IBM MQ Q Utility User Guide

Version 9.4.0

4th March 2025

Paul Clarke

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”

Tenth Edition, March 2025

This edition applies to Version 9.4.0 of the *IBM MQ Q Utility* and to all subsequent releases and modifications until otherwise indicated in new editions.

Notices

Table of Contents

History

SupportPac

The MA01 (Q Utility) SupportPac was one of the first MQ SupportPacs created. It has been a useful tool in the MQ administrators tool box for many years since its first release in June 1995.

It was originally created because the **`amqsput`**, **`amqsget`** and **`amqsbcg`** samples were lacking in some basic application control features such as setting the persistence or priority of a message, and has grown from there.

MQGem Product

In October 2012, after working at IBM for 28 years, most of which were spent in IBM MQ Product Development, I decided to leave IBM to pursue a different career writing tools and utilities for IBM MQ. To that end I founded MQGem Software (www.mqgem.com).

The Q product is heavily based on the MA01 SupportPac. A number of our customers asked for us to include Q in our product portfolio as the MA01 SupportPac has been discontinued and they were not keen to run unsupported code anyway. By having a Q product now customers can get new features added as well as support for the later IBM MQ versions.

To make Q a product we have made some changes for the sake of consistency within the Q product and our other products. As such there are minor differences between Q and MA01 in terms of both parameters and output. We strongly suggest that you try out the free trial of Q before you make a purchase.

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. IBM MQ Q Utility

1.1. Overview

The IBM MQ Q Utility (Q for short) is a veritable pocket knife of features, allowing many of the common tasks when interacting with messages on queues or topics, to be done from this single tool.

Q is a simple pipe line program which takes messages from one source and outputs to a target. The operation of the 'pipe' is controlled by switches to the program. The source and target can either be the console, a file, or an IBM MQ Queue. In addition, since input/output can be taken from stdin/stdout these can also be simple files. Two instances of Q can therefore be used to implement a primitive file transfer program.

1.2. Changes from previous version

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

1. **User Format messages**
   Support for MQGem User Formats, using the same format files as you may already be familiar with from using MQEdit. For more information see ***10.2. User Format Messages*** on page 46.

2. **End of Structure Detail Level**
   Various MQ formats and user formats, have a marker to show the end of the structure. Configure whether these markers are shown as low, medium or high detail with the `-dE:`*`<level>`* flag.

3. **Print hex as characters**
   Print hex fields with the characters represented by the hex field below the field using the `-dk` flag.

4. **Formatted Put Date and Time**
   When displaying the Message Descriptor, the Put Date and Put Time fields will show a formatted version over on the right to ensure the date conveyed is unambiguous.

5. **Ability to strip headers, such as Dead Letter Queue Header or MQHRFH**
   You can now ask Q to remove any standard header from the message as it is displaying, copying or moving messages. Please see Chapter **9.9 "Stripping Headers"** on page 40 for more details.

6. **Ability to remove Report Options**
   It is now possible to remove Report options from a message using the **`-n`** parameter.
   For example, using the command: **`q -iQ1 -oQ2 -n!*`**
   will remove any Report options specified in message descriptor as messages are transferred.

   Please see the **`-n`** parameter described in Chapter **5.2 Parameters Flags** on page 9 for more information.

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
Windows	Extract using your favourite zip utility
Z/OS	Extract using your favourite zip utility before transferring to a z/OS system. See ***1.3.3 z/OS Installation Instructions*** below for more details

1.3.1. Linux, Unix and Windows

Once unzipped you should have the **q** 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. Unix Compatibility

A number of Unix platforms tend to be fairly bad at maintaining compatibility with future versions of the C runtime so it may be that the latest version of Q would not run on an older version of Linux. I do, wherever possible, try to build Q on older versions of the Operating System however it may not always be possible to go back far enough. If you have problems running Q, particularly if it complains about GLIBC version, then please contact support@mqgem.com and we'll see what we can do.

1.3.3. z/OS Installation Instructions

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

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

```
receive inds(Q.SEQ)
```

When prompted for a filename, reply

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

Q can be run on z/OS in BATCH – an example piece of JCL is provided in the zip file. Q 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.

1.3.4. z/OS UNIX Installation Instructions

If you wish to run Q 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(Q)' '/u/gemuser/bin/q' BIN
```

Chapter 2. Licensing

To access all the features (beyond any trial period) of Q you will require a licence file. The licence file also entitles the user to email support. If you would like to try out Q for free then a 1-month trial licence can be obtained by sending an email to support@mqgem.com.

Each licence is for a certain period of time, usually one year. There are number of advantages of this scheme:

- **Customers of Q get more dedicated support**
  Resources can be targeted towards customers who have made a financial contribution to the development of Q.
- **Purchasing decision is simpler**
  The Q licence covers a period of time not a release. Any licence, will enable the user to run any version of Q. It is therefore not necessary to concern oneself about whether a bigger, better version is about to come out soon since whatever licence you buy now will also work for that version.
- **Features are available sooner**
  Using this model it is not necessary for us to collect a large group of features together to 'justify' a new release of Q. Instead a new release can be made available whenever a new feature is added which is regarded as sufficiently useful since all current users will be able to migrate to the new version at no cost to themselves.

There are five types of licences which allow different levels of flexibility about who can run the program. Essentially this is controlled by the presence, or not, of userid, machine or location fields.

Type	Fields Set	Description
Emerald	userid, machine	Q is only supported on one machine using one userid.
Ruby	machine	Q is supported by any number of users using it on the same machine.
Sapphire	userid	Q is supported on any number of machines using the same userid.
Diamond	location	Q is supported by any number of users at the same site on any set of machines. The location field gives the location, for example “London, England” of where the licence is based.
Enterprise	none	Q can be run by any number of users within your company, world-wide. An Enterprise licence is priced at three times the Diamond licence price.

2.1. Userid and Machine Information

Some licence types limit the execution of the program to particular machines and userid. It is important, therefore, that when you purchase these licences that you specify these values correctly. If you are in any doubt run the Q program with the following command on the machine you wish to run Q on.

```
q -vm
```

This will print out the User Identifier and Machine Name values that you should include in your licence order.

2.2. Licence File Location

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

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

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

2.3. Multiple licences

If you have multiple licences then they can be concatenated into a single **mqgem.lic** file. This can be done using simple OS commands such as **copy** or by using your favourite editor.

2.4. Licence Renewal

An extra years licence can be purchased at any time and a new licence will be sent which extends the current licence by a year. There is therefore no concern with losing time by renewing early.

2.5. Changing your licence file

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

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

Chapter 3. Introduction

The Q program can be useful for a wide variety of tasks.

3.1. Uses

Q can be used for a huge range of tasks which include the following:

- **To write to one or more queues/topics**
  - One of the simplest to understand, and yet most powerful features of the Q program is putting messages onto a queue. In doing so you can set all the Message Descriptor fields (format, persistence, priority...) including context fields, if you have authority, and you can add Message Properties to the messages you put as well. You have more flexibility than that available with the `amqsput` sample. Message data can be taken from the console, from files or randomly generated, depending on your requirements.
- **To browse or get messages from a queue**
- **To display formatted messages**
  - The Q program can display your messages in hex or, more helpfully, formatted appropriately based on the contents. It understands all the MQ defined formats, a number of common industry standard formats and can format out your own message formats if you provide an MQGem Format File describing that format. It can also format and display any message properties on your messages, either natively or forced into an MQRFH2 header. You have much more flexibility than that available with the `amqsget` and `amqsbcg` samples.
- **Move messages between queues**
- **Copy messages between queue**
- **Queue Replicator**
  - You can use the Q program to copy (or move) messages from one queue to another. The target can be a single queue or multiple queues. This allows you to use Q as a queue replicator, taking a feed of messages from one queue and duplicating them onto two or more queues. This can be done without using publish/subscribe and therefore can maintain all the fields in the Message Descriptor as required.
- **Testing IBM MQ infrastructure**
  - The Q program will be very useful to aid in testing your IBM MQ infrastructure. It can be used to mimic a server application, echoing messages back to the ReplyTo fields in the Message Descriptor. This can be further tuned by configuring a “work time” to mimic the time spent by a server application to do the requested action, and the Q program can be triggered to further mimic expected or existing applications when load testing the infrastructure.
  - Q can also be used to drive the workload through the system, sending messages at regular or randomised intervals onto queues to be processed. The sizes of these messages can be regulated to match those likely to be used by your own applications.
  - Publish/subscribe can also be used through Q so you can test the topic tree and subscription management of IBM MQ using this utility as well.

Chapter 4. Examples

Q can be useful for a number of tasks. As you will see, as you learn more about the features of Q, the examples below are just a few of the things you can do with Q. This just gives you a quick idea about what problems it can solve. All of these examples can be modified with the use of a number of other parameters which are documented in “Chapter 5: Parameters” on page 9.

4.1. Example 1. Write to a queue

You can write a message to a queue by invoking the Q program as follows and then typing in your message text at the prompt. A new message is generated each time the return key is pressed. Typing `#end` at the prompt closes the program.

```
q -m QM1 -o Q1
```

Typing in a message that starts with the '`#`' character has a special meaning as discussed in ***“9.1. Q message operation syntax“*** on page 30.

4.2. Example 2. Write to two queues

You can write a message to two queues by invoking the Q program as follows. This mode will use distribution lists, but that can be overridden with the `-ai` option.

```
q -m QM1 -o Q1 -o Q2
```

4.3. Example 3. Write to a queue on a remote queue manager

You can address remote queues with Qmgr/Queue naming format. In this example, the '/' character is the separator between the two names.

```
q -m QM1 -o QM2/Q1
```

For more on this queue name format, see ***“7.1. Queue Name Format“*** on page 25. Take care with this syntax on z/OS in JCL jobs (see ***13.2 Use of the slash (/) character in JCL PARM strings*** on page 50).

4.4. Example 4. Read from a queue

You can destructively get messages from a queue by invoking the Q program as follows.

```
q -m QM1 -IQ1
```

To read the queue with a browse instead of a destructive get, use the lower case '`i`' instead.

```
q -m QM1 -iQ1
```

4.5. Example 5. Show Message Descriptor

You can format out the fields of the message descriptor of a message you have read from a queue by invoking the Q program as follows.

```
q -m QM1 -i Q1 -dd3
```

There are lots more formatting options explored later.

4.6. Example 7. Copy messages between queues

To copy messages from one queue to another, you do a browse (`-i`) on the first queue and then output (`-o`) the messages to the second queue.

```
q -m QM1 -i Q1 -o Q2
```

4.7. Example 6. Move messages between queues

To move messages from one queue to another, you do a destructive read (`-I`) on the first queue and then output (`-o`) the messages to the second queue. You might consider it sensible to do this in a transaction (`-p`) which in this example commits after every 10 messages.

```
q -m QM1 -I Q1 -o Q2 -p 10
```

4.8. Example 8. Subscribe to a topic

The Q program can also interact with topics. Invoke the Q program in the following way to subscribe to a topic.

```
q -m QM1 -Ss:TopicStr -w60
```

There are lots more subscription options explored later – see ***"8.2. Publishing"*** on page 26. Take care with this syntax on z/OS in JCL jobs (see ***13.2 Use of the slash (/) character in JCL PARM strings*** on page 50).

4.9. Example 9. Make a durable subscription

This example expands on the previous example, now making a durable subscription, which also therefore requires a subscription name.

```
q -m QM1 -Sds:TopicStr -Sn:MySubName -w60
```

4.10. Example 10. Publish on a topic

To publish to a topic directly (rather than via an alias queue definition) you can invoke the Q program as follows.

```
q -m QM1 -Ts:TopicStr
```

There are lots more publication options explored later – see ***"8.2. Publishing"*** on page 26.

Chapter 5. Parameters

There are a number of parameters that can be passed to Q to control the behaviour you need. These are detailed in this chapter after a brief introduction to the way the parameters work.

5.1. How parameter flags work

There are many, many different flags on the Q program and they take one of the following forms.

5.1.1. Top level flags

The main, and most commonly used parameters have their own individual flag after which the value can be specified, Some examples include specifying the queue manager name to connect to (using the **-m** flag) and specifying a queue name to operate on (using the **-o** flag for output in this example).

```
q -m QM1 -o Q1
```

5.1.2. Second level flags

Due to the large number of parameters into the Q program, many are grouped together under a single top level flag, with second level flags for each individual parameter. The section of “Message Attribute” flags under **-a** are a good example of these, but there are many others.

You might for example request that the Q program creates a persistent (**-ap**) report (**-at**) message, using the asynchronous put option (**-af**). You could do this by repeatedly providing the **-a** flag:-

```
q -m QM1 -o Q1 -ap -at -af
```

but it is much simpler, and involves less typing, to provide them all at once on a single instance of the **-a** flag:-

```
q -m QM1 -o Q1 -aptf
```

5.1.3. Second level flags with values

When a second level flag provides a value, the flag is followed by a colon (:) character and then the value. There are a number of example of these, especially in the Publish/Subscribe options. These values can be combined with other non-value second level flags as follows. This example creates (**-Sc**) a fixed user subscription (**-Sf**) with the topic string (**-Ss:**) of *MyTopicString*.

```
q -m QM1 -Scfs:MyTopicString
```

but if two second level flags with values are needed, the top level flag must be repeated, as follows, where this example adds that the subscription is durable (**-Sd**) and thus needs a subcription name (**-Sn:**).

```
q -m QM1 -Scdfs:MyTopicString -Sn:MySubName
```

The non-value second level flags can go on either instance of the main flag, and do not need to be repeated on both.

5.2. Parameters Flags

Parameters are passed to the program as flags on the command line.

The following parameters are available:

Flag	Meaning
-a	Sets message attributes **p** Forces persistence **n** Forces non-persistence **q** Uses persistence as Queue Definition **d** Uses a message type of Datagram **R** Uses a message type of Request **r** Uses a message type of Reply **t** Uses a message type of Report **f** Use the Asynchronous Put option **F** Use the Synchronous Put option **a** Use the Read Ahead Get option **A** Use the No Read Ahead Get option **c** Complete message **s** Allow segmentation **C** Use the close quiesce option **D** Use the close delete option **i** Use individual queues instead of distribution lists if multiple queues are specified for output. Must be specified before **-o** flags **o** When using distribution lists, use Offsets rather than Pointer records **x** No # message processing. See ***"9.1. Q message operation syntax"*** on page 30 for more details on the '#' character. **X** Use Input exclusive when opening a queue for get **z** Zero out the Message Id **y:*expiry*** Set the message expiry time (in tenths of a second) **j:*format*** The format name to be used on the MQPUT **P:*priority*** Sets the priority of put messages **u:*Userid*** Use Alternate user Id with this user identifier
-A	Set the value of context fields. This should be used in combination with **-C** and the appropriate option. **a:*string*** Accounting Token **i:*string*** Application Identity data **o:*string*** Application Origin data **u:*Userid*** User Identifier
-C	Context options **a** Pass all context **i** Pass identity context **A** Set all context **I** Set identity context **n** No context
-c	**< CCSID > [: X 'Encoding']** Causes the MQGET call from the input queue to specify **MQGMO_CONVERT** with the CCSID and Encoding values specified. The encoding value should be specified in hex. For example **-c850:111**

-d	Controls what detail level is output to stdout. **h** Message is printed in hex **N** Don't print the message **d/D** Print the MQMD after/before the MQI calls **o/O** Print the MQOD after/before the MQI calls **p/P** Print the MQPMO after/before the MQI calls **g/G** Print the MQGMO after/before the MQI calls **s/S** Print the MQSD after/before the MQI calls **r/R** Print the MQSRO after/before the MQI calls **a** Print out all properties **n** Don't print out properties **H** Force properties into MQRFH2 format **B** Don't display PCF group boundaries **c** Ignore CR/LF **C** Ignore conversion errors. See ***“9.5. Data Conversion“*** on page 36 for details. **e** Entity substitution. See ***“9.2. Entity Substitution“*** on page 32 for details. **E:n** Detail level for the End of Structure marker. **x** Use XML Shortform **X** No XML Auto detect **k** Print hex fields with the characters represented by the hex field below the field. **L** Don't format XML leaf nodes on a single line **l** Print message length **f** Message content is printed formatted **F:type** Message content is printed formatted in one the following types:- **AUTO | CSV | FIX | JSON | TEXT | XML | EDIFACT** Or using a named User Format structure. See “***Chapter 10. Formatting Messages for display***” on page 45 for more details on these. **t** Print message offsets as a decimal number **T** Print message offsets as a hexadecimal number **u** Print characters in raw (unchecked). Doesn't check whether characters are printable. See ***“9.5.2 Conversion for Display“*** on page 36 for more details. U Strip (or Unchain) standard headers from message. See **Chapter 9.9 “Stripping Headers“** on page 40 for details. **y** Display message summary (should be used with browse) **z** Display put date and time in ISO 8601 format in UTC timezone **Z** Display put date and time in ISO 8601 format in local timezone **1** Low level of detail **2** Medium level of detail **3** High level of detail
-e	This parameter echoes the message to the reply queue. It causes the program to MQPUT any messages got from the input queue to the reply queue.

-E	This parameter echoes the message to the reply queue. It causes the program to MQPUT any messages got from the input queue to the reply queue, and fill in the Reply Qmgr.
-f **-f[=]**	**`<input file name>`** Each line of the file will be put to the output queue as a different message. On z/OS only, if '=' is specified then the program will use the full record length for each message. Take care with naming HFS file names on z/OS in JCL jobs (see ***13.2 Use of the slash (/) character in JCL PARM strings*** on page 50).
-F **-F[+]**	**`<input file name>`** when combined with **`-o`** **`<output file name>`** when combined with **`-i`** or **`-I`** The entire file will be put to the output queue as a single message, or a message will be written to the output file. Use -X to put an entire file in hex as a single message. On z/OS only, if '+' is specified, then the program will retain the attributes of an existing output dataset.

-g	Use a specific message id, correlation id or group id. **Note:** If you use a character id and do not specify all 24 characters, the remainder will be blank padded. If you use a hex id and do not specify all 24 bytes, the remainder will be null padded. **`m:`*`MsgId`*** Get by specified message id of 24 characters **`c:`*`CorrelId`*** Get by correlation id of 24 characters **`g:`*`GroupId`*** Get by group id of 24 characters **`xm:`*`MsgId`*** Get by specified message id in hex **`xc:`*`CorrelId`*** Get by correlation id in hex **`xg:`*`GroupId`*** Get by group id in hex **`pm:`*`MsgId`*** Put with specified message id of 24 characters **`pc:`*`CorrelId`*** Put with correlation id of 24 characters **`pg:`*`GroupId`*** Put with group id of 24 characters **`pxm:`*`MsgId`*** Put with specified message id in hex **`pxc:`*`CorrelId`*** Put with correlation id in hex **`pxg:`*`GroupId`*** Put with group id in hex **`pC`** Use MQCI New session For example, here is a hex correlation id. **-gxc:414D5120514D31202020202020202020D924F35B22AA0302**
-h	**`<string filter>`** Only pass messages through the pipe that contain the specified filter string. If the string filter is multiple words, please put the value in quotes so that the shell or command interpreter treats them as a single parameter.
-H	**`<SQL92 Message Selector>`** For example **`-H "Value > 100"`** For more information, please see ***"9.6. Selection by SQL92 Selector"*** on page 37.
-i	**`<input queue name>`** Specifies the input queue to browse. For more on the queue name format, see ***"7.1. Queue Name Format"*** on page 25.
-I	**`<input queue name>`** Specifies the input queue to consume messages from. For more on the queue name format, see ***"7.1. Queue Name Format"*** on page 25.
-k	Use browse lock

-l	**`<library name>`** Specifies the name of the IBM MQ library to run against. This parameter controls whether the program runs as a local application or as a client. **`mqm`** Local Application **`mqic`** Client Application If not specified the program will try to use the Queue Manager interface. If that is not present it will try to use the MQ Client interface. This parameter does not apply when the program is running on z/OS. You can of course connect to a z/OS queue manager using this parameter when the program is running on a platform other than z/OS.
-L	**`[ <start> ][ : <end > ]`** Specifies the start and end position of messages to process/display. For example: • `-L10` will display the first 10 messages returned • `-L4:` will display just the fourth message returned • `-L6:9` will display messages 6, 7, 8 and 9 returned Note that the fourth message returned may not actually be the fourth message on the queue if you are also using, say, the `-H` (SQL92) parameter.
-m	**`<Queue Manager name>`** Specifies the name of the Queue Manager to connect to if not the default. For example ***–m QM1*** This parameter can be specified twice in order to provide one queue manager for the input queue and another different queue manager for the output queue. If this parameter is specified twice then its relative order in relation to other parameters is important. It must precede the parameter for the queue it qualifies and additionally the first `-m` must be the first parameter provided to Q. For example **`-m QM1 -i INPUT.Q -m QM2 -o OUTPUT.Q`** [1] Similarly, parameters which qualify the queue manager connection, such as **`-l`** (to qualify it as a client connection) or **`-x`** (to use MQCONNX options), must be supplied after the **`-m`** for the queue manager in question and before the **`-m`** for the second queue manager. For example **`-m QM1 -l mqic -u myuserid -i INPUT.Q -m QM2 -o OUTPUT.Q`**
-M	**`<simple text message>`** Allows a simple text message to be put. The string supplied can also make use of the '#' format. See ***"9.1. Q message operation syntax"*** on page 30.

[1]If you use this example command with two local queue managers, you will receive an error from the MQ API of **`MQRC_ANOTHER_Q_MGR_CONNECTED`**. To work round this problem, add **`-xb`** to your command to qualify one or both of the queue manager connections as shared.

-n

A comma separated list of the report option to be added to the message when messages are put to the output queue

`ca`	Confirm on arrival
`cad`	Confirm on arrival with data
`cafd`	Confirm on arrival with full data
`cd`	Confirm on delivery
`cdd`	Confirm on delivery with data
`cdfd`	Confirm on delivery with full data
`e`	Exception
`ed`	Exception with data
`efd`	Exception with full data
`x`	Expiration
`xd`	Expiration with data
`xfd`	Expiration with full data
`pan`	Positive action notification (PAN)
`nan`	Negative action notification (NAN)
`newm`	New Message Id
`passc`	Pass Correl Id
`passm`	Pass Message Id
`copym`	Copy Message Id
`disc`	Discard Message
`passd`	Pass discard and expiry
`act`	Activity Reports

By preceding the option with a '!' character[2] then the given option will be removed rather than added. So, for example, the following command:

```
q -iQ1 -oQ1 -n!ca
```

will remove the 'Confirm on Arrival' report option from any messages transferred.
In addition a special value of '*' may be used to remove all Report Options. For example the following command:

```
q -iQ1 -oQ1 -n!*
```

will transfer messages and remove **all** Report options.
Additions and Removals can be done in a single flag. For example:

```
q -iQ1 -oQ1 -n!*,x
```

will remove any existing Report options and add the Expiration report option to all messages transferred.

[2]Note that on some systems, such as Linux, it may be necessary to enclose the ! parameter in quotes.

`-o`	**`<output queue name>`** For more on the queue name format, see ***"7.1. Queue Name Format"*** on page 25. Multiple **`-o`** (or **`-O`**) flags can be provided which will cause a distribution list to be used except on z/OS where they are not available (unless **`-ai`** is specified prior to the **`-o`** flags in which case, each queue will be opened separately).
`-O`	**`<output queue name>`** As for **`-o`**, but using the "Bind on Open" option.
`-p`	**`<Interval[/Wait time]>`** **`Interval`** Number of messages after which an MQCMIT is taken. If a negative number is specified, then MQGMO_SYNCPOINT_IF_PERSISTENT is used. **`Wait time`** Length of time in milliseconds to wait before committing a partial transaction. If not specified message operations are not under syncpoint.
`-q`	Quiet mode - doesn't write messages to screen
`-r[+]`	**`<Reply queue name>`** The name of the Reply Queue to be placed in the message descriptor when MQPUTing to the output queue. If '+' is specified then the program will read the reply from the reply queue before attempting the next put. To see the reply message, add the **`-s`** flag to your command.
`-s`	Force output to stdout. This is used to force the messages to be output to stdout at times when they wouldn't normally be displayed, for example:- ● to monitor messages if transferring messages from one queue to another ● reading reply messages when using `-r+`

-S	Subscribe to a topic	
	o:*TopicObject*	Subscribe to a Topic Object
	s:*TopicString*	Subscribe to a Topic String
	n:*SubName*	Subscription Name
	u:*SubUserData*	Subscription User Data
	l:*SubLevel*	Subscription Level
	c	Create a subscription (default flag if none specified)
	r	Resume a subscription
	a	Alter a subscription
	d	A durable subscription
	v	Subscription can be used by Any User
	f	Subscription can only be used by a Fixed User
	g	A group subscription
	N	Use the New Publications Only option
	R	Use the Publications on Request option
	C	Wildcards use the Character scheme
	T	Wildcards use the Topic scheme
	D	Delete a durable subscription when finished
	For more on Publish/Subscribe, see ***“Chapter 8. Publish/Subscribe“*** on page 26.	
-t	Print timings for API calls. See ***“Chapter 12. Time Reporting“*** on page 49 for more details.	
-T	Publish to a topic	
	o:*TopicObject*	Publish to a Topic Object
	s:*TopicString*	Publish to a Topic String
	r	Use the retain publication option
	p	Suppress the reply To fields being filled in
	n	Use the Not Own Subs option
	w	Warn if no matching subscriptions
	For more on Publish/Subscribe, see ***“Chapter 8. Publish/Subscribe“*** on page 26.	
-u	**`<Userid>`**	
	The userid that should be used for the connection. If a password is not provided, using the **`-U`** parameter, then Q will prompt for the password and what you type in will be echoed to the screen as asterisks.	
-U	**`<Password>`**	
	The password that goes with the userid, **`-u`**, parameter.	

-v	Verbose Level Controls what information about what the program is doing is printed. **L** Prints out licence information including the remaining licence period. **m** Prints out machine information. Useful for ensuring the correct licence fields are requested **p** Causes a pause before each API **P** Print out command line parameters **q** Quiet mode **Q** Very quiet mode **E** Write structure titles and time to stderr **u** Report user format file name in use. **>=1** Information about each message is printed **>=2** Each API call is printed
-V	Sets the return value from the Q program. Default behaviour is to return MQRC value. On some platforms the return value from a program is limited to a number smaller than a 2000+ MQRC value. In those cases the value returned will be the least significant byte of the reason code. This can clearly be confusing! Use one these flags if you have that restriction. z Forces a zero return code c Return MQCC mapped to 0,4,8 (standard representation for OK, Warning, Error on some platforms)
-w	**<Wait interval in seconds>** Number of seconds to wait for a message if reading from an input queue.
-W	**<Wait interval in milliseconds>** Number of seconds to wait before each MQGET call. Can be used to represent 'think' time if you are trying to emulate a server application.

-x	Use MQCONNX – some of these subflags will cause a menu prompt to appear where you will select further options. f Use FASTPATH bindings s Use STANDARD bindings i Use ISOLATED bindings b Use shared handles (blocking) n Use shared handles (non-blocking) c Specify channel (for use with client connection) – submenu, see ***“6.3. Explicitly provide all channel connection details“*** on page 23. q No fail if quiescing option used N No MQDISC on end t Specify connection tag (submenu) v Register event handler (client mode only). This allows you to see reconnection events and multi-cast events that are reported to your connection handle.
-X	**`<input file name>`** The entire file will be put to the output queue in hex as a single message. Use -F if the file just contains the message characters directly.
-y	Message Header and Footer This parameter allows you to specify a sequence of characters which should be written before and after a message is written to the output. The allows for easier parsing by downstream programs. For more information please see **“9.4 Headers and Footers“** described on page 35.
-Z	**`<hours different from UTC>`** The TimeZone to display message Put Date/Time in rather than UTC. Example **`-Z -3.5`**
-1	Use MQPUT1 rather than MQPUT
-$	**`<separator character>`** Sets separator character in queue names Without this option you can use a queue value of QM1/Q1,QM1#Q1 or QM1,Q1 However, '/' can be used in a queue name so it may be useful to restrict the processing of this character. For more on the queue name format, see ***“7.1. Queue Name Format“*** on page 25.

-#	Set MQ structure versions m MQMD o MQOD g MQGMO p MQPMO x MQCNO c Use Current versions for all For example.**-#m1p2g3o4** would ask for Version 1 MQMD Version 2 MQPMO Version 3 MQGMO Version 4 MQOD For example **-#c** uses current versions Any group can be specified independently.
-*	Repeat execution without program terminating
-=[t]	**<message size>** Set Max message size. Optionally adding the letter 't' will allow truncation. See ***“9.8. Message Truncation“*** on page 40.
-?	Search help text. For more details, see ***“5.3. Getting help from the command“*** on page21.

5.3. Getting help from the command

There are quite a lot of parameters to remember and it would be nice if Q could prompt your memory. Well, it can do that in a number of ways.

- **All Parameter overview**

  ```
  q
  ```

  By entering the command on it's own Q will output the parameters it accepts

- **All parameters**

  ```
  q -?
  ```

  This command will output all the parameters and information about the options for each parameter.

- **Parameter information**

  ```
  q -d?
  ```

  This command will give information about the -d parameter

- **Parameter search**

  ```
  q -?persist
  ```

  This command will output any parameter usage text containing the string 'persist'

- **Parameter search prompt**

  ```
  q -??
  ```

  This command will display the following prompt, where you can then type in a string to be found within the usage text.

  ```
  Help - Enter topic:
  ```

Chapter 6. Connecting to the Queue Manager

You have two connection methods available to you with Q. You can either connect directly to a local queue manager, or connect using a client connection. The default behaviour is to connect directly to the queue manager. If you have a default queue manager you can omit the **-m** switch which provides the queue manager name.

6.1. Connecting as a client

To connect to the Queue Manager via a client the normal IBM MQ client configuration rules apply. The program can use either the MQSERVER environment variable or the Client Channel Definition Table (CCDT). The same program can connect both as a local application and as a client. The **-l** flag controls which IBM MQ library is loaded at runtime, the default being to connect locally.

To run as a local application connecting directly to a local Queue Manager you would use a command such as:-

```
q -m QM1 -iQ1
```

To run as a client you would use a command such as:-

```
q -m QM1 -iQ1 -l mqic
```

6.2. Passing in a Userid & Password on the connection

Q supports the CONNAUTH feature provided by IBM MQ in V8.0. There are two ways of using this feature.

You can either pass both the userid and password on the command line such as the following:

```
q -m QM1 -iQ1 -u myuserid -U mypassword
```

Or you can just provide the userid. In which case Q will prompt for the password

```
q -m QM1 -iQ1 -u myuserid
```

Will result in the following

```
IBM MQ Queue Utility
  Version V9.1.0 Build date:Nov  6 2018
   (C) Copyright MQGem Software 2018,2018. (http://www.mqgem.com)

Please enter password for QM(QM1) User(myuserid) >****
```

Any characters which are typed at the password prompt will be echoed as asterisk (*) characters.

On z/OS, Q will only prompt for the password when running in z/OS UNIX, but not when running in TSO or in Batch from JCL.

6.3. Explicitly provide all channel connection details

You can use the **-xc** flag to enter a prompt menu to set all the channel details yourself instead of using a CCDT.

```
q -m QM1 -o Q1 -xc
```

This will prompt you as follows (blue text below shows what the user typed in). When the prompt says **(NULL for ...)**, you can press enter on an empty line to accept that default value. In the example below, in order to show all prompts, we have answered 'y' to both exits and SSL. If you chose 'n' for either you will not see the related prompts.

```
Enter Channel Name
MQGEM.SVRCONN
Enter Channel Type (NULL for CLNTCONN)

Enter Transport Type (NULL for TCP)

Enter connection name (NULL for localhost)
localhost(1701)
Do you want exits ?
y
Enter Send exit

Enter Receive exit

Enter Security exit
mqccred(ChlExit)
Enter Security user data
NOCHECKS
Do you want SSL ?
y
Enter Cipher Spec (NULL for TLS_RSA_WITH_AES_256_CBC_SHA256)

Enter Key Repository (NULL for C:\mqm\key)
c:\MQGem\key
Enter CryptoHardware

Enter Authinfo Conn Name

Connecting ...
```

6.4. Transactions

By default Q will not use transactions. If you want your messages operations done inside a transaction, use the **-p** flag to indicate the commit interval, that is, the number of messages after which Q will issue an MQCMIT call. This will also ensure that Q uses syncpoint for the get and put operations.

Clearly you can not specify a value larger than the maximum uncommitted messages value for the Queue Manager and you need to ensure that the Queue Manager log is large enough for your transaction.

To use the **`MQGMO_SYNCPOINT_IF_PERSISTENT`** option on your MQGETs, specify the **`-p`** flag with a negative number. The absolute value of the number you provide will be used as the commit interval as above, but **`MQGMO_SYNCPOINT_IF_PERSISTENT`** will be use on the MQGETs instead of **`MQGMO_SYNCPOINT`**, and only persistent messages will be counted towards the commit interval. If you are moving/copying messages from one queue to another, and using **`MQGMO_SYNCPOINT_IF_PERSISTENT`** then any non-persistent messages will be put to the target queue using **`MQPMO_NO_SYNCPOINT`**.

6.4.1. Transaction completion

If you use a commit interval greater than one, there is the possibility that Q will run out of messages with a partially filled transaction. These partial transactions can be committed in one of the following two ways.

If the Q program ends (its wait interval times out and there are no more messages for example) then Q will issue an MQCMIT to complete any partial transaction that is left.

If you are running the Q program as a long running task moving messages from one queue to another, it may not end for hours! In this case, use the second numeric parameter on **`-p`** to indicate how many milliseconds to wait before committing any partial transactions.

```
q -m QM1 -I Q1 -o Q2 -p10/500
```

6.4.2. Transactions across Queue Managers

Q is capable of copying and moving messages between Queue Managers. It will use transactions to try and ensure consistency. However, since it does not run under a transaction coordinator, it is not a global transaction. The transaction used for the source Queue Manager is separate to the one used for the target Queue Manager and they are committed separately. The target Queue Manager is always committed before the source Queue Manager. This means that in the unlikely event of a failure the risk is that messages are duplicated rather than they are lost.

If you are concerned about the possibility of duplicate messages then you can do the operation in two stages. First, you copy/move the messages to a holding queue on the target Queue Manager. If that is successful then you can copy/move the messages to the intended queue. Since this now an operation involving only a single Queue Manager you know that there will be no duplication.

Chapter 7. Using a queue

Most of the interactions you will use the Q program for will involve using a queue. You name your queue(s) using the **-i -I** and **-o** flags for browse, input and output respectively. In all those flags you can use the same naming format described below.

7.1. Queue Name Format

A number of the parameters on the Q program allow the specification of a queue name. For example the output queue. The name is passed to the MQOPEN verb and can either consist of one or two parts. A one part name is assumed to be the name of the queue (the queue manager name will be set to blank). With a two part name the first part is taken to be the queue manager name, and the second part is the name of the queue. By default. a two part name must use one of the following characters as a separator.

Default separator characters
/
\
#
,

For example the following examples tells Q to take input from the keyboard and put the messages to different queues.

```
Q -m QM1 -oQ1
```

```
Q -m QM1 -oQM2/Q1
```

```
Q -m QM1 -oQM3#Q1
```

If your queue names (or queue manager names) contain one the '/' separator character, then the Q program would misinterpret that character as implying a separation between queue manager name and queue name. You can avoid this problem by choosing a single separator character that is not used by your queue and queue manager names, and tell the Q program to use only that separator character instead, by using the -**$** flag.

```
q -m QM1 -$# -oQM3#Q1/PROD
```

Chapter 8. Publish/Subscribe

The Q program can also publish messages to topics, and subscribe to and receive messages published to topics.

This chapter covers in more detail how to do that with the various flags provided.

8.1. Referencing your topic

To publish or subscribe to a topic, you can either

- use the topic string directly
- use a topic object name
- use the anchor point method where you have both a topic object and and an additional piece of a topic string that is appended to the topic string found in the given topic object.

The Q program allows you to use any of these three mechanisms with the topic object flag on both publish (`-To:`) and subscribe (`-So:`) and the topic string flag on both publish (`-Ts:`) and subscribe (`-Ss:`). You can combine them as necessary as the examples below demonstrate.

Publishing, using the topic string directly:

```
q -m QM1 -Ts:MQGem/Product/Q
```

Publishing, using a topic object name:

```
q -m QM1 -To:MQGEM
```

Publishing, using the anchor point method:

```
q -m QM1 -To:MQGEM -Ts:Product/Q
```

8.2. Publishing

Publishing to a topic is very similar to putting a message to a queue. You reference the object you wish to use and either use the `-M` flag to put a simple message; supply a file name containing some message text; or input message data from the keyboard at the message input prompt. The special message processing in the Q program where you prefix your text with the '#' character also applies to publishing messages. See ***"9.1. Q message operation syntax"*** on page 30 for more details.

The above examples showing how to reference a topic object will all take you to the message input prompt.

In addition to the topic object and topic string parameters, there are also a few Put Message Options that apply to topics, and these can be selected as sub-flags in the **`-T`** flag.

Flag	Meaning
o:*TopicObject*	Publish to a Topic Object
s:*TopicString*	Publish to a Topic String
r	Use the retain publication option
p	Suppress the reply To fields being filled in
n	Use the Not Own Subs option
w	Warn if no matching subscriptions

8.3. Subscribing

Subscribing to a topic is a little more complex than publishing, as you might expect. You reference the topic you wish to subscribe to in the same way as described above, but then there are quite a few other options as well. Perhaps some examples are the best way to illustrate it.

8.3.1. Creating a non-durable managed subscription

This is perhaps the simplest example and needs very few flags. By not providing any queue name, your subscription instead uses the managed option, but you can still wait (**`-w60`**) on the messages from the managed subscription queue that the queue manager creates for you.

```
q -m QM1 -Scs:MQGem/Product/Q -w60
```

8.3.2. Creating a durable subscription

When a subscription is durable (**`-Sd`**) then a subscription name is also required (**`-Sn:`**) so now you have two string values to supply and the **`-S`** flag needs to be used twice. This example is still a managed subscription however, no mention of a queue name here either. This example also uses a wild card in the topic string (the '**`#`**' character on the end of the topic string).

```
q -m QM1 -Scds:MQGem/Product/# -Sn:QSub -w60
```

8.3.3. Resuming, and deleting a previously created durable subscription

A durable subscription stays around after your application disconnects, unless you ask for it to be removed. This example illustrates resuming a previously created durable subscription (**`-Sr`**) and removing it (**`-SD`**) when the Q program ends. You could of course combine the delete subscription flag (**`-SD`**) with the earlier example.

```
q -m QM1 -SrdDs:MQGem/Product/# -Sn:QSub -w60
```

8.3.4. Using a non-managed subscription

So far the examples have all been managed subscriptions, but you can also use your own queue as the subscription queue as shown in this example. This example uses a non-durable subscription, and no subscription name, but you can of course combine those parts of the earlier examples with using your own queue too.

```
q -m QM2 -I Q1 -Scs:MQGem/Product/# -w60
```

8.3.5. Other subscription options

In addition to the flags demonstrated above, there are a number of others that allow you to set certain options on your subscribe call. Most of these are in the `-S` flag, but there are also some others that also apply to subscribing as shown in the table below.

Flag	Sub-Flag	Meaning
`-S`	`o:`***`TopicObject`***	Subscribe to a Topic Object
`-S`	`s:`***`TopicString`***	Subscribe to a Topic String
`-S`	`n:`***`SubName`***	Subscription Name
`-S`	`u:`***`SubUserData`***	Subscription User Data
`-S`	`l:`***`SubLevel`***	Subscription Level
`-S`	`c`	Create a subscription (default flag if none specified)
`-S`	`r`	Resume a subscription
`-S`	`a`	Alter a subscription
`-S`	`d`	A durable subscription
`-S`	`v`	Subscription can be used by “Any User”
`-S`	`f`	Subscription is Fixed to the User that creates it
`-S`	`g`	A group subscription
`-S`	`N`	Use the “New Publications Only” option
`-S`	`R`	Use the “Publications on Request” option
`-S`	`C`	Wildcards use the Character scheme ('*' and '?')
`-S`	`T`	Wildcards use the Topic scheme ('#' and '+')
`-S`	`D`	Delete a durable subscription when finished
`-a`	`P:Priority`	Set the priority of messages sent to the subscriber
`-a`	`y:Expiry`	Set the expiry of the subscription
`-a`	`a`	Use read ahead
`-a`	`A`	Don't use read ahead
`-a`	`u:Userid`	User Alternate User Authority
`-g`	`c:CorrelID`	Set the Correlation ID for the subscription
`-x`	`q`	Turn off Fail if Quiescing
`-C`	`I`	Set identity context on the subscription. Combine with `-Ai` and `-Aa`.
`-A`	`i`	Set Pub Application Identity Data
`-A`	`a`	Set Put Accounting Token
`-H`		Use a selection string. See ***“9.6. Selection by SQL92 Selector“*** on page 37.

Chapter 9. Dealing with Messages

The Q program has many options to manipulate your messages. Fuller explanations of a number of those options are included in this chapter.

9.1. Q message operation syntax

When typing messages in from the keyboard a message is generated each time the return key is pressed. However, a message starting with a '#' character is interpreted to allow multiple messages to be generated. The syntax is displayed by entering '?' a the message input prompt.

This '#' character prefix is also processed when providing simple message text using the **`-M`** flag.

Perhaps some examples are the best way to illustrate it.

9.1.1. Put Multiple messages

Type the following command in at the message input prompt to put 250 1000 byte messages to your queue.

```
#250/1000 My message text
```

Alternatively, you can have the same effect if you use the above string as the contents of a simple message using the **`-M`** flag as follows.

```
q -m QM1 -o Q1 -M"#250/1000 My message text"
```

9.1.2. Random sized messages

To put multiple messages that are not all the same size, you can use the following command to request 250 random sized messages up to 2000 bytes.

```
#250/-2000 My message text
```

or from the command line as

```
q -m QM1 -o Q1 -M"#250/-2000 My message text"
```

9.1.3. Omit # instruction

If you've tried out the above two examples, you'll see that each message contains the # instruction string at the beginning of the message. To omit this from being sent with the message, use the '**`!`**' character, and ensure that you have a space after the # instruction string and some follow on text for it to put into the message instead.

```
#!250/-50 My message text
```

or from the command line as

```
q -m QM1 -o Q1 -M"#!250/-50 My message text"
```

9.1.4. Random content

For long messages, you'll find that you don't type enough characters into the command to fill the message size, so you might like to just have some randomly generated data. Here's how. In order for the '`!`' to take effect, you must have a space on the end of this command.

```
#!c250/-100 
```

This uses one half of the check-summing mechanism, but since you have requested the # instruction string to be omitted from the message, the checking side of the process won't happen. To learn more about checksumming, see ***"9.10. Check-summing messages"*** on page 43.

9.1.5. Timing of messages

You can trickle feed messages onto your queue, instead of dumping them all at once, by adding a delay. This is measured in milliseconds. In the following example we put 250 500 byte messages, one every half a second.

```
#250/500/500 Timed message
```

You can make this delay into a random one by using a negative number instead. In this example we put 250 150 byte messages with a random delay up to 1 second (1000 milliseconds).

```
#250/150/-1000 Randomly timed message
```

9.1.6. Commit Interval

If you are using the # instruction string to put large numbers of persistent messages, you might like to put them in transactions to make more efficient use of your queue manager.

This example puts 2500 random data messages of sizes up to 5000 bytes, with no delay, and commits after every 500 messages. (N.B. There is a space on the end of this command as explained above).

```
#!c2500/-5000/0/500 
```

The complete syntax is as follows:-

```
#[!][x][X][i][I][c[g[Count]]][p]Reps[/Size[/Delay[/Commits]]]
```

Flag	Meaning
`!`	Don't include the # instruction string as part of the message. The value up to the first space is discarded, so ensure you have a space between the # instruction string and your message text. This cannot be combined with checksum processing since the # instruction string is required at get time to check the checksum value. However, you can combine '`!`' with '`c`' if you simply want random data content in your messages.
`x`	Message is full range of hex characters
`X`	Message is indexed full range of hex characters. See ***"9.5.3 Codepoints example"*** on page 36 for an example of this in use.
`i`	Index the messages. See ***"9.11. Indexing messages"*** on page 43.
`I`	
`c`	Asks for the message to be checksummed. The message will contain random characters, unless the 'g' option is also used. For more information on checksums see ***"9.10. Check-summing messages"*** on page 43.
`g[Count]`	In checksum processing, don't randomise the characters of the message, use sets (of optional *count* specified) of letters, increasing alphabetically from A to Z, then a to z. This flag must be combined with the 'c' flag.
`p`	Add message properties to subsequent message. For more details, see ***"9.7. Using Message Properties"*** on page 39. This option cannot be used with the **`-M`** simple message flag.
`Reps`	Number of messages to produce
`Size`	Size of each message. This can be specified as a negative number in which case the program will randomly choose the size of the message up to the value specified.
`Delay`	Delay in milliseconds between each MQPUT. This can be specified as a negative number in which case the program will randomly choose the delay up to the value specified.
`Commits`	Commit after this many messages

When you are are in the message input prompt, typing **`#end`** will end the program. In most shells, <Ctrl>+<Z> will also end it.

9.2. Entity Substitution

Sometimes messages can contain sequences of characters which represent a single character. For example, the sequence **`&gt;`** is sometimes used to represent the 'greater than' (>) symbol. Having these character sequences can be easier for a target program to process but can make the message difficult to read. Use the flag **`-dfe`** to request that the Q program displays these sequences as the single character they represent.

The character sequences that are recognised by the Q program are:-

Sequence	Character
`&gt;`	`>`
`&lt;`	`<`
`&nbsp;`	`(space)`
`&amp;`	`&`
`&quot;`	`“`
`&apos;`	`'`

9.3. Context Options

There are two sets of context information in the Message Descriptor (MQMD), the identity context fields and the origin context fields. These are described in the IBM Knowledge Center: Message Context. The default action of Q is to use default context. However, you can choose to set the context information in the MQMD with the use of various options. This requires the user ID under which the utility is running to have appropriate authority to set some or all of the context fields.

The context fields can be manipulated with the **`-C`** flag and the **`-A`** flag. The table below details the options that can be used. The “pass” options are only applicable when moving messages from one queue to another queue.

`-C` flag	Meaning
`A`	**Set All Context** When you choose this `-C` flag, you can specify one or more of the following -A flags to set the value of the given MQMD field. **-A flags** | `i:` | Application Identity data `o:` | Application Origin data `a:` | Accounting Token `u:` | User Id On it's own (without any -A flags), this flag can be used to mimic pass all context when you are moving messages between two queue managers (and thus the “pass all context” option is not applicable), as the MQMD from the get call is used on the put call.

`-C` flag	Meaning
`I`	**Set Identity Context** When you choose this `-C` flag, you can specify one or more of the following -A flags to set the value of the given MQMD field. **-A flags** `i:` Application Identity data `a:` Accounting Token `u:` User Id On it's own (without any -A flags), this flag can be used to mimic pass identity context when you are moving messages between two queue managers (and thus the “pass identity context” option is not applicable), as the MQMD from the get call is used on the put call. This flag is also applicable to a subscription. When you choose this `-C` flag, you can specify one or more of the following -A flags to set the value of the given MQSD field. **-A flags** `i:` Pub Application Identity data `a:` Pub Accounting Token
`a`	**Pass All Context** Pass all the context fields in the MQMD from the messages on the input queue to the messages on the output queue. Both queues must be accessed from the same connection
`i`	**Pass Identity Context** Pass only the identity context fields in the MQMD from the messages on the input queue to the messages on the output queue. Both queues must be accessed from the same connection
`n`	**No Context** There will be no information in the Context fields in the MQMD of messages on the output queue.

9.4. Headers and Footers

If Q is used to output the contents of a queue to a file then, by default, each message is written on a new line. This is great for some applications and for being readable by a human however for some other applications it would be nice to be able to add some special characters around the messages to make them easier to parse. This is where the **-y** parameter comes in.

So, suppose we have three messages on our queue, containing the text “One”,”Two” and “Three”. If we just output them to a file using the following command:

```
q -iQ1 > Q1.out
```

we would get the following contents in Q1.out

```
One
Two
Three
```

You can see how Q adds a new line after each message. However, we could ask Q to put our own headers and footers on the message with the following command

```
q -iQ1 -yh:HEADER -yf:FOOTER > Q1.out
```

this would yield the following file

```
HEADEROneFOOTERHEADERTwoFOOTERHEADERThreeFOOTER
```

Of course in reality you would use a rather more unique sequence of characters, perhaps an XML tag or similar. Note that it is not necessary to have *both* header and footer specified. If you just specify one or the other then that's fine.

There is also a shorthand for when you need both a header and a footer and they are of equal length and that is the **-yd:** option. When used the first half of the parameter is the header, the second half is the footer. For example we could say the following:

```
q -iQ1 -yd:[[]] > Q1.out
```

and this would result in the following file:

```
[[One]][[Two]][[Three]]
```

Clearly an output of this sort would make it very much easier for a downstream program to parse the message boundaries. Just make sure to use a header and/or footer that is unique.

9.5. Data Conversion

9.5.1. IBM MQ Data Conversion

If the messages on the queue being read need to be converted, the **`-c`** flag should be used to cause the MQGET call from the input queue to specify **`MQGMO_CONVERT`** with the CCSID and Encoding values specified. The