they require READ access on the *`hlq`*`.`*`queuename`* `MQQUEUE` profile.

The user id under which MQMONA is running requires access to use alternate user ids.

- **On distributed this**
  means it requires `+altusr` on the `qmgr` object.

- **On z/OS this means**
  it requires UPDATE access on the *`hlq`*`.ALTERNATE.USER.`*`userid` `MQADMIN` profile.

9.4. Inquire Individual Message

Use one of the PCF commands

- MQG_CMD_INQUIRE_MESSAGE
- MQG_CMD_INQUIRE_MSG_NEXT_FAST
- MQG_CMD_INQUIRE_MSGDETAIL
- MQG_CMD_INQUIRE_MSGDETAIL_NEXT_FAST

to return the details of an individual messages on the nominated queue. The availability of these commands can be turned off using the messagecmds switch in the configuration file. See Chapter 18. Configuration File on page 55 for details.

These commands all operate in a very similar way with the following exceptions.

When using MQG_CMD_INQUIRE_MSG_NEXT_FAST or MQG_CMD_INQUIRE_MSGDETAIL_NEXT_FAST if you supply a message token, then the command will directly browse the message with that message token, and then will browse the next message (MQGMO_BROWSE_NEXT) and return that message to you. This is much more efficient than browsing to a particular position on a queue, and is the recommended command to use to step to the next message on a queue.

9.4.1. Required Parameters

QueueName (MQCFST)

The name of the queue to browse a message on (parameter identifier: MQCA_Q_NAME).

The maximum length of the string in MQ_Q_NAME_LENGTH (48).

9.4.2. Optional Parameters

Position (MQCFIN)

The numeric position of the message on the queue (parameter identifier: MQG_ATTR_POSITION).

MessageSelectorString (MQCFST)

An SQL92 expression to search for messages (parameter identifier: MQG_MSG_SELECTOR).

This string will be used in the MQOD SelectionString field. See IBM MQ documentation on this field to understand its contents.

MessageId (MQCFBS)

The message ID of a specific message on the queue (parameter identifier: MQG_ATTR_MSG_ID).

ConvertCCSID (MQCFIN)

The CodedCharSetId to convert messages into (parameter identifier: MQG_CONVERT_CCSID).

If not specified, the value MQCCSID_Q_MGR (0) will be used.

ConvertEncoding (MQCFIN)

The Encoding to convert messages into (parameter identifier: MQG_ATTR_MQMONA_CONVERT_ENCODING).

If not specified, the value MQENC_NATIVE for the queue manager platform will be used.

MessageToken (MQCFBS)

The message token of a specific message on the queue (parameter identifier: MQG_ATTR_MSG_TOKEN).

It is expected that this is taken from a previous response to an **InquireMessages** command.

MaxMsgSize (MQCFIN)

The maximum size of message data to return on the command (parameter identifier: MQG_MAX_MSG_LENGTH).

It is not possible to browse the full length of a 100MB length message using MQMONA. This is simply because in order to return a response message containing the message data, there are PCF headers wrapping the data and other fields that are being returned at the same time, and so it won't all fit into a single response message.

This parameter is not applicable to MQG_CMD_INQUIRE_MESSAGE or MQG_CMD_INQUIRE_MSG_NEXT_FAST.

MessageOptions (MQCFIN)

Options to control the way messages are browsed on the queue (parameter identifier: MQG_MSG_OPTIONS). Possible values are shown below.

MQG_MO_NO_CONVERT

Do not data convert the messages. The default behaviour, if this option is not used, is to use MQGMO_CONVERT to convert the messages into the CCSID provided in the **ConvertCCSID** parameter.

MQG_MO_NO_PROPERTIES

Do not force messages properties into an MQRFH2 on the front of the message. The default behaviour if this option is not used, is to use the MQGMO_PROPERTIES_FORCE_MQRFH2 option.

MQG_MO_CHECK_POSITION

For platforms where the queue manager does not support message tokens, this option can be used to ensure that the message position of the message matches that previously return (for example using **InquireMessages** to create a list), thus allowing you to detect that the contents of the queue have changed between the two operations.

This option is only applicable to MQG_CMD_INQUIRE_MSGDETAIL.

MQG_MO_CHECK_MSGID

For platforms where the queue manager does not support message tokens, this option can be used to ensure that the message ID of the message matches that previously return (for example using **InquireMessages** to create a list), thus allowing you to detect that the contents of the queue have changed between the two operations.

This option is only applicable to MQG_CMD_INQUIRE_MSGDETAIL.

MQG_MO_CHECK_BOTH

For platforms where the queue manager does not support message tokens, this option can be used to check both the position and the message ID as described in the preceding options.

This option is only applicable to MQG_CMD_INQUIRE_MSGDETAIL.

9.4.3. Response data

All the following fields are returned in the response to the command. The table also shows the MQMD field that this represents. Review the IBM MQ documentation on the field to understand the returned values.

Attribute	Description	MQMD Field
MQCA_Q_NAME	The name of the queue	N/A
MQG_ATTR_POSITION	Numeric position on the queue of the message	N/A
MQG_ATTR_MSG_LENGTH	The length of the message	N/A
MQG_ATTR_MSG_TOKEN	The message token representing the message	N/A
MQG_ATTR_MESSAGE	The message data	N/A
MQG_ATTR_REPORT	The report options set in the message	`Report`
MQG_ATTR_MSG_TYPE	The type of the message	`MsgType`
MQG_ATTR_EXPIRY	The remaining time to live of the message	`Expiry`
MQG_ATTR_FEEDBACK	The feedback code in the message	`Feedback`
MQG_ATTR_ENCODING	The encoding of the message	`Encoding`
MQG_ATTR_CODECHARSETID	The CCSID of the message	`CodedCharSetId`
MQG_ATTR_FORMAT	The message format	`Format`
MQG_ATTR_PRIORITY	The priority of the message	`Priority`
MQG_ATTR_PERSISTENCE	The persistence of the message	`Persistence`
MQG_ATTR_MSG_ID	The message ID of the message in hex	`MsgId`
MQG_ATTR_CHAR_MSG_ID	The message ID of the message as a string	`MsgId`
MQG_ATTR_CORREL_ID	The correlation ID of the message in hex	`CorrelId`
MQG_ATTR_CHAR_CORREL_ID	The correlation ID of the message as a string	`CorrelId`
MQG_ATTR_BACKOUT_COUNT	How often this message has been rolled back	`BackoutCount`
MQG_ATTR_REPLYTOQ	The reply-to-queue in the message	`ReplyToQ`
MQG_ATTR_REPLYTOQMGR	The reply-to-queue manager in the message	`ReplyToQMgr`
MQG_ATTR_MSG_USER_ID	The user id in the message	`UserIdentifier`
MQG_ATTR_ACCOUNTING_TOKEN	The accounting token in the message	`AccountingToken`
MQG_ATTR_APPL_IDENTITY_DATA	Application data relating to identity	`ApplIdentityData`
MQG_ATTR_PUT_APPL_TYPE	The type pf application that put the message	`PutApplType`
MQG_ATTR_PUT_APPL_NAME	The name of the application put the message	`PutApplName`
MQG_ATTR_PUT_DATE	The date on which the message was put	`PutDate`
MQG_ATTR_PUT_TIME	The time at which the message was put	`PutTime`
MQG_ATTR_APPL_ORIGIN_DATA	Application data relating to origin	`ApplOriginData`
MQG_ATTR_GROUP_ID	The Group ID of the message in hex	`GroupID`
MQG_ATTR_MSG_SEQ_NUMBER	Sequence number of logical message in group	`MsgSeqNumber`
MQG_ATTR_OFFSET	Offset of data in physical message from start of logical message	`Offset`
MQG_ATTR_MSG_FLAGS	Message flags	`MsgFlags`

9.4.4. Security Requirements

The user issuing the Inquire Individual Messages commands requires access to browse the queue.

- **On distributed this means**
  they require `+browse` on the `queue` being browsed.

- **On z/OS this means**
  they require READ access on the *`hlq`*`.`*`queuename`* `MQQUEUE` profile.

The user id under which MQMONA is running requires access to use alternate user ids.

- **On distributed this means**
  it requires `+altusr` on the `qmgr` object.

- **On z/OS this means**
  it requires UPDATE access on the *`hlq`*`.ALTERNATE.USER.`*`userid`* `MQADMIN` profile.

9.5. Move/Copy Message

Use the PCF command MQG_CMD_MOVE_MSGS or MQG_CMD_COPY_MSGS to move or copy one or many messages from a source queue to a target queue. The availability of these commands can be turned off using the messagecmds switch in the configuration file. See Chapter 18. Configuration File on page 55 for details.

9.5.1. Required Parameters

SourceQueueName (MQCFST)

The name of the queue to move or copy messages from (parameter identifier: MQCA_Q_NAME).

The maximum length of the string in MQ_Q_NAME_LENGTH (48).

TargetQueueName (MQCFST)

The name of the queue to move or copy messages to (parameter identifier: MQG_TARGET_Q).

Alternatively, if the MQG_MO_STRIP_HEADER option is specified, this can be provided as a single asterisk '*' to convey that the target queue name should be taken from the header.

The maximum length of the string in MQ_Q_NAME_LENGTH (48).

TargetQMgrName (MQCFST)

The name of the queue manager where the target queue resides (parameter identifier: MQG_TARGET_QM).

Alternatively, if the MQG_MO_STRIP_HEADER option is specified, this can be provided as a single asterisk '*' to convey that the target manager queue name should be taken from the header.

The maximum length of the string in MQ_Q_MGR_NAME_LENGTH (48).

9.5.2. Optional Parameters

LinkId (MQCFIN)

A unique ID to link together actions that might cause queues to be opened, so that they can all be closed together when work is completed (parameter identifier: MQG_ATTR_MQMONA_QUEUE_LINK_ID).

MessageSelectorString (MQCFST)

An SQL92 expression to search for messages (parameter identifier: MQG_MSG_SELECTOR).

This string will be used in the MQOD SelectionString field. See IBM MQ documentation on this field to understand its contents.

MessageId (MQCFBS)

One or many message IDs of specific messages on the queue that are to be copied or moved (parameter identifier: MQG_ATTR_MSG_ID).

MessageToken (MQCFBS)

One or many message tokens of specific message on the queue that are to be copied or moved (parameter identifier: MQG_ATTR_MSG_TOKEN).

Position (MQCFIL)

The numeric positions of the messages on the queue that are to be copied or moved (parameter identifier: MQG_ATTR_POSITION).

MessageOptions (MQCFIN)

Options to control the way messages are browsed on the queue (parameter identifier: MQG_MSG_OPTIONS). Possible values are shown below.

MQG_MO_CHECK_POSITION

For platforms where the queue manager does not support message tokens[3], this option can be used to ensure that the message position of the message matches that previously returned (for example using **InquireMessages** to create a list), thus allowing you to detect that the contents of the queue have changed between the two operations.

The **Position** parameter must also be provided if this option is used.

MQG_MO_CHECK_MSGID

For platforms where the queue manager does not support message tokens[3], this option can be used to ensure that the message ID of the message matches that previously return (for example using **InquireMessages** to create a list), thus allowing you to detect that the contents of the queue have changed between the two operations.

The **MessageId** parameter must also be provided if this option is used.

MQG_MO_CHECK_BOTH

For platforms where the queue manager does not support message tokens[3], this option can be used to check both the position and the message ID as described in the preceding options.

The **Position** and **MessageId** parameters must both be provided if this option is used.

MQG_MO_APPLY_TO_ALL

This move or copy command applies to all messages on the queue. Any **MessageID**, **Position** or **MsgToken** supplied will be ignored.

MQG_MO_STRIP_HEADER

Any transmission queue header (MQXQH) or dead-letter queue header (MQDLH) will be stripped off the message before putting it to the target queue.

MQG_MO_MULTI_TRANSACTION

The gets and puts will be done in multiple transactions, the size of which will be dictated by the queue managers' Maximum Uncommitted Messages attribute. If this option is not used, all the gets and puts will be attempted in a single transaction which may encounter MQRC_SYNCPOINT_LIMIT_REACHED (2024) if there are too many messages.

MQG_MO_CONTEXT_SETALL

The put of the message to the target queue will be done by setting all the context fields that were retrieved from the source queue.

MQG_MO_CONTEXT_PASS

The put of the message to the target queue will be done passing the context from the message retrieved from the source queue. This is only applicable for the MQG_CMD_MOVE_MSG command, since passing context cannot be done after a browse.

[3] All supported versions of queue managers support Message Token, so while this option is provided for backwards compatibility, it is expected that all new uses of this command will be able to use Message Token.

MQG_MO_CONTEXT_DEFAULT

The put of the message to the target queue will be done with default context, and so the context fields in the MQMD will lose their original values from the source queue and will have new values filled in representing MQMONA.

9.5.3. Progress Response Messages

If there are a lot of messages being moved or copied, this command could take some time, so progress responses messages will be sent back to update the requesting application on the progress of the command. This is controlled by the progdelay and progfreq attributes in the configuration file. See Chapter 18. Configuration File on page 55 for details.

For details of the contents of the response messages, see 9.9. Progress Message (Response) on page 40.

9.5.4. Security Requirements

The user issuing the Copy Messages (MQG_CMD_COPY_MSGS) command requires access to browse the source queue and put to the target queue.

- **On distributed this means**
  they require +browse on the source queue and +put on the target queue.
- **On z/OS this means**
  they require READ access on the *hlq.sourcequeuename* MQQUEUE profile and UPDATE access on the *hlq.targetqueuename* MQQUEUE profile

The user issuing the Move Messages (MQG_CMD_MOVE_MSGS) command requires access to get from the source queue and put to the target queue.

- **On distributed this means**
  they require +get on the source queue and +put on the target queue.
- **On z/OS this means**
  they require UPDATE access on the *hlq.sourcequeuename* MQQUEUE profile and UPDATE access on the *hlq.targetqueuename* MQQUEUE profile.

In addition, depending on which MQG_MO_CONTEXT options are used, additional authorities will be needed.

- **On distributed this means**
  - If MQG_MO_CONTEXT_PASS is used they require +passall on the qmgr object and on the target queue.
  - If MQG_MO_CONTEXT_SETALL is used they require +setall on the qmgr object and on the target queue.
- **On z/OS this means**
  - If MQG_MO_CONTEXT_PASS is used they require READ access on the *hlq*.CONTEXT.*queuename* MQADMIN profile.
  - If MQG_MO_CONTEXT_SETALL is used they require CONTROL access on the *hlq*.CONTEXT.*queuename* MQADMIN profile.

The user id under which MQMONA is running requires access to use alternate user ids.

- **On distributed this means**
  it requires +altusr on the qmgr object.
- **On z/OS this means**
  it requires UPDATE access on the *hlq*.ALTERNATE.USER.*userid* MQADMIN profile.

9.6. Delete Message

Use the PCF command MQG_CMD_DELETE_MSGS to delete one or many messages from a queue. The availability of this command can be turned off using the messagecmds switch in the configuration file. See Chapter 18. Configuration File on page 55 for details.

9.6.1. Required Parameters

QueueName (MQCFST)

The name of the queue to delete messages from (parameter identifier: MQCA_Q_NAME).

The maximum length of the string in MQ_Q_NAME_LENGTH (48).

9.6.2. Optional Parameters

LinkId (MQCFIN)

A unique ID to link together actions that might cause queues to be opened, so that they can all be closed together when work is completed (parameter identifier: MQG_ATTR_MQMONA_QUEUE_LINK_ID).

MessageToken (MQCFBS)

One or many message tokens of specific message on the queue that are to be deleted (parameter identifier: MQG_ATTR_MSG_TOKEN).

MessageOptions (MQCFIN)

Options to control the way messages are browsed on the queue (parameter identifier: MQG_MSG_OPTIONS). Possible values are shown below.

MQG_MO_CHECK_POSITION

For platforms where the queue manager does not support message tokens[4], this option can be used to ensure that the message position of the message matches that previously returned (for example using **InquireMessages** to create a list), thus allowing you to detect that the contents of the queue have changed between the two operations.

The **Position** parameter must also be provided if this option is used.

MQG_MO_CHECK_MSGID

For platforms where the queue manager does not support message tokens[4], this option can be used to ensure that the message ID of the message matches that previously return (for example using **InquireMessages** to create a list), thus allowing you to detect that the contents of the queue have changed between the two operations.

The **MessageId** parameter must also be provided if this option is used.

MQG_MO_CHECK_BOTH

For platforms where the queue manager does not support message tokens[4], this option can be used to check both the position and the message ID as described in the preceding options.

The **Position** and **MessageId** parameters must both be provided if this option is used.

[4] All supported versions of queue managers support Message Token, so while this option is provided for backwards compatibility, it is expected that all new uses of this command will be able to use Message Token.

MQG_MO_APPLY_TO_ALL

This delete command applies to all messages on the queue. Any **MessageID**, **Position** or **MsgToken** supplied will be ignored.

MessageSelectorString (MQCFST)

An SQL92 expression to search for messages (parameter identifier: `MQG_MSG_SELECTOR`).

This string will be used in the `MQOD SelectionString` field. See IBM MQ documentation on this field to understand its contents.

MessageId (MQCFBS)

One or many message IDs of specific messages on the queue that are to be deleted (parameter identifier: `MQG_ATTR_MSG_ID`).

Position (MQCFIL)

One or many numeric positions of the messages on the queue that are to be deleted (parameter identifier: `MQG_ATTR_POSITION`).

9.6.3. Progress Response Messages

If there are a lot of messages being deleted, this command could take some time, so progress responses messages will be sent back to update the requesting application on the progress of the command. This is controlled by the `progdelay` and `progfreq` attributes in the configuration file. See Chapter 18. Configuration File on page 55 for details.

For details of the contents of the response messages, see 9.9. Progress Message (Response) on page 40.

9.6.4. Security Requirements

The user issuing the Delete Messages (`MQG_CMD_DELETE_MSGS`) command requires access to get from the queue.

- **On distributed this means**
  they require `+get` on the `queue`.

- **On z/OS this means**
  they require UPDATE access on the *`hlq`*`.`*`queuename`* `MQQUEUE` profile.

The user id under which MQMONA is running requires access to use alternate user ids.

- **On distributed this means**
  it requires `+altusr` on the `qmgr` object.

- **On z/OS this means**
  it requires UPDATE access on the *`hlq`*`.ALTERNATE.USER.`*`userid`* `MQADMIN` profile.

9.7. Close Queue

Use the PCF command MQG_CMD_CLOSE_QUEUE to inform MQMONA that you are finished operating on a queue that you previously used in one of the following commands:

- MQG_CMD_INQUIRE_MESSAGES
- MQG_CMD_INQUIRE_MESSAGE
- MQG_CMD_INQUIRE_MSG_NEXT_FAST
- MQG_CMD_INQUIRE_MSGDETAIL
- MQG_CMD_INQUIRE_MSGDETAIL_NEXT_FAST
- MQG_CMD_MOVE_MSGS
- MQG_CMD_COPY_MSGS
- MQG_CMD_DELETE_MSGS

The availability of this command can be turned off using the messagecmds switch in the configuration file. If you don't use this command, the object handle that MQMONA holds open, will be closed using the same hobjttl configuration attribute used for other queues (reply queues mainly) that MQMONA opens. See Chapter 18. Configuration File on page 55 for details of both of these configuration attributes.

9.7.1. Required Parameters

QueueName (MQCFST)

The name of the queue to close (parameter identifier: MQCA_Q_NAME).

The maximum length of the string in MQ_Q_NAME_LENGTH (48).

9.7.2. Optional Parameters

LinkId (MQCFIN)

Close all the queues previously opened on behalf of commands providing this unique ID (parameter identifier: MQG_ATTR_MQMONA_QUEUE_LINK_ID).

9.8. Abort Command

Use the PCF command MQG_CMD_ABORT_COMMAND to inform MQMONA that you wish to interrupt one of the following commands you previously issued:

- MQG_CMD_INQUIRE_MESSAGES
- MQG_CMD_MOVE_MSGS
- MQG_CMD_COPY_MSGS
- MQG_CMD_DELETE_MSGS

The availability of this command can be turned off using the messagecmds switch in the configuration file. See Chapter 18. Configuration File on page 55 for details.

9.8.1. Required Parameters

MessageId (MQCFBS)

The message id of the previous command message that was put on the MQMONQ command queue that should be interrupted (parameter identifier: MQBACF_MSG_ID).

The maximum length of the string in MQ_MSG_ID_LENGTH (24).

9.9. Progress Message (Response)

In response to a possibly long-running action, periodic response messages will be returned to the requestor to detail the progress of the command. This message consists of the response header followed by the combination of attribute parameter structures. The frequency of these progress response messages can be controlled using the `progfreq` attribute in the configuration file. See Chapter 18. Configuration File on page 55 for details.

9.9.1. Always returned

MessagesIn, MessagesOut, BytesIn, BytesOut, ProgressPosition, ProgressMaximum

9.9.2. Response data

MessagesIn (MQCFIN)

Messages consumed from the queue (parameter identifier: `MQG_ATTR_MQMONA_PRG_MSGSIN`).

MessagesOut (MQCFIN)

Messages put to the queue (parameter identifier: `MQG_ATTR_MQMONA_PRG_MSGSOUT`).

BytesIn (MQCFIN)

Bytes consumed from the queue (parameter identifier: `MQG_ATTR_MQMONA_PRG_BYTESIN`).

BytesOut (MQCFIN)

Bytes put to the queue (parameter identifier: `MQG_ATTR_MQMONA_PRG_BYTESOUT`).

ProgressPosition (MQCFIN)

The current progress position (parameter identifier: `MQG_ATTR_MQMONA_PRG_POS`).

ProgressMaximum (MQCFIN)

The target progress position (parameter identifier: `MQG_ATTR_MQMONA_PRG_MAX`).

ProgressStatus (MQCFIN)

The status of the command (parameter identifier: `MQG_ATTR_MQMONA_PRG_STATUS`).

Possible values are:

MQG_PROGRESS_DONE

Progress on this command is complete. These are the final numbers.

MQG_PROGRESS_ABORTED

Progress on this command has been aborted.

9.10. Inquire Queue Manager (Response) – Annotation

When you issue the PCF command MQCMD_INQUIRE_Q_MGR, MQMONA annotates the response message with the following fields.

MQMONACmdLevel (MQCFIN)

The command level of MQMONA (parameter id: MQG_ATTR_MQMONA_CMDLEVEL).

MQMONAFeatures (MQCFIN)

The features currently enabled in MQMONA (parameter identifier: MQG_ATTR_MQMONA_FEATURES).

Zero or more flags will be set. Possible value flags are:

MQG_FEATURES_UNLICENSED

MQMONA was not able to find a valid licence file and so all features are turned off, and MQMONA passes all commands through to IBM MQ.

MQG_FEATURES_COALESCING

Response messages from IBM MQ are gathered together into fewer, larger response messages. See Chapter 4. Consolidation on page 7 for more information.

MQG_FEATURES_PCF_BLANKS_REMOVAL

PCF response messages from IBM MQ have their spurious blanks removed, with corresponding lengths updated accordingly. See Chapter 5. Compression on page 9 for more information.

MQG_DISPLAY_CERT

MQMONA will accept DISPLAY CERT commands. See 9.1. DISPLAY CERT on page 19 for more information.

MQG_FEATURES_CERT_EXPIRY_EVENTS

MQMONA will create certificate expiry events. See Chapter 7. Certificate Events on page 13 for more information.

MQG_FEATURES_MESSAGES

Message manipulation commands are available. See Chapter 6. Message Manipulation on page 10 for more information.

MQMONABuildDate (MQCFST)

The build date of MQMONA (parameter id: MQG_ATTR_MQMONA_BUILD).

The date is always a string literal of 11 characters in the form "Mmm dd yyyy".

Chapter 10. Event Reference

The MQMONA Agent can generate event messages if it can find an MQEV licence in the `mqgem.lic` file you direct it to. This chapter describes the contents of those event messages.

10.1. Channel SSL Certificate Nearing Expiry

This event message can either be generated on request by issuing a `DISPLAY CERT` command with a non-zero value in the `EVTTHRES` parameter, or by configuring the MQMONA Agent to regularly check for expiring certificates in the background.

Event name:	Channel SSL Certificate Nearing Expiry
Reason code in MQCFH:	11000: MQGRC_CERTIFICATE_NEARING_EXPIRY
Event description:	This is issued when a certificate is detected to have less than the requested number of days remaining before it expires.
Event type:	SSL
Platforms:	All
Event queue:	SYSTEM.ADMIN.CHANNEL.EVENT

10.1.1. Event data

QmgrName

Description:	Name of the queue manager generating the event
Identifier:	MQCA_Q_MGR_NAME
Data type:	MQCFST
Maximum length:	MQ_Q_MGR_NAME_LENGTH (48)
Returned:	Always

CertificateLabel

Description:	Label of the certificate that is nearing expiry
Identifier:	MQCA_CERT_LABEL
Data type:	MQCFST
Maximum length:	MQ_CERT_LABEL_LENGTH (64)
Returned:	Always

CertificateUser

Description:	The owning user ID of the certificate that is nearing expiry
Identifier:	MQG_ATTR_MQMONA_CERT_USER
Data type:	MQCFST
Maximum length:	8
Returned:	Only on a z/OS queue manager

SSLKeyRepository

Description:	The Key Repository where there certificate is stored. This can be the full path and name of a KDB file, a PKCS#12 file, or on z/OS a keyring.
Identifier:	MQCA_SSL_KEY_REPOSITORY
Data type:	MQCFST
Maximum length:	MQ_SSL_KEY_REPOSITORY_LENGTH (256)
Returned:	Always

KeyRingUser

Description:	The owning user ID of the keyring.
Identifier:	MQG_ATTR_MQMONA_KEYRING_USER
Data type:	MQCFST
Maximum length:	8
Returned:	Only on a z/OS queue manager

SSLPeerName

Description:	The Subject's Distinguished Name in the certificate that is nearing expiry
Identifier:	MQCACH_SSL_PEER_NAME
Data type:	MQCFST
Maximum length:	MQ_DISTINGUISHED_NAME_LENGTH (1024)
Returned:	Always

CertificateSerialNumber

Description:	The Serial Number of the certificate that is nearing expiry
Identifier:	MQG_ATTR_MQMONA_CERT_SERIAL
Data type:	MQCFST
Maximum length:	MQG_CERT_SERIAL_LENGTH (64)
Returned:	Always

SSLCertIssuerName

Description:	The Issuer's Distinguished Name in the certificate that is nearing expiry
Identifier:	MQCACH_SSL_CERT_ISSUER_NAME
Data type:	MQCFST
Maximum length:	MQ_DISTINGUISHED_NAME_LENGTH (1024)
Returned:	Always

CertificateIssueTime

Description:	An integer representation[5] of the date and time the certificate was issued.
Identifier:	MQG_ATTR_MQMONA_CERT_ISSUE_TIME
Data type:	MQCFIN64
Returned:	Always

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

CertificateExpiryTime

Description:	An integer representation[5] of the date and time the certificate was issued.
Identifier:	MQG_ATTR_MQMONA_CERT_EXPIRY_TIME
Data type:	MQCFIN64
Returned:	Always

LifespanThreshold

Description:	The lifespan threshold in effect when the event was generated.
Identifier:	MQG_ATTR_MQMONA_LIFESPAN_THRESHOLD
Data type:	MQCFIN
Returned:	Always

Lifespan

Description:	The lifespan of the certificate, in seconds. If the number is negative, this means the certificate has already expired.
Identifier:	MQG_ATTR_MQMONA_CERT_LIFESPAN
Data type:	MQCFIN64
Returned:	Always

KeySize

Description:	The key size of the certificate.
Identifier:	MQG_ATTR_MQMONA_CERT_KEYSIZE
Data type:	MQCFIN
Returned:	Only on non-z/OS queue managers.

DefaultCertificate

Description:	Whether the certificate is marked as the default in the key repository.
Identifier:	MQG_ATTR_MQMONA_CERT_DEFAULT
Data type:	MQCFIN
Values:	**MQG_DEFAULT_CERT_YES** This certificate is the default. **MQG_DEFAULT_CERT_NO** This certificate is not the default.
Returned:	Always

CertificateUsage

Description:	What the certificate is used for.
Identifier:	MQG_ATTR_MQMONA_CERT_USAGE
Data type:	MQCFIN
Values:	**MQG_CERT_USAGE_PERSONAL** This certificate is a personal certificate. **MQG_CERT_USAGE_CACERT** This certificate is a CA certificate. **MQG_CERT_USAGE_OTHER** This certificate is neither a personal certificate nor a CA certificate.
Returned:	Always

TrustedCertificate

Description:	Whether the certificate is marked as trusted in the key repository.
Identifier:	MQG_ATTR_MQMONA_CERT_TRUSTED
Data type:	MQCFIN
Values:	**MQG_TRUSTED_CERT_YES** This certificate is trusted. **MQG_TRUSTED_CERT_NO** This certificate is not trusted.
Returned:	Always

QueueManagerCertificate

Description:	Whether the certificate is the queue manager's certificate.
Identifier:	MQG_ATTR_MQMONA_CERT_QMGR
Data type:	MQCFIN
Values:	**MQG_QMGR_CERT_YES** This certificate is the queue manager's certificate. **MQG_QMGR_CERT_NO** This certificate is not the queue manager's certificate.
Returned:	Always

Chapter 11. Running MQMONA as a SERVICE

You can setup MQMONA to run as a service by defining an IBM MQ SERVICE object so that it is always started up when your queue manager is started, and is ready and waiting to process your command messages.

Here is an example command showing how to configure this.

```
DEFINE SERVICE(MQGEM.MQMONA) SERVTYPE(SERVER) CONTROL(QMGR) +
       STARTCMD('c:\MQGem\mqmona.exe') +
       STARTARG('-m +QMNAME+ -f C:\MQGem\MQMONA.CFG -k') +
       STOPCMD('c:\MQGem\mqmona.exe')  STOPARG('-m +QMNAME+ -c STOP') +
       DESCR('MQGem Command Server Agent')
```

This service object can now be started and stopped using the IBM MQ commands:-

```
START SERVICE(MQGEM.MQMONA)
```

and

```
STOP SERVICE(MQGEM.MQMONA)
```

Additionally, you can check that it is running using the following command:-

```
DISPLAY SVSTATUS(MQGEM.MQMONA)
```

Chapter 12. Triggering

The MQMONA Agent can be trigger started. This may be useful instead of running it as a SERVICE which was described in the previous chapter. This is useful when you only need MQMONA to run when commands arrive. If you are using MQMONA to generate event messages in the background, you are advised to run it as a long-running program, such as as a SERVICE.

This chapter shows a full working example of triggering MQMONA. You may already have a trigger monitor in place, and so can skip some of the steps that follow.

12.1. Set up a trigger monitor

Define an initiation queue, and a service object to automatically run your trigger monitor, then start it. The DEFINE SERVICE command example below is for Windows, the bin directory is in a slightly different place on Unix, please adjust the STARTCMD and STOPCMD values accordingly (remove the '64').

```
DEFINE QLOCAL(MQGEM.INITQ) DESCR('Initiation Queue')
```

```
DEFINE SERVICE(MQGEM.TRIGGER.MONITOR) SERVTYPE(SERVER) CONTROL(QMGR) +
       DESCR('Trigger Monitor Service') +
       STARTCMD('+MQ_INSTALL_PATH+bin64/runmqtrm') +
       STARTARG('-m +QMNAME+ -q MQGEM.INITQ') +
       STOPCMD('+MQ_INSTALL_PATH+bin64/amqsstop') +
       STOPARG('-m +QMNAME+ -p +MQ_SERVER_PID+') +
       STDOUT('+MQ_Q_MGR_DATA_PATH+errors/MQGemTrigMonOut.txt') +
       STDERR('+MQ_Q_MGR_DATA_PATH+errors/MQGemTrigMonErr.txt')
```

```
START SERVICE(MQGEM.TRIGGER.MONITOR)
```

12.2. Describe MQMONA, the triggered application

It is assumed at this point, that you have already successfully run MQMONA manually from a command line and that the queue MQMONA requires to run has already been defined as described in Chapter 3. Getting Started on page 4. In the example commands below, adjust the path of the executable and configuration file for your system.

Here's an example DEFINE PROCESS command for Windows.

```
DEFINE PROCESS(MQGEM.MQMONA.PROC) DESCR('MQGem Command Server Agent') +
       APPLICID('START "" /B c:\MQGem\mqmona.exe') +
       USERDATA('-f c:\MQGem\MQMONA.CFG -k')
```

Here's an example DEFINE PROCESS command for Linux.

```
DEFINE PROCESS(MQGEM.MQMONA.PROC) DESCR('MQGem Command Server Agent') +
       APPLICID('/u/mqgemusr/bin/mqmona') ENVDATA('&') +
       USERDATA('-f /u/mqgemusr/MQMONA.CFG -k')
```

And here are the changes to make to your MQMONA command queue to set it up for triggering.

```
ALTER QLOCAL(MQGEM.MQMONA.COMMAND.QUEUE) INITQ(MQGEM.INITQ) +
      PROCESS(MQGEM.MQMON.PROC) TRIGGER TRIGTYPE(FIRST)
```

Now that you have the trigger configuration in place, you can easily test that the triggering is correctly set up by sending a command message to MQMONA. If you get your responses back, then you know it worked.

When MQMONA is trigger started, it will report this in its error log with message **MQGA003I**.

Triggering is useful to combine with the configuration file keyword `discint`, whose default value is 60 minutes. This means the MQMONA program will gracefully end after this time if there is no work for it to do.

Chapter 13. Security

There are two aspects of security to consider here; the requirements on the administrative applications making use of the MQMONA Command Server agent, and that of the MQMONA program itself.

13.1. MQ Administrator using MQMONA

If an MQ Administrator is issuing commands via MQMONA, say from the MO71 GUI, then there are some additional authorities their user ID will require over and above what is normally needed to use MQ from MO71.

The generic profile below covers the `MQGEM.MQMONA.COMMAND.QUEUE`, and the responses sent by the Queue Manager command server to the MQMONA reply queue which by default will be a dynamic queue beginning with `MQGEM.MQMONA.*`

If you change the `TempQPfx` configuration keyword (see Chapter 18. Configuration File on page 55), then a profile covering your choice of name will be needed. This is because the commands sent to the Queue Manager's command server retain your user id so that commands are issued by your user id. This naturally also results in the responses put by the Queue Manager's command server being done under your user id (to the MQMONA reply queue).

13.1.1. Example security command for Distributed Platforms

Suppose you have a user, in the group MO71ADM, who wishes to use MO71 through the MQMONA program. Below are listed the minimum additional authorities the user requires in order to successfully use MQMONA.

```
SET AUTHREC PROFILE(MQGEM.MQMONA.**)              GROUP('MO71ADM') AUTHADD(PUT)
```

13.1.2. Example security command for z/OS

Suppose you have a user, in the group MO71ADM, who wishes to use MO71 through the MQMONA program. Below are listed the minimum additional authorities the user requires in order to successfully use MQMONA. It assumes 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.MQGEM.MQMONA.**              CLASS(MQQUEUE) ID(MO71ADM) ACCESS(UPDATE)
```

13.2. MQMONA Program

If you run the MQMONA program as a SERVICE, or by trigger starting it, as described in the previous two chapters, then, on the distributed platforms, the program will be running as a privileged user id. This means specific authorities will not need to be granted. However, if you choose to run it under a non-privileged user id, or you're running it on z/OS, then this section shows what the program requires.

Notes:

- Alternate User authority is required to ensure any message manipulation commands, (browse, copy, move, delete) are done under the authority of the originating user id and not the MQMONA user id. In addition, responses for all commands are put to the originating users reply queue using its authority.
- Pass all context authority is required on the MQ command queue to ensure the commands sent on to the Queue Manager Command Server retain the user id of the originating user so that the command is issued under their authority.

- Pass all context authority is required on the user reply queues to pass the context information in the replies from the Queue Manager Command Server back to the user. Examples are shown below for default MO71 and MQSCX reply queue names. If you use different reply queue names, then you will need to adjust your authorities accordingly.

Other features of MQMONA will have additional security requirements. For example, producing event messages requires access to put to the event queue. Details of those security requirements are noted in the specific chapters.

13.2.1. Example security commands for Distributed Platforms

Suppose you have a user ID, in the group MQMONA, which will run the MQMONA program. Below are listed the minimum authorities the user requires in order for MQMONA to operate correctly.

```
SET AUTHREC OBJTYPE(QMGR)                          GROUP('MQMONA') AUTHADD(CONNECT,INQ,ALTUSR)
SET AUTHREC PROFILE(MQGEM.MQMONA.**)                 GROUP('MQMONA') AUTHADD(GET,PUT)
SET AUTHREC PROFILE(SYSTEM.ADMIN.COMMAND.QUEUE) GROUP('MQMONA') AUTHADD(PUT,PASSALL)
SET AUTHREC PROFILE(SYSTEM.DEFAULT.MODEL.QUEUE) GROUP('MQMONA') AUTHADD(GET,DSP)
SET AUTHREC PROFILE(MQMON.**)                        GROUP('MQMONA') AUTHADD(PASSALL)
SET AUTHREC PROFILE(MQSCX.**)                        GROUP('MQMONA') AUTHADD(PASSALL)
```

13.2.2. Example security commands for z/OS

Suppose you have a user ID, in the group MQMONA, which will run the MQMONA program. Below are listed the minimum authorities the user requires in order for MQMONA to operate correctly. It assumes 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(MQMONA) ACCESS(READ)
PERMIT qmgr.ALTERNATE.USER.*               CLASS(MQADMIN)  ID(MQMONA) ACCESS(UPDATE)
PERMIT qmgr.MQGEM.MQMONA.**                CLASS(MQQUEUE)  ID(MQMONA) ACCESS(UPDATE)
PERMIT qmgr.SYSTEM.COMMAND.INPUT           CLASS(MQQUEUE)  ID(MQMONA) ACCESS(UPDATE)
PERMIT qmgr.SYSTEM.DEFAULT.MODEL.QUEUE CLASS(MQQUEUE)  ID(MQMONA) ACCESS(UPDATE)
PERMIT qmgr.CONTEXT.MQMON.**               CLASS(MQADMIN)  ID(MQMONA) ACCESS(READ)
PERMIT qmgr.CONTEXT.MQSCX.**               CLASS(MQADMIN)  ID(MQMONA) ACCESS(READ)
```

Chapter 14. Timing out commands

There are error conditions when the IBM MQ Command Server will never reply, often when the original command message has been put to the Dead-letter Queue.

In this situation, MQMONA will wait for the response to arrive for a configurable period of time, which is by default 10 seconds. You can adjust this interval using the `maxage` keyword in the configuration file.

If this time period elapses for a command without any response from the IBM MQ Command Server, MQMONA will take the following actions.

- It will send a response message with error message **MQGA501E** to the original application to tell it that there was no response from the IBM MQ Command Server.
- It will write message **MQGA303E** to the error log to record the message ID of the command that has now been discarded.

Chapter 15. Status Reporting

MQMONA will periodically report some status about how much work it has done, and what it has achieved for you. This chapter details what those status values are.

These status reports will be written to the error log file, by default, every 60 minutes.

You can change this interval using the `reportint` keyword in the configuration file.

Additionally, they can be requested at any time by sending a simple command, REPORT, to MQMONA. See Chapter 17. Simple Commands on page 54 for details.

Status	Meaning
Commands Received	The number of commands (including MQSC, PCF and simple string commands) received on the MQMONA input queue
Command Bytes Received	The command messages received totalled this many bytes
Replies Received	The number of reply messages received from the IBM MQ Command Server
Reply Bytes Received	The reply messages received totalled this many bytes
Responses Sent	The number of responses sent to original application reply queues. This number reflects the messages after they have been consolidated.
Response Bytes Sent	The response messages sent totalled this many bytes. This number reflects the messages after they have been compressed.
Compression Achieved	A comparison between Response Bytes Sent and Reply Bytes Received to show how much compression has been achieved.
Failed Puts	The number of puts which were not successfully.
Put Successfully Retried	The number of puts, that initially failed, but after some number of retries were eventually successful.
Worker Threads	The high water mark of worker threads.

Chapter 16. Parameters

There are a number of parameters that can be passed to MQMONA to control the main behaviour you need. Additionally there is a configuration file containing optional attributes for more detailed control of various behaviours. This is described in the next chapter.

16.1. Parameters Flags

Parameters are passed to the program as flags on the command line. The following parameters are available:

Flag	Meaning
-c	<Simple Command> This parameter can be used to put a very simple string command onto the MQMONA input queue. See Chapter 17. Simple Commands on page 54 for the list of valid commands. The program puts the string command onto the queue and then immediately ends.
-f	<configuration file name> This parameter provides the name of the configuration file. If not specified, MQMONA will look for a file called MQMONA.CFG in the same directory as the program (Windows and Linux) or in the current directory (AIX and z/OS UNIX). For more details about this configuration file, see Chapter 18. Configuration File on page 55.
-k	Run as a SERVICE
-m	<Queue Manager name> Specifies the name of the Queue Manager to connect to if not the default.
-q	<input queue name> Specifies the input queue to read from. By default this will use a queue called MQGEM.MQMONA.COMMAND.QUEUE.
-v	Verbose Level Controls which informational or error messages are written to the log. Review Chapter 22. Messages on page 65 to see which messages are written at which verbose levels. Value can be Quiet, Low, Medium, or High. Only the first character need be provided.

Chapter 17. Simple Commands

You can send some very simple string commands to MQMONA.

You can do this in a number of ways.

- Put a simple string format message to the MQMONA input queue, which is, by default `MQGEM.MQMONA.COMMAND.QUEUE`, using
  - A simple MQ sample program such as `amqsput`, or the MO71 Put Message dialog
  - the MQMONA program itself with the -c flag
- On z/OS, you can also use the MVS `MODIFY` command, putting the command in as the `APPL=` string. e.g.

```
/MODIFY MQG1MONA,APPL=REPORT
```

- On z/OS, you can also use the MVS `STOP` command, e.g.

```
/STOP MQG1MONA
```

Valid commands are:-

- **STOP**
  MQMONA will end when it receives this command
- **REPORT**
  MQMONA will write out a status report when it receives this command. See Chapter 15. Status Reporting on page 52.

Chapter 18. Configuration File

In addition to the main parameters shown in the previous chapter, the configuration file can be used to supply additional optional configuration of how the program operates.

18.1. Example Configuration File

To demonstrate the format of the MQMON configuration file, here is an example one.

```
#################################################################
# MQMONA configuration file
#################################################################
ReportInt = 90                      # 1 hour 30 minutes
ReplyQInq = no                      # This one is a yes|no switch
HobjTTL   = 60
LogPath   = "/home/gemuser/+QMNAME+"
Tempqpfx  = "MQGEM.MQMONA.GEMUSER.*"
mqgeml    = "/home/gemuser"
DiscInt   = 5
```

18.2. Configuration File keywords

The configuration file is free-format and keywords are case insensitive. Keywords are specified in the file with an equals sign and then the value. Surround string values with double quotes as demonstrated in the example above.

Keyword	Meaning
certDisplay	This `yes|no` switch can be used to turn off the availability of the `DISPLAY CERT` command. Read more about this command in 9.1.DISPLAY CERT on page 19.
certEvent	This three value switch can be used to configure whether certificate expiry events are generated. Possible values are `yes|asqmgr|no`. The `asqmgr` setting looks to the queue manager `SSLEV` event switch to determine whether these events are generated. The default value is `asqmgr`. Read more about the certificate expiry events in 10.1.Channel SSL Certificate Nearing Expiry on page 42.
certFreq	The time period between checks for expiring certificates, The value is in hours with a default of 24 (Daily).
certThresh	A certificate which has a remaining lifespan less than this value will have an event written about it. The value is in days with a default of 30.
Coalesce	This `yes|no` switch can be used to turn off consolidation which is on by default. Read more about consolidation in Chapter 4. Consolidation on page 7.

CommandQ	The IBM MQ Command Server queue name. By default MQMONA will use `SYSTEM.ADMIN.COMMAND.QUEUE` unless the queue manager is discovered to be a z/OS platform queue manager in which case it will use `SYSTEM.COMMAND.INPUT` If you need a different queue than the above, you can use this keyword. If you do need to use this, please get in touch to let us know so that we can improve MQMONA to recognise other command queue names.
Compress	This `yes|no` switch can be used to turn off compression which is on by default. Read more about compression in Chapter 5. Compression on page 9.
DiscInt	If MQMONA has found nothing to do for this time period then it will end and await being trigger started again should any more command messages arrive. The value is in minutes with a default of 60 (Hourly). This keyword only applies if MQMONA was triggered started. It is ignored otherwise.
HobjTTL	The time period that an object handle will remain in MQMONA's cache without having been used. After this time the handle will be closed. The value is in seconds with a default of 300.
LogFailFatal	This `yes|no` switch controls whether MQMONA should continue running if it has a failure when working with its own error log files. By default it will stop running.
LogPath	The location where the rotating error log files will be written. If not specified, MQMONA will use the same directory as the program (Windows and Linux) or in the current directory (AIX and z/OS UNIX). A special value `+QMNAME+` can be used as a substitution for the queue manager name, allowing the ability to use the same configuration file for multiple queue managers. An example of this is shown in the example configuration file above. This keyword is not used if MQMONA is running in batch or as a started task on z/OS as error messages are written to the JESMSGLG instead.
MaxAge	The maximum time period that MQMONA will hold onto a request waiting for an answer from the IBM MQ Command Server. The value is in seconds with a default of 10. Once this time period is exceeded and no response has been received, MQMONA will throw away the request, writing message **MQGA303E** to the log.
MaxBrwsMsgs	The maximum number of messages that can be browsed in a single command. MQMONA supports browsing up to 30,000 messages at once. However, the default is 10,000 to avoid unnecessary delays when browsing deep queues. Essentially this value controls how wide the message display range is. Even with a small maximum browse range, and therefore a small display range it is still possible to browse deep queues. For example, to see a few messages near the 100,000th message one could specify a range of 99995,100005 on the **Inquire Messages** command.
MaxLogSize	The maximum size that an error log file can grow to before MQMONA will start writing to a new error log file. The value is in bytes with a default of 1048576 (1MB).

MaxMsgSize	The maximum size of response message that MQMONA will create when consolidating the replies from the IBM MQ Command Server. The value is in bytes with a default of 1048576 (1MB). Read more about consolidation in Chapter 4. Consolidation on page 7.
MaxWrkThrd	The maximum number of worker threads that will be allowed to run in MQMONA. Each worker thread will have its own connection to the queue manager. For more information about what these worker threads do, please see section Chapter 19. Worker Threads on page 60. The default value is 10.
messagecmds	This `yes|no` switch can be used to turn off the use of the various message manipulation commands which are available by default. Read more about these commands in Chapter 6. Message Manipulation on page 10.
mqerlCCSID	By default the code-page of the IBM MQ Error Log files are assumed to be in the same code-page as the Queue Manager. However, this parameter allows you to be more specific. Ensure that any clients issuing the the `MQCMD_INQUIRE_ERROR_LOG` command are capable of converting from the given code-page to their native value. Please see Chapter 8. Error Log Command on page 16.
mqerlChunk	To implement the `MQCMD_INQUIRE_ERROR_LOG` command MQMONA must read the error log files. Rather than read the entire error log (remember that error logs can get pretty big) MQMONA reads them in chunks. This parameter allows you to fine tune how large these chunks are. The value is in bytes with a default of 5242880 (5 MB). Please see Chapter 8. Error Log Command on page 16.
mqerlCmd	This `yes|no` switch controls whether MQMONA will support the `MQCMD_INQUIRE_ERROR_LOG` command. The default is `no`. Please see Chapter 8. Error Log Command on page 16.
mqerlPath	MQMONA will attempt to determine the location of the error logs automatically using a variety of methods depending on the platform. However, by setting this parameter you can be explicit about where the path is. Please see Chapter 8. Error Log Command on page 16.
mqerlKeepAge	MQMONA will perform housekeeping and discard any loaded messages and chunks after a certain period of time. This value controls the age of messages which are ripe for deletion. In other words you can prevent 'recent' messages from being discarded and you get to decide what is considered recent. The value is in seconds with a default of 7200 (2 hours). Please see Chapter 8. Error Log Command on page 16.
mqerlTTL	MQMONA will not perform housekeeping on any file chunks that have been referred to recently. In other words if a chunk was referred to to satisfy an error log command then it is exempt from housekeeping for a period of time. The value is in seconds with a default of 1800 (30 minutes) Please see Chapter 8. Error Log Command on page 16.

mqerlRespLimit	Error logs can get pretty huge so you might wish to limit how much of the error log can be queried in a single command. The value is in megabytes with a default of 100 which would allow the query of three full error logs using the default error log size which is 32MB. Please see Chapter 8. Error Log Command on page 16.
mqerlRespMax	When MQMONA responds with the contents of the error log it will split the answer into multiple responses. This parameter controls the maximum size that each response will be. The value is in bytes with a default of 1048576 (1 MB). Please see Chapter 8. Error Log Command on page 16.
mqgeml	The directory where the MQGem Software licence file is located. This will over-ride the default use of the location where the executable is stored. There is also the option to set the **MQGEML** environment variable for the same purpose. Supply the directory path where the licence file can be found without the name (the name will be assumed to be ***mqgem.lic***) – see example above. This attribute does not apply on z/OS where the expectation is that the DD card **MQGEML** references the location of the licence file.
NumLogFiles	The number of rotating error log files MQMONA will write to. The default value if this is not specified is 3.
progDelay	The interval before MQMONA sends the first of the progress responses to the caller. After which the frequency is handled by the `progfreq` attribute. If `progfreq` is zero, this attribute will have no effect. A value of zero means a progress response is sent immediately, which can be useful if MO71 is configured to show progress dialog boxes immediately. The value is in seconds and the default value is whatever `progfreq` is set to.
progFreq	The interval that MQMONA will send progress responses to the caller when processing browse, move, copy and delete message commands. If the command completes within this interval, no progress response will be sent, only the response to the command itself. A value of zero turns off progress responses entirely. The value is in seconds with a default value of 3.
ReplyQ	The reply queue to use. This can be a local queue, or a model queue. If it is a model queue, MQMONA will create a dynamic queue with the prefix `MQGEM.MQMONA.*` To use a different prefix, please use `tempqpfx` described below. By default MQMONA will use `SYSTEM.DEFAULT.MODEL.QUEUE` for this queue.
ReplyQInq	This `yes|no` switch can be used to disallow MQMONA from inquiring the ReplyToQ in a bid to discover the Maximum Message Length.
ReportInt	The interval that status is reported to the log. You can also force status to be reported with a simple string command. See the previous chapter for more details. The value is in minutes with a default of 60 (Hourly).
RetryCount	The number of times to re-attempt a failed MQPUT of a message before giving up. The default is 10 times.
RetryInt	The time period between re-attempted MQPUTs of failed messages. The value is in seconds with a default of 1.

TempQPfx	The prefix used when MQMONA creates a dynamic reply queue from the model queue name provided in the `replyq` keyword. If not specified, MQMONA will use a prefix of `MQGEM.MQMONA.*`

18.3. Configuration inline in JCL on z/OS

You may find it useful to keep your configuration file inline in your started task or batch job JCL. Here's an example of that.

```
//MQG1MONA  JOB 456,GEMUSER,MSGCLASS=H
//*****************************************************************
//* Run MQMONA
//*****************************************************************
//AGENT    EXEC PGM=MQMONA,
//         PARM=('-m MQG1 -f DD:CFG')
//STEPLIB  DD   DSN=GEMUSER.USER.LOAD,DISP=SHR
//         DD   DSN=IBM.MQ.SCSQAUTH,DISP=SHR
//         DD   DSN=IBM.MQ.SCSQANLE,DISP=SHR
//MQGEML   DD   DSN=GEMUSER.MQGEM.LIC,DISP=SHR
//CFG      DD   *
################################################################
# MQMONA configuration file
################################################################
reportint = 90
tempqpfx = "MQGEM.MQMONA.GEMUSER.*"
certevent = no
/*
//SYSPRINT DD   SYSOUT=*
//SYSOUT   DD   SYSOUT=*
//
```

Chapter 19. Worker Threads

MQMONA currently has three different classifications of commands.

- Commands to go to the IBM MQ Command Server, such as **`DISPLAY QUEUE(*)`**. These commands are handled by the main consumer as they are promptly passed onto the IBM MQ Queue Manager's command server's input queue. Worker threads are not involved in this classification of commands.
- The **`DISPLAY CERT`** command. This command is handled directly by MQMONA, and because there is the possibility that running an O/S command might take some time, it is handed off to a worker thread to process, to ensure that the main consumer remains responsive.
- Message Manipulation commands, such as browse, copy, move and delete messages. Depending on how many messages are on the queue being manipulated, there could be a great many MQGET and MQPUT calls being made. These commands are also handed off to a worker thread to process, to ensure that the main consumer remains responsive. In addition, any commands in this set that come from the same requester, are processed by the same worker thread for the lifetime of the object handle. This ensures that these commands are serialized for any one user and that several different worker threads don't all need to open the same queue for the same user.

MQMONA will self-tune the number of worker threads based on the number of concurrent input requests. If three different users all request to browse a queue at the exact same time, then there will likely be three worker threads running to process these requests. If these three different users come in at different times from each other, then it is likely that a single worker thread will process all of the requests.

Each worker thread will make its own connection to the queue manager, in order to access queues to browse, get and put messages. If you want limit the number of connections that the MQMONA program, as a whole, makes to IBM MQ, you can limit the number of worker threads using the `maxwrkthrd` keyword in the configuration file. See Chapter 18. Configuration File on page 55 for more details.

If you restrict it too much, and users are trying to do things concurrently, then your users may notice delays. You will also see error message **MQGA234E** if all threads were busy when a request came in and no more threads were allowed.

The regular status reports written by MQMONA include the high-water mark number of threads since MQMONA started, and the number in the most recent interval. This will help you decide whether the number of connections is acceptable. See Chapter 15. Status Reporting on page 52 for details.

Chapter 20. Performance Comparisons

Clearly the benefit you receive from using MQMONA will depend hugely on the latency and speed of your network. We deliberately tested this on poorly performing network connections, hopefully much worse connections than you have to deal with.

20.1. IBM MQ Administrative Commands

The benefits you can see with your MQ administrative commands will depend on your network, and also whether you are requesting large numbers of definitions, or large numbers of string attributes in your definitions.

20.1.1. Environment description

In this example we have a particularly high latency connection to a z/OS queue manager on the other side of the planet (New Zealand to United Kingdom) and low bandwidth.

We measured the network latency reported by TCP/IP ping which averages around 315 ms. FTP moving large files gave a throughput of around 56 Kbytes/sec.

20.1.2. Test results

The tests below are all retrieving a list of 1000 alias queue definitions with their Inhibit Put and Inhibit Get attributes using MO71 over a non-TLS client connection. When connecting to a z/OS queue manager using MO71, the command type in use is MQSC.

#	Test Description	# Responses	Time Taken
1	MQ Client Read-ahead is disabled	1000	5m 48s
2	MQ Client Read-ahead is enabled	1000	19s
3	MQMONA is used with maxmsgsize = 100,000	2	9s
4	MQMONA is used with maxmsgsize = 900,000	1	6s

These tests also illustrate the benefits that read-ahead can bring you when using a network with high latency. Hopefully queues such as the `SYSTEM.DEFAULT.MODEL.QUEUE` which are often used to create dynamic reply queues for administrative tools already have `DEFREAD(ON)` set, so you will not experience the numbers in test #1!

MQ Client Read-ahead can only go so far though, and that's where MQMONA can offer additional help. Rather than speeding up how fast an MQ Client can consume a 1000 small messages, MQMONA changes the pattern so that the MQ Client is consuming a very small number of much larger messages. The comparison between test #2 (which is where we hope everyone is at) and test #4 is a three times improvement.

20.2. Message Manipulation

The benefits you can see when manipulating messages will depend on your network, and also the number and size of messages you are manipulating (moving, copying or deleting).

20.2.1. Environment description

In this example we have another high latency connection to a z/OS queue manager somewhere around the globe (New Zealand to Dallas) and low bandwidth.

We measured the network latency reported by TCP/IP ping which averages around 220 ms. FTP moving large files gave a throughput of around 120 Kbytes/sec.

20.2.2. Test results

The tests below are copying 1000 persistent messages from one queue to another. That is a browse followed by a put, so the read ahead setting is applicable. The tests use increasing message sizes to show different benefits. The three different configurations of client in the test are as follows:-

- **A) Client Native:** A client program, using queues configured without Read Ahead or Async Put.
- **B) Client with RA/AP:** A client program, using queues configured to use both Read Ahead and Async Put, but using the default size for the read-ahead buffer which is 100KB.
- **C) Client via MQMONA:** A client program, going via MQMONA.

#	Test Description	Time Taken		
		A) Client Native	B) Client with RA/AP	C) Client via MQMONA
1	Copy 1000 x 100 byte messages	7m 52s	7s	0.984s
2	Copy 1000 x 1K messages	7m 57s	20s	1s
3	Copy 1000 x 10K messages	7m 55s	4m 58s	2s

These tests illustrate the benefits achieved when the messages don't have to be moved up to the client application and then back down to the queue manager.

These tests also illustrate the benefits that read-ahead and asynchronous put can bring you when using a network with high latency. The source queue was configured with `DEFREADA(YES)` and the target queue was configured with `DEFPRESP(ASYNC)` for configuration B. Since the messages are persistent, the client program also does the copy in a transaction which will give Asynchronous Put the best possible benefit.

While read ahead and asynchronous put can help a great deal, they are not changing the basic pattern which has the full message data going up to the client program and then back down to the queue manager. When using MQMONA, the message data goes from source to target queue without ever leaving the machine where the queue manager is running.

If you're wondering why the A column is essentially the same in every test, it's because the latency of the system is the gating factor. The client is able to do 2 MQGET calls and 2 MQPUT calls each second because of the required line turn-around.

20.3. Error Log Viewing

Using MQMONA to view error logs provides support for this feature for queue managers at versions prior to V9.4.5 (but must be at least V9.0.3). You can also use the MQMONA provided function with a V9.4.5 queue manager as well, which may help with performance.

20.3.1. Environment description

We compared the speed of the Inquire Error Log command and the results are shown in the table below. This was a local test – the queue manager was on the same machine as MO71 which was issuing the command and a locally bound connection was used. Clearly if a client connection was used, you would have to add time to send all the data from the queue manager machine to the client application. We have reported this behaviour of the V9.4.5 queue manager to IBM and hope that it will be rectified in a future release.

IBM MQ queue manager error logs are, by default 32MB each, but can be configured up to 2GB in size using the `ErrorLogSize` attribute in the `QMErrorLog` stanza of the qm.ini file.

20.3.2. Test results

To run the test below, we picked a time that would return the oldest 5 messages and a time that would result in only the most recent 5 messages being returned.

#	Test Description	Time Taken	
		IBM MQ V9.4.5	MQMONA
1	Reading the first 5 messages of an error log sized 32MB	1s	2s
2	Reading the last 5 messages of an error log sized 32MB	6s	1s
3	Reading the first 5 messages of an error log sized 650MB	26s	5s
4	Reading the last 5 messages of an error log sized 650MB	2m 34s	1s

As you know error logs can be even larger than the 650MB in our test. If you run with very large Error Logs then we suspect your performance would be considerably worse than shown here.

Chapter 21. Error reporting

On z/OS, when running as a batch job or a started task, MQMONA will write informational and error messages to the JESMSGLG. In all other environments (z/OS UNIX, AIX, Linux, Windows) these messages are written to a rotating set of error log files.

These files will be created either in a default location or a location you specify.

The default location is in the same directory as the executable when running on Windows or Linux and the current directory when running on AIX on z/OS UNIX.

You can specify a location to over-ride this default by using the `logpath` keyword in the configuration file. This keyword has a special substitution value `+QMNAME+` to allow the creation of a configuration file that can be used by multiple queue managers.

The error log file will be named `MQMONA_Log<nn>.txt` where *nn* is the number of the log file. By default there will be three rotating error log files, very similar to IBM MQ. When the first error log gets full, then it will be renamed to the next number and a new error log will be created.

The number of error log files can be configured using the `numlogfiles` keyword in the configuration file. The size at which an error log is considered to be full can be configured using the `maxlogsize` keyword.

The current error log being written to is always called `MQMONA_Log01.txt`.

Chapter 22. Messages

MQMONA writes a number of informational and error messages to the error log files, or JESMSGLG on z/OS. This chapter details those messages.

Some messages are only shown when you are running MQMONA with a specific verbosity level (`-v` parameter flag). This is detailed in each message.

MQGA001I MQGem Software MQMONA : Command Server Agent for IBM MQ
Version *version-number* Build Date: *date*
(C) Copyright MQGem Software 2023,*current-year*. (https://www.mqgem.com)

A welcome banner message showing the *version-number* and *date* the program was built.

Minimum Verbosity Level: Always shown

User Action Required: None

MQGA002I Current Date: *date*

When an error message is written to the error log on a new date, this message precedes it in the file. This message is not written to the JESMSGLG as every message has the date recorded.

Minimum Verbosity Level: Always shown

User Action Required: None

MQGA003I Program Triggered

The MQMONA program was started as a result of a trigger message.

Minimum Verbosity Level: Always shown

User Action Required: None

MQGA004I Parameters in effect:
Queue Manager : *qmgr-name*
Input Queue : *input-q-name*
Reply Queue : *reply-q-name*
Actual Reply Queue : *actual-reply-q-name*
Verbose Level : *verbose-level*

A brief report of the parameters in use. The *actual-reply-q-name* will be different from the *reply-q-name* if the *reply-q-name* was the name of a MODEL queue.

Minimum Verbosity Level: Always shown

User Action Required: If these do not match what you expect, ensure you have used the correct parameter flags.

MQGA005I Default overrides from file '*config-file*' in effect: *details*

The *details* will show the list of overrides read from the *config-file*. If there are keywords in the file that do not apply to the current mode of running, for example attributes that only apply if the program was triggered, they will not be shown.

Minimum Verbosity Level: Always shown

User Action Required: None

MQGA006E Usage: MQMONA <Optional flags as below>

```
  [-m <Queue Manager Name>]
  [-q <Input Queue Name>  ]
  [-f <Config File Name>  ]
  [-k                     ] Running as a service
  [-c <Simple command>    ]
  [-v {Quiet|Low|Med|High}] Verbosity level
```

The MQMONA program usage statement.

Minimum Verbosity Level: Always shown

User Action Required: If you did not request to see the usage statement, this means that you have incorrectly specified some parameter flags on your invocation of the program. Review any instances of error message **MQGA007E** to discover what was wrong.

MQGA007E Unrecognised parameter '*parm-flag*'

You have specified an unrecognised parameter flag '*parm-flag*'. Review the invocation of MQMONA and correct any flags that are not supported by MQMONA. A full description of the parameter flags can be read in Chapter 16. on page 53.

Minimum Verbosity Level: Always shown

User Action Required: Correct the parameter flags.

MQGA008E Integer configuration parameter '*attr-name*' value *num* exceeded max (*max-num*)

A configuration attribute *attr-name* was found in the configuration file to have a numeric value *num* which is greater than the maximum allowed value of *max-num*.

Minimum Verbosity Level: Always shown

User Action Required: Reduce the value given for *attr-name* to be less than *max-num* and restart MQMONA.

MQGA020I Disconnect Interval, *n* minutes, expired

MQMONA is stopping because there has been no work for it to do for at least *n* minutes. Disconnection only takes place when MQMONA is trigger started, so it will now be in a position to be restarted by a trigger message when more work arrives on the queue.

Minimum Verbosity Level: Always shown

User Action Required: None

MQGA021I STOP command received

MQMONA is stopping because it received a STOP command, either from a simple string message containing the text “STOP” on it's input queue or from an MVS STOP command.

Minimum Verbosity Level: Always shown

User Action Required: None – unless you did not expect a STOP command to have been issued.

MQGA080I MQMONA Status Report

This multi-line message will report a number of status attributes. For full details see Chapter 15. Status Reporting on page 52. This message is written at regular timed intervals; when prompted to do so with a REPORT simple command; and just before ending.

Minimum Verbosity Level: Always shown

User Action Required: None

MQGA099I MQMONA Agent shutting down

The MQMONA agent is shutting down. See preceding messages to understand the reason why it is shutting down.

Minimum Verbosity Level: Always shown

User Action Required: None

MQGA100E Error loading MQAPI mqm.lib rc=*return-code*

MQMONA failed to successfully load the MQ libraries. The *return-code* provides more information.

If the reason for the failure is not obvious, you can set the environment variable, `MQACCESS_DEBUG` for more information.

Minimum Verbosity Level: Always shown

User Action Required: Ensure the MQ environment is set up correctly, for example by using `setmqenv` or by having a primary installation.

MQGA101E MQCONN to queue manager '*qmgr*' failed with reason *mqrc* (*string*)

MQMONA was unable to connect to queue manager *qmgr*, The reason code *mqrc* and its *string* equivalent provide more information about the failure.

Minimum Verbosity Level: Always shown

User Action Required: Look up the *mqrc* in IBM Docs, or with the IBM MQ `mqrc` tool. Follow the guidance in IBM Docs to solve the problem. Remember that there is often additional information in the queue manager error log `AMQERR01.LOG` especially if the failure is `MQRC_NOT_AUTHORIZED`.

MQGA102E MQOPEN of queue manager object failed with reason *mqrc* (*string*)

MQMONA was unable to open the queue manager object for inquire. The reason code *mqrc* and its *string* equivalent provide more information about the failure.

Minimum Verbosity Level: Always shown

User Action Required: Look up the *mqrc* in IBM Docs, or with the IBM MQ `mqrc` tool. Follow the guidance in IBM Docs to solve the problem. Remember that there is often additional information in the queue manager error log `AMQERR01.LOG` especially if the failure is `MQRC_NOT_AUTHORIZED`.

MQGA103E MQINQ of queue manager object failed with reason *mqrc* (*string*)

MQMONA was unable to inquire the queue manager object. The reason code *mqrc* and its *string* equivalent provide more information about the failure.

Minimum Verbosity Level: Always shown

User Action Required: Look up the *mqrc* in IBM Docs, or with the IBM MQ `mqrc` tool. Follow the guidance in IBM Docs to solve the problem. Remember that there is often additional information in the queue manager error log `AMQERR01.LOG`.

MQGA104E MQOPEN of queue '*q-name*' failed with reason *mqrc* (*string*)

MQMONA was unable to open one of the queues it needs to operate, *q-name*. The reason code *mqrc* and its *string* equivalent provide more information about the failure.

Minimum Verbosity Level: Always shown

User Action Required: Look up the *mqrc* in IBM Docs, or with the IBM MQ `mqrc` tool. Follow the guidance in IBM Docs to solve the problem. Remember that there is often additional information in the queue manager error log `AMQERR01.LOG` especially if the failure is `MQRC_NOT_AUTHORIZED`.

MQGA105E MQCB for queue '*q-name*' failed with reason *mqrc* (*string*)

MQMONA was unable to set up a call-back function to read from queue *q-name*. The reason code *mqrc* and its *string* equivalent provide more information about the failure.

Minimum Verbosity Level: Always shown

User Action Required: Look up the *mqrc* in IBM Docs, or with the IBM MQ `mqrc` tool. Follow the guidance in IBM Docs to solve the problem. Remember that there is often additional information in the queue manager error log `AMQERR01.LOG`.

MQGA106E MQCTL on queue manager '*qmgr*' failed with reason *mqrc* (*string*)

MQMONA was unable to either start or stop the message consumers. The reason code *mqrc* and its *string* equivalent provide more information about the failure.

Minimum Verbosity Level: Always shown

User Action Required: Look up the *mqrc* in IBM Docs, or with the IBM MQ `mqrc` tool. Follow the guidance in IBM Docs to solve the problem. Remember that there is often additional information in the queue manager error log `AMQERR01.LOG`.

MQGA107E MQPUT to queue '*q-name*' failed with reason *mqrc* (*string*)

MQMONA was unable to put a message to the queue *q-name*. The reason code *mqrc* and its *string* equivalent provide more information about the failure. This message will be seen after message retry has been attempted and all retry attempts have been exhausted. No more attempts will be made to retry this message and it will be discarded.

Minimum Verbosity Level: Always shown

User Action Required: Look up the *mqrc* in IBM Docs, or with the IBM MQ `mqrc` tool. Follow the guidance in IBM Docs to solve the problem.

MQGA108E MQPUT to queue '*q-name*' failed with reason *mqrc* (*string*) - Retry #*retry-attempt* after *retry-interval* seconds

MQMONA was unable to put a message to the queue *q-name*. The reason code *mqrc* and its *string* equivalent provide more information about the failure. This put will be reattempted after *retry-interval* seconds have passed. The next put attempt will be number *retry-attempt*.

If all retry attempts are exhausted, error message **MQGA107E** will be written to the error log.

Minimum Verbosity Level: Always shown

User Action Required: Look up the *mqrc* in IBM Docs, or with the IBM MQ `mqrc` tool. Follow the guidance in IBM Docs to solve the problem.

MQGA109E MQCLOSE of queue '*q-name*' failed with reason *mqrc* (*string*)

MQMONA attempted to close the object handle of queue *q-name* in the cache, and was unable to. The reason code *mqrc* and its *string* equivalent provide more information about the failure.

The handle will still be removed from the cache.

Minimum Verbosity Level: Always shown

User Action Required: Look up the *mqrc* in IBM Docs, or with the IBM MQ `mqrc` tool. Follow the guidance in IBM Docs to solve the problem.

MQGA110E MQINQ of reply queue '*q-name*' failed with reason *mqrc* (*string*)

MQMONA attempted to inquire details about reply queue *q-name*, and was unable to. The reason code *mqrc* and its *string* equivalent provide more information about the failure.

Minimum Verbosity Level: Always shown

User Action Required: Look up the *mqrc* in IBM Docs, or with the IBM MQ `mqrc` tool. Follow the guidance in IBM Docs to solve the problem. To stop MQMONA from inquiring reply queues, use the `replyqinq` switch in the configuration file. See Chapter 18. Configuration File on page 55 for more details.

MQGA111I MQCLOSE of cached *open-type* handle to queue '*q-name*' successful

MQMONA closed an object handle of *open-type* to queue *q-name* either because

- it had not been used for at least the number of seconds specified in configuration attribute `hobjttl`.
- an `MQG_CMD_CLOSE_QUEUE` command requested its closure

The *open-type* shows the main open options in use on the handle that has been closed, and will be one of the following words/phrases:

- input/browse
- input/output
- input
- output
- browse

Minimum Verbosity Level: High – this informational message is only seen when running with high verbosity.

User Action Required: None – unless you did not expect the queue to be closed. See Chapter 18. Configuration File on page 55 for more details on the `hobjttl` attribute that controls how long object handles are kept in the cache.

See 9.7. Close Queue on page 39 for more details about the command that can close off previously opened queue handles.

MQGA112I A client-connection was made using channel '*channel-name*'. Certificate Event checking disabled

It is expected that MQMONA runs as a locally bound connection to the queue manager. In this case a client connection has been made using channel *channel-name*. It is not possible to do inquiries on the queue manager's key repository to detect expiring certificates over a client connection, so certificate event checking has been disabled.

Minimum Verbosity Level: Always shown

User Action Required: Run MQMONA as a locally bound application.

MQGA113E MQPUT of a progress message to queue '*q-name*' failed with reason *mqrc* (*string*)

MQMONA was unable to put a progress response message to the queue *q-name*. The reason code *mqrc* and its *string* equivalent provide more information about the failure. Message retry is not attempted for progress messages.

Minimum Verbosity Level: Always shown

User Action Required: Look up the *mqrc* in IBM Docs, or with the IBM MQ `mqrc` tool. Follow the guidance in IBM Docs to solve the problem.

MQGA120E Queue '*q-name*' has been GET(INHIBITED)

MQMONA has received notification that queue *q-name* has been inhibited for get. This will cause MQMONA to shut down.

Minimum Verbosity Level: Always shown

User Action Required: None – unless you did not expect the queue to have been get inhibited.

MQGA121E Command Q '*q-name*' opened *open-type* by Application '*app-name*', PID(*pid*)

MQMONA was unable to open its command queue, *q-name*, because it is already open by another application. MQMONA expects to open its command queue with exclusive access.

The application that has the command queue open for *open-type* access is *app-name* and is running with a PID of *pid*.

Minimum Verbosity Level: Always shown

User Action Required: Determine why there is another application using the MQMONA command queue. The most likely reason is that there is already an instance of MQMONA running, and the application name shown in this message should help to quickly determine that.

MQGA199E MQMONA received reason code *mqrc* (*string*)

The MQMONA consumer has received reason code *mqrc*. The reason code *mqrc* and its *string* equivalent provide more information about the failure.

Minimum Verbosity Level: Always shown

User Action Required: Look up the *mqrc* in IBM Docs, or with the IBM MQ `mqrc` tool. Follow the guidance in IBM Docs to solve the problem.

MQGA201E Can not allocate *n* bytes for a message!

MQMONA is trying to allocate a memory buffer to store an MQ message in, but was unable to allocate a buffer of *n* bytes in length.

Minimum Verbosity Level: Always shown

User Action Required: Unless you are running in a very memory restricted environment, report this issue to MQGem Software support.

MQGA202E Can not allocate *n* bytes for a '*struct-name*' structure

MQMONA is trying to allocate a memory buffer to store s structure called *struct-name* in, but was unable to allocate a buffer of *n* bytes in length.

Minimum Verbosity Level: Always shown

User Action Required: Unless you are running in a very memory restricted environment, report this issue to MQGem Software support.

MQGA203E Unable to open config file '*file-name*' rc=*errno* (*string*)

MQMONA was unable to open the provided configuration file, file-name. The reason code *errno* and its *string* equivalent provide more information about the failure.

Minimum Verbosity Level: Always shown

User Action Required: If the *string* equivalent of the *errno* does not immediately make the reason for the failure obvious, look up the *errno* in your platform documentation and follow the guidance there to solve the problem.

MQGA204E Unable to open error log file '*file-name*' rc=*errno* (*string*)

MQMONA was unable to open the error log file, *file-name*. The reason code *errno* and its *string* equivalent provide more information about the failure.

MQMONA will shut down. To force MQMONA to continue in this situation, use the `logfailfatal` switch in the configuration file. See Chapter 18. Configuration File on page 55 for more details.

Minimum Verbosity Level: Always shown

User Action Required: If the *string* equivalent of the *errno* does not immediately make the reason for the failure obvious, look up the *errno* in your platform documentation and follow the guidance there to solve the problem.

MQGA205I Deleting old error log file 'file-name'

MQMONA is rotating its error log files and is deleting the oldest one, file-name.

Minimum Verbosity Level: High – this informational message is only seen when running with high verbosity.

User Action Required: None, unless your error log files are rotating too frequently, in which case consider setting their maximum size to a larger value. For more details read Chapter 21. Error reporting on page 64.

MQGA206E Unable to delete old error log file '*file-name*' rc=errno (*string*)

MQMONA was unable to delete the error log file, *file-name*. The reason code *errno* and its *string* equivalent provide more information about the failure.

MQMONA will shut down. To force MQMONA to continue in this situation, use the `nologfailfatal` switch in the configuration file. See Chapter 18. Configuration File on page 55 for more details.

Minimum Verbosity Level: Always shown

User Action Required: If the *string* equivalent of the *errno* does not immediately make the reason for the failure obvious, look up the *errno* in your platform documentation and follow the guidance there to solve the problem.

MQGA207I Renaming old error log file '*file-name*' to '*new-file-name*'

MQMONA is rotating its error log files and is renaming an old error log file, *file-name*, to have a new name, *new-file-name*.

Minimum Verbosity Level: High – this informational message is only seen when running with high verbosity.

User Action Required: None, unless your error log files are rotating too frequently, in which case consider setting their maximum size to a larger value. For more details read Chapter 21. Error reporting on page 64.

MQGA208E Unable to rename old error log file '*file-name*' to '*new-file-name*' rc=*errno* (*string*)

MQMONA was unable to rename its own error log file, *file-name* to have a new name, *new-file-name*. The reason code *errno* and its *string* equivalent provide more information about the failure.

MQMONA will shut down. To force MQMONA to continue in this situation, use the `logfailfatal` switch in the configuration file. See Chapter 18. Configuration File on page 55 for more details.

Minimum Verbosity Level: Always shown

User Action Required: If the *string* equivalent of the *errno* does not immediately make the reason for the failure obvious, look up the *errno* in your platform documentation and follow the guidance there to solve the problem.

MQGA209E Error log File access with *func* failed. rc=*errno* (*string*)

Access on the current MQMONA error log file with function *func* failed. The reason code *errno* and its *string* equivalent provide more information about the failure.

MQMONA will shut down. To force MQMONA to continue in this situation, use the `logfailfatal` switch in the configuration file. See Chapter 18. Configuration File on page 55 for more details.

Minimum Verbosity Level: Always shown

User Action Required: If the *string* equivalent of the *errno* does not immediately make the reason for the failure obvious, look up the *errno* in your platform documentation and follow the guidance there to solve the problem.

MQGA210I Logging to '*file-name*'

This informational message is only written to `stdout` and is simply there to inform you the full location of the error log file, *file-name*, MQMONA is using to write out informational and error messages.

Minimum Verbosity Level: Always shown

User Action Required: None.

MQGA211E __console() failed, rc=*errno* (*string*)

MQMONA used the system function `__console()` which failed. The reason code *errno* and its *string* equivalent provide more information about the failure. This error message is only seen on z/OS.

Use of MVS `MODIFY` or MVS `STOP` commands will not be operational on this instance of MQMONA.

Minimum Verbosity Level: Always shown

User Action Required: Report this error to MQGem Software support.

MQGA212E MQMONA does not recognise the simple command '*string*'

A simple command has been entered, either by a simple string message on it's input queue or by using the MVS MODIFY command, but it was not one that MQMONA recognised. The command entered was *string*.

Minimum Verbosity Level: Always shown

User Action Required: Use a recognised simple command as detailed in Chapter 17. Simple Commands on page 54.

MQGA213E R_datalib callable service failed, SAF RC = *safrc*; RACF RC = *racfrc*; RACF reason = *reason*

MQMONA used the `R_datalib` callable service (`IRRSDL64`) to retrieve certificate information from a keyring. This call failed. The SAF return code *safrc*; RACF return code *racfrc* and RACF reason code *reason* can be used to determine why this failed. This error message is only seen on z/OS.

Minimum Verbosity Level: Always shown

User Action Required: Look up the reported return and reason codes in R_datalib: Certificate Data Library: Return and reason codes and follow the guidance there to solve the problem.

MQGA214E System SSL function *func* failed, rc=*fncrc* (*string*)

MQMONA used the System SSL function *func* which failed. The reason code *fncrc* and its *string* equivalent provide more information about the failure. This error message is only seen on z/OS.

Possible values of *func* are:-

- `gsk_decode_certificate`
- `gsk_name_to_dn`

Minimum Verbosity Level: Always shown

User Action Required: Report this error to MQGem Software support.

MQGA215I Inquiring Certificates in '*key-repository*' using runmqakm

MQMONA is about to invoke the runmqakm command to discover the contents of *key-repository*. This is either as a result of a user `DISPLAY CERT` command or MQMONA looking for expiring certificates. The preceding instance of either message **MQGA224I** or message **MQGA401I** will indicate which was the initiator.

This message is not seen on z/OS.

Minimum Verbosity Level: High – this informational message is only seen when running with high verbosity.

User Action Required: None, unless you don't expect any inquiries for either reason detailed above.

MQGA216I Inquiring Certificates in '*user-id**keyring*' using IRRSDL64

MQMONA is about to invoke the IRRSDL64 callable service to discover the contents of *keyring*. This is either as a result of a user `DISPLAY CERT` command or MQMONA looking for expiring certificates. The preceding instance of either message **MQGA224I** or message **MQGA401I** will indicate which was the initiator.

This message is only seen on z/OS.

Minimum Verbosity Level: High – this informational message is only seen when running with high verbosity.

User Action Required: None, unless you don't expect any inquiries for either reason detailed above.

MQGA217E Failed to run OS command *cmd-str*, return code *rc*

MQMONA attempted to run the OS command *cmd-str*, but this was not successful and reported a return code of *rc*.

This message is not seen on z/OS.

Minimum Verbosity Level: Always shown

User Action Required: Report this error to MQGem Software support.

MQGA218E Command *cmd-str* not found

MQMONA ran the OS command *cmd-str*, but the command was not found.

This message is not seen on z/OS.

Minimum Verbosity Level: Always shown

User Action Required: Ensure that OS command *cmd-str* is in the path available to the MQMONA program.

MQGA219E OS command *cmd-str* output text not recognised: *text*

MQMONA ran the OS command *cmd-str*, but output text from the command was not recognised. The first 100 characters of the unrecognised *text* is shown.

This message is not seen on z/OS.

Minimum Verbosity Level: Always shown

User Action Required: Report this error to MQGem Software support.

MQGA220E No valid licence found

MQMONA searched for MQGem Software licenses as described in Chapter 2. Licensing on page 3, but did not find any valid licences.

MQMONA will run in pass-through mode where all messages that arrive on the input queue will be passed straight onto the IBM MQ Command Server queue and MQMONA will not be involved with them further.

Minimum Verbosity Level: Always shown

User Action Required: Ensure your valid MQGem Software licences can be found by MQMONA.

MQGA221I *product-name* Licence details: *details*

MQMONA found a valid licence for *product-name*, and will allow messages from that software product to be consolidated and compressed before returning the replies to the application. Licence details are shown in *details*.

Minimum Verbosity Level: Low – this informational message is seen with a verbosity of at least low.

User Action Required: None.

MQGA222E Application '*product-name*' is unlicensed - request for ReplyQ '*q-name*' passed through

A command message from *product-name* arrived on the MQMONA input queue using a reply queue name of *q-name*. However, *product-name* is unlicensed so the command message was passed through to the IBM MQ Command Server queue and MQMONA will not be involved with that command further.

Minimum Verbosity Level: Always shown

User Action Required: Ensure your valid MQGem Software licences can be found by MQMONA.

MQGA223I Certificate Expiry Events are [enabled|disabled] MQEV licence [not found|found] certevent config set to '*config-setting*' [and SSLEV(*switch-setting*)]

This multi-line message describes whether certificate expiry events are enabled or disabled, and provides the reason why in the subsequent lines.

Certificate Expiry events will be enabled if all the following are true:-

- An MQEV licence can be located by MQMONA
- The `certevent` config setting is either `asqmgr` or `yes`
- If `certevent=asqmgr`, the queue manager SSLEV switch is ENABLED.

Otherwise, certificate expiry events will be disabled.

Minimum Verbosity Level: Always shown

User Action Required: None if the message describes the expected outcome. If the certificate events are not as expected, review the reasons conveyed in the message, and correct accordingly.

MQGA224I Checking Certificates in '*key-repository*' for Expiry

The MQMONA agent is about to check *key-repository* to discover whether any certificates have a remaining lifespan worth reporting with an event message.

Minimum Verbosity Level: Medium – this informational message is only seen when running with medium or high verbosity.

User Action Required: None if you expect MQMONA to be checking certificates for expiry. Otherwise review the settings shown in message **MQGA223I** and change them accordingly.

MQGA225E Unable to open '*key-repository*' runmqakm error message number *capi-error*

An attempt to view the contents of *key-repository* with the `runmqakm` command failed. The error message number returned was *capi-error*.

Minimum Verbosity Level: Always shown

User Action Required: Look up *capi-error* in the “Runtime messages” chapter of GSKCapiCmd Users Guide and follow the guidance there to solve the problem.

MQGA226E The key repository 'key-repository' does not exist.

An attempt to view the contents of *key-repository* failed because *key-repository* does not exist.

On Distributed platforms, the error message number returned to MQMONA by `runmqakm` was `CTGSK2101W` or `CTGSK3026W`.

On z/OS, the RACF reason code returned to MQMONA on the `R_datalib` call was 84.

Minimum Verbosity Level: Always shown

User Action Required: Ensure that the key repository exists.

MQGA227E An invalid key repository password was encountered for '*key-repository*'

An attempt to view the contents of *key-repository* with the `runmqakm` command failed because an invalid password was encountered.

This could also mean that the stash file does not exist, or the user ID MQMONA is running with does not have access to the stash file.

The error message number returned to MQMONA by `runmqakm` was `CTGSK2016W`.

Minimum Verbosity Level: Always shown

User Action Required: Ensure that the key repository password is stashed and that the user ID MQMONA is running with (and thus the `runmqakm` command is run with) has access to the stash file.

MQGA228I Certificate event written for DN '*subjects-distinguished-name*' with *lifespan* days left

The MQMONA agent found a certificate with less than the configured number of days left until expiry and wrote an event message detailing this to the SYSTEM.ADMIN.CHANNEL.EVENT queue. The certificate has a Subject's DN of *subjects-distinguished-name* and has only *lifespan* days left until expiry.

Minimum Verbosity Level: Low – this informational message is seen with a verbosity of at least low.

User Action Required: Arrange for a new certificate to replace this one sometime in the next *lifespan* days.

MQGA229E Checking Certificates failed

An error occurred which meant that certificate checking cannot proceed.

Minimum Verbosity Level: Always shown

User Action Required: Check the previous error messages to see more details.

MQGA230E The key repository '*key-repository*' is empty

An attempt to view the contents of *key-repository* failed because *key-repository* does not contain any certificates.

On z/OS, the RACF reason code returned to MQMONA on the `R_datalib` call was 44.

Minimum Verbosity Level: Always shown

User Action Required: Ensure that the key repository contains some certificates.

MQGA231I Time zone in runmqakm output date string not recognised but found to match local Output: '*date-string*'

MQMONA has attempted to parse a certificate Issue or Expiry date and time string (*date-string*) from the output of a `runmqakm` command, and has discovered a time zone that is not known.

This time zone has been discovered to be the same as the local time zone, so the certificate time field in question has been able to be processed.

This message will only be shown once for each unique unknown time zone.

Minimum Verbosity Level: Always shown

User Action Required: Report this error to MQGem Software support so our list of time zones may be updated.

MQGA232E Time zone in runmqakm output date string not recognised Output: '*date-string*'

MQMONA has attempted to parse a certificate issue or expiry date and time string (*date-string*) from the output of a `runmqakm` command, and has discovered a time zone that is not known.

Displays of the certificate details will show the `ISSUETI` and `EXPIRYTI` fields as an unknown time (probably a question mark).

This message will only be shown once for each unique unknown time zone.

Minimum Verbosity Level: Always shown

User Action Required: Report this error to MQGem Software support so our list of time zones may be updated.

MQGA233E Unable to parse runmqakm output date string Output: '*date-string*'

MQMONA has attempted to parse a certificate issue or expiry date and time string (*date-string*) from the output of a `runmqakm` command, and has found an unexpected format.

Displays of the certificate details will show the `ISSUETI` and `EXPIRYTI` fields as an unknown time (probably a question mark).

Minimum Verbosity Level: Always shown

User Action Required: Report this error to MQGem Software support so new formats of this output can be handled.

MQGA234E Worker Thread Limit *n* reached

A request was made to MQMONA that needs to be scheduled on a worker thread (see Chapter 19. Worker Threads on page 60 to understand which types of commands these are). There are no worker threads available as they are all busy processing other requests, and no new threads can be started as the maximum number *n* allowed by configuration attribute `maxwrkthrd` have already been started. The new request will wait until there is a free thread to pick it up.

Minimum Verbosity Level: Always shown

User Action Required: If the delay to users that this causes is not acceptable, increase the value of `maxwrkthrd`.

MQGA235E File function *func* failed on *file-name*. rc=*errno* (*string*)

MQMONA used the file function *func* on file *file-name* which failed. The reason code *errno* and its *string* equivalent provide more information about the failure.

Possible values of *func* are:-

- `fstat()`
- `fseek()`

Minimum Verbosity Level: Always shown

User Action Required: If the *string* equivalent of the *errno* does not immediately make the reason for the failure obvious, look up the *errno* in your platform documentation and follow the guidance there to solve the problem.

MQGA236E Internal Error '*string*'

When reading MQ error log files, MQMONA encountered a situation that should not happen.

Minimum Verbosity Level: Always shown

User Action Required: Report this error to MQGem Software support.

MQGA237E Can not open file '*file-name*'. rc=*errno* (*string*)

MQMONA tried to open the IBM MQ Error log file *file-name* which failed. The reason code *errno* and its *string* equivalent provide more information about the failure.

If you are using the configuration attribute `mqerlPath`, ensure the file-name shown is in the expected place. MQMONA will added the queue manager specific part of the path to what is specified in `mqerlPath`.

Minimum Verbosity Level: Always shown

User Action Required: If the *string* equivalent of the *errno* does not immediately make the reason for the failure obvious, look up the *errno* in your platform documentation and follow the guidance there to solve the problem.

MQGA301E Can not compress PCF type *n*

MQMON can compress all the PCF types that are found in IBM MQ PCF commands. If a new PCF type is used in a command that MQMONA does not currently cater for, no compression will take place for that command attribute, but the command responses will still be returned to the application.

Minimum Verbosity Level: Always shown

User Action Required: If this error is still seen when using the latest version of MQMONA, contact MQGem Software support to have an improvement made to MQMONA to handle the reported PCF type *n*.

MQGA302E Unrecognised reply received MsgId: *message-id*

MQMONA has received a reply message for a command it has no record of sending. This is most likely to happen if the response to the command was delayed in some way, for example because of a retrying channel, or because it spent some time on a Dead-letter queue. MQMONA does not remember about the commands it has issued forever, instead ageing them out after a configurable number of seconds.

Minimum Verbosity Level: Always shown

User Action Required: You should see the same message id reported in an instance of error message **MQGA303E** which will record the Reply queue that was linked to this message. Determine whether the administration tool using that Reply queue is using a “via connection”, i.e. one going across MCA channels, and consider increasing the time period after which commands are aged out of MQMONA's memory. You can do this using the `maxage` keyword in the configuration file.

If this is not a message using a via connection, investigate whether reply messages went via a Dead-letter queue.

If this message is regularly seen, consider increasing `maxage` as noted above.

MQGA303E Ageing out request for ReplyQ '*q-name*' issued at *time* MsgId: *message-id*

MQMONA is ageing out, that is deliberately forgetting about, a command request that was issued at *time* but for which no response have yet been returned. This command used Reply queue *q-name* and had a message of *message-id*.

This is most likely to happen if the response to the command was delayed in some way, for example because of a retrying channel, or because it spent some time on a Dead-letter queue. If a response message later turns up for a command MQMONA has deliberately forgotten about, error message **MQGA302E** will be reported and you can tie this error message to that one using the message id.

Minimum Verbosity Level: Always shown

User Action Required: See error message **MQGA302E**

MQGA304I ReplyQ '*q-name*' (*deftype*) has smaller MaxMsgLen (*q-max-len*) than configured maxmsgsize (*mqmona-max-len*). Consolidation size reduced accordingly

MQMONA inquired reply queue *q-name* and discovered that its Maximum Message Length was smaller than the MQMONA configured size, *mqmona-max-len*, set using `maxmsgsize` (see Chapter 18. Configuration File on page 55 for more details on how to set it). As a result, MQMONA will use a smaller consolidation size for messages sent to this reply queue.

Minimum Verbosity Level: Always shown

User Action Required: The application using reply queue *q-name* will continue to operate successfully but will not be getting the same efficiency savings as other applications whose reply queues have a larger Maximum Message Length. It is suggested that you increase the Maximum Message Length of the reply queue to be at least that which is used by MQMONA, *mqmona-max-len*. If the reply queue is a dynamic queue (which is shown by *deftype*) then you will have to make the change to the MODEL queue it was created from.

MQGA401I Received PCF '*pcf-command*' command message of length *n* bytes

MQMONA has received a PCF command message for command *pcf-command* on its input queue. It has passed this message onto the IBM MQ Command Server queue after changing the reply queue in the MQMD.

Minimum Verbosity Level: Medium – this informational message is only seen when running with medium or high verbosity.

User Action Required: None.

MQGA402I Received MQSC command message of length *n* bytes

MQMONA has received an MQSC command message on its input queue. It has passed this message onto the IBM MQ Command Server queue after changing the reply queue in the MQMD.

Minimum Verbosity Level: Medium – this informational message is only seen when running with medium or high verbosity.

User Action Required: None.

MQGA403I Received reply message of length *n* bytes

MQMONA has received a reply message on its reply queue. It will compress and consolidate this reply message with others for the same command before sending a large reply message onto the application's reply queue.

Minimum Verbosity Level: Medium or High – this informational message is only seen when running with high verbosity for commands that have multiple replies. It will be seen at Medium verbosity when a command only has one reply in total.

User Action Required: None.

MQGA404I Received *count* reply messages of total length *n* bytes

MQMONA has received a *count* reply messages on its reply queue. It will compress and consolidate these reply messages with others for the same command before sending a large reply message onto the application's reply queue.

Minimum Verbosity Level: Medium – this informational message is only seen when running with medium verbosity. When running at High verbosity, message **MQG403I** is issued multiple times, once for each reply message received.

User Action Required: None.

MQGA405E PCF Message failure *mqrc* (*string*)

While processing a PCF format IBM MQ message, a failure occurred. The reason code *mqrc* and its *string* equivalent provide more information about the failure.

Minimum Verbosity Level: Always shown

User Action Required: Report the problem to MQGem Software support.

MQGA406I Issued PCF '*pcf-command*' command

MQMONA has issued a PCF command to the queue manager.

Examples of this include issuing an Inquire Queue Manager command in order to discover the Key Repository name so that a Display Certificate command can be satisfied.

Minimum Verbosity Level: Medium – this informational message is only seen when running with medium or high verbosity.

User Action Required: None.

MQGA450I Error log command processing is not active

The Inquire Error Log command processing has not been enabled.

This message is not seen on z/OS.

Minimum Verbosity Level: Always shown

User Action Required: If you need MQMONA to handle Inquire Error Log commands, enable it using the `mqerlCmd` switch in the configuration file.

MQGA451I Error log command processing not supported on pre 9.0.3 Queue Managers. Current CmdLevel is *cmd-level*

MQMONA relies upon the ISO 8601 format message time included in an error message when it is written to the IBM MQ error log. This was added in IBM MQ V9.0.3.

The current queue manager has a command level of *cmd-level* which is older than this.

Minimum Verbosity Level: Always shown

User Action Required: Upgrade your queue manager to a version newer than V9.0.3.

MQGA452I Error log rollover detected

MQMONA is reading the IBM MQ error logs and has detected that the files have been renamed since it last processed them. This may result in a slight longer processing time for the Inquire Error Log command that caused this detection.

Minimum Verbosity Level: Always shown

User Action Required: None.

MQGA453E Error log response data limit of *n* bytes reached

While reading an IBM MQ error log, MQMONA reached the number of bytes, n, configured as the maximum amount of data to be returned in response to an Inquire Error Log command.

The application issuing the response will be returned MQGRCCF_DATA_INCOMPLETE.

Minimum Verbosity Level: Always shown

User Action Required: Request a smaller portion of the error log, or increase the limit by configuring the attribute `mqerlRespLimit`.

MQGA454I Inquire Error Log Command : From *iso-time*

This high detail message shows the time parameter sent as part of the Inquire Error Log Command.

Minimum Verbosity Level: High – this informational message is only seen when running with high verbosity.

User Action Required: None.

MQGA455I Opening error log file '*file-name*'

MQMONA is about to open the IBM MQ error log file *file-name*. The file name is shown with its full path.

Minimum Verbosity Level: High – this informational message is only seen when running with high verbosity.

User Action Required: None.

MQGA456I Closing error log file *file-name*

MQMONA is about to close the IBM MQ error log file *file-name*. The file name is shown without its path.

Minimum Verbosity Level: High – this informational message is only seen when running with high verbosity.

User Action Required: None.

MQGA457I Read error log *file-name* : *x* bytes at offset *n*

MQMONA is about to read data from the IBM MQ error log file file-name. It will read *x* bytes from the file starting at file position *n*.

Minimum Verbosity Level: High – this informational message is only seen when running with high verbosity.

User Action Required: None.

MQGA458I Free error log chunk : *file-name* at offset *n* Bytes *x* Msgs: *m*

MQMONA holds previously read chunks of the IBM MQ error log file in memory to provide more efficient answers when multiple requests for the same data might be made. After a period of time, housekeeping of these chunks will free off any that have exceeded the time period to be kept.

MQMONA has freed a chunk originally read from error log *file-name* starting from file position *n*, that was *x* bytes long and found to contain *m* error messages.

Minimum Verbosity Level: High – this informational message is only seen when running with high verbosity.

User Action Required: None, unless you want to increase the timespan that previously read data is kept, or reduce the amount of memory in use by MQMONA by reducing the timespan. In which case review the values used in configuration attributes `mqerlChunk`, `mqerlKeepAge` and `mqerlTTL`.

Chapter 23. Response Messages

MQMONA writes a number of informational and error messages as response messages to administrative applications. This chapter details those messages.

MQGA501E No response from IBM MQ Command Server

MQMONA forwarded the users MQ command to the IBM MQ Command Server and no response was received before the interval configured by the `maxage` attribute was exceeded.

This message will be seen in the PCF or MQSC response the administrative application received from MQMONA.

User Action Required: See Chapter 14. Timing out commands on page 51 for more information.

MQGA502I No Matching objects found

An MQSC command received by MQMONA, to be processed by MQMONA and not sent on to IBM MQ, found no objects to return as matching the attributes requested on the command.

User Action Required: None, unless some objects were expected to be returned.

MQGA550I Browse operation aborted by user

The in-progress browse operation was aborted by the user.

User Action Required: None, unless the user did not choose to abort the browse operation.

MQGA551E Browse operation aborted by system

The in-progress browse operation was aborted by MQMONA because it was shutting down.

User Action Required: None, unless it was unexpected that MQMONA was shut down.

MQGA552E [Copy/Move] of Message *num* failed. No header and no target queue given

A copy or a move of messages failed. Message number *num* was found to have no header, and a target queue name was not provided.

User Action Required: Provide the target queue where this message should be copied or moved to.

MQGA553E [Copy/Move] of Message *num* failed. No target queue given and 'strip header' not specified

A copy or a move of messages failed at message number *num*. No target queue name was provided and the ‘strip header’ option was not used.

User Action Required: Provide the target queue where this message should be copied or moved to, or if the target is to be found in a header on the message, select the ‘strip header’ option.

MQGA554E [Copy/Move] of message *num* failed. Opening queue '*qmgr-name/q-name*', failed with reason *mqrc* (*string*)

A copy or a move of messages failed. Message number *num* could not be put to the target queue *q-name* on *qmgr-name* because opening it failed. The reason code *mqrc* and its *string* equivalent provide more information about the failure.

User Action Required: Look up the *mqrc* in IBM Docs, or with the IBM MQ `mqrc` tool. Follow the guidance in IBM Docs to solve the problem.

MQGA555E [Copy/Move] of message *num* failed. Putting to queue '*qmgr-name/q-name*', failed with reason *mqrc* (*string*)

A copy or a move of messages failed. Message number *num* could not be put to the target queue *q-name* on *qmgr-name* because opening it failed. The reason code *mqrc* and its *string* equivalent provide more information about the failure.

User Action Required: Look up the *mqrc* in IBM Docs, or with the IBM MQ `mqrc` tool. Follow the guidance in IBM Docs to solve the problem.

MQGA556I [Copy/Move/Delete] of message *num* failed. Operation aborted by user

A copy or a move of messages was aborted by the user at message number *num*.

User Action Required: None, unless the user did not choose to abort the copy or move operation.

MQGA557E [Copy/Move/Delete] of message *num* failed. Operation aborted by system

A copy or a move of messages was aborted at message number *num* by MQMONA because it was shutting down

User Action Required: None, unless it was unexpected that MQMONA was shut down.

MQGA558I The message is no longer on the queue

When browsing the details of a message on a queue, it is found that the message is no longer on the queue. This could happen because other applications are also interacting with the queue, or perhaps because the message has expired.

User Action Required: None, unless this is an unexpected situation.

MQGA601E Syntax error detected at or near end of command segment below:-

An MQSC command received by MQMONA, to be processed by MQMONA and not sent on to IBM MQ, contained a syntax error.

The command is displayed up to the point where the syntax error was detected and is followed a message showing the valid syntax for the command.

User Action Required: Review the valid syntax for the command, correct it and reissue the command. See Chapter 9. Command Reference on page 19 for full details of the command syntax for MQMONA commands.

MQGA602E Parameter '*parm-name*' is too long

An MQSC or PCF command received by MQMONA, to be processed by MQMONA and not sent on to IBM MQ, contained a parameter that is longer than its maximum allowed length. The MQSC keyword for the parameter is given in *parm-name*.

If this was a PCF command, the PCF parameter id is also returned in the response message.

User Action Required: Review the maximum length for the parameter, correct it and reissue the command. See Chapter 9. Command Reference on page 19 for full details of the command syntax for MQMONA commands.

MQGA603E Parameter '*parm-name*' is not a valid integer

An MQSC command received by MQMONA, to be processed by MQMONA and not sent on to IBM MQ, contained a parameter *parm-name* that should contain an integer value but contains characters that are not a valid integer.

User Action Required: Review the valid syntax for the command, and ensure that *parm-name* supplies an integer value. See Chapter 9. Command Reference on page 19 for full details of the command syntax for MQMONA commands.

MQGA604E Parameter '*parm-name*' should be an integer

A PCF command received by MQMONA, to be processed by MQMONA and not sent on to IBM MQ, contained a parameter *parm-name* that should be an integer (i.e. an MQCFIN) but was sent as some other type.

User Action Required: Review the valid syntax for the command, and ensure that *parm-name* is sent as an integer. See Chapter 9. Command Reference on page 19 for full details of the command syntax for MQMONA commands.

MQGA605E Parameter '*parm-name*' is not a recognised value

A PCF command received by MQMONA, to be processed by MQMONA and not sent on to IBM MQ, contained a parameter *parm-name* whose value should be one of a small set of possible values, and the provided value was not recognised.

User Action Required: Review the valid syntax for the command, and ensure that the value of *parm-name* is one of the expected values. See Chapter 9. Command Reference on page 19 for full details of the command syntax for MQMONA commands.

MQGA606E Parameter '*parm-name*' should be a string

A PCF command received by MQMONA, to be processed by MQMONA and not sent on to IBM MQ, contained a parameter *parm-name* that should be a string (i.e. an MQCFST) but was sent as some other type.

User Action Required: Review the valid syntax for the command, and ensure that *parm-name* is sent as a string. See Chapter 9. Command Reference on page 19 for full details of the command syntax for MQMONA commands.

MQGA610E Unknown variable '*var-name*'

An MQSC or PCF command received by MQMONA, to be processed by MQMONA and not sent on to IBM MQ, contained a WHERE clause with an unknown variable name *var-name*.

User Action Required: Review the valid syntax for the command, and ensure the WHERE clause is referencing a valid attribute to match against. See Chapter 9. Command Reference on page 19 for full details of the command syntax for MQMONA commands.

MQGA611E Variable '*var-name*' is ambiguous

An MQSC or PCF command received by MQMONA, to be processed by MQMONA and not sent on to IBM MQ, contained a WHERE clause with an abbreviated variable name *var-name* that is ambiguous.

Enough of a variable name must be provided to ensure it can be unambiguously interpreted.

User Action Required: Review the valid syntax for the command, and ensure the WHERE clause is referencing enough of the attribute name to match against to ensure it can be unambiguously interpreted. If you are not certain, simply use the full name of the attribute. See Chapter 9. Command Reference on page 19 for full details of the command syntax for MQMONA commands.

MQGA618E Command Failed (*mqrc*) *mqrc-str*

The command issued has failed for reason *mqrc*. The string version of the error is *mqrc-str*.

Further information may also be written to the MQMONA log file. Examples of errors that might be reported include:-

- Queue Manager Key Repository does not exist
- Queue Manager Key Repository has no stash file or access to the stash file is not permitted
- The command being issued, e.g. runmqakm, does not exist in the path.

Use Action Required: Correct the problem reported by the combination of the mqrc, and error messages in the MQMONA log file.

MQGA619E Invalid Parameter (*parm-id*) *parm-str*

A parameter supplied on a command was invalid. The parameter ID is *parm-id*, and the string version of the parameter is *parm-str*.

User Action Required: Review the valid syntax for the command and also see Chapter 9. Command Reference on page 19 for full details of the command syntax for MQMONA commands.

MQGA620E Valid syntax for the MQMONA command:

The text shown is the valid syntax for the MQMONA command.

User Action Required: Review the valid syntax for the command and also see Chapter 9. Command Reference on page 19 for full details of the command syntax for MQMONA commands.

MQGA700I Display Certificate details

The DISPLAY CERT command completed successfully. Details follow this message.

User Action Required: None.

Chapter 24. 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.

24.1. Changes made in version 9.4.0

The main changes from the previous version of the program are:

1. **Command to display certificate details**
   MQMONA can now provide responses to a DISPLAY command to show the certificate details found in the KDB or PKCS#12 file being used by the distributed queue manager, or the keyring being used by the z/OS queue manager it is connected to. See 9.1. DISPLAY CERT on page 19 for full details.
2. **Generation of event messages for Certificates nearing Expiry**
   MQMONA will now regularly interrogate the queue manager's key repository to discover if any certificates have a remaining lifespan less than a configurable length, and if so, will write an event message to report this fact. See Chapter 7. Certificate Events on page 13 for full details.
3. The build date of MQMONA is added to the fields returned on the Inquire Queue Manager PCF command and shown in the MQMONA tab in MO71. See 3.1. Configuring MO71 to use MQMONA on page 5 for a screenshot.
4. The configuration file is updated to include an attribute for setting the licence file location instead of using the MQGEML environment variable. See 2.1. Licence File Location on page 3 and 18.2. Configuration File keywords on page 55.
5. Unknown time zones seen in `runmqakm` output, are reported in new error messages **MQGA231I** and **MQGA232E**. Report these to us to improve our list of recognised time zones.
6. IBM MQ V9.4.4 changes the behaviour of the queue manager when the TLS Key Repository supplied in the SSLKEYR queue manager attribute has no file extension. Previously it was assumed to be a .kdb file. In V9.4.4 it can now be either a .p12 file or a .kdb file, with the former looked for first. MQMONA is updated to match this behaviour. See 9.1.8. Default Key Repository file extension on page 25 for details.
7. A new date and time string format is seen on some platforms with IBM MQ V9.4.4. MQMONA is updated to understand it.
8. Linux for IBM Z added as a platform.

24.2. Changes made in version 9.3.0

The main changes from the original version of the program are:

1. **Message Consumers**
   MQMONA has been changed to use the IBM MQ Asynchronous Consumer interface instead of polling MQGET calls.
2. **Object Handle Cache**
   MQMONA will cache the object handles that it uses to improve the efficiency of subsequent requests as it is likely that a requester will come back again with another request.
3. **Reply when timed out**
   Commands that get no reply from the IBM MQ Command Server are handled more visibly. See Chapter 14. Timing out commands on page 51

4. **MQMONA can be Triggered**
   To ensure MQMONA is always available to process your command messages, you can define MQMONA to run as a SERVICE object (see Chapter 11. on page 46) and/or triggered upon the arrival of command messages (see Chapter 12. Triggering on page 47).

5. **Status Reporting**
   MQMONA reports status on a number of items that are interesting. For more details see Chapter 15. Status Reporting on page 52.

6. **Licensing**
   MQMONA is licensed based on your pre-existing MQGem product licenses. For more details see Chapter 2. Licensing on page 3.

7. **Configuration File**
   In addition to the main program parameters, MQMONA has a number of other optional configuration attributes supplied to it in a configuration file which are detailed in Chapter 18. Configuration File on page 55.

8. **Numbered Error Messages**
   When MQMONA needs to report some information or an error, it does so by writing a message with a message number. See Chapter 22. Messages on page 65 for a full catalogue of the error messages output by MQMONA.

9. **Error messages written to JESMSGLG on z/OS**
   When running MQMONA as a batch job or started task on z/OS, error messages are written to the JESMSGLG.

10. **Error Log file**
    On distributed platforms, and on z/OS when running in z/OS UNIX, MQMONA will write error messages to a rotating error log file. See Chapter 21. Error reporting on page 64 for full details.

11. **Stopping MQMONA**
    The MQMONA agent can be asked to stop gracefully with a STOP command, issued either by running the MQMONA executable with a simple stop command, or by using the MVS STOP command.

Chapter 25. Migration from previous versions

We always try to ensure that, as each version is shipped, all the features that were working on the previous versions remain intact. When installing a new version, we always recommend you backup the previous MQMONA program and the configuration file `MQMONA.CFG`. 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.

25.1. Migrating from a version prior to V9.4.1 Build Date 09-09-2026

If you have not granted the authorities documented in the MO71 and MQSCX user guides for these respective tools, you will find that you will need all those authorities as documented and the additional authorities documented in <insert link> to run MO71 and/or MQSCX using MQMONA. Previous to Build Date 09-09-2026 there were some situations where not all those authorities would be needed and this has now been rectified.

End of Document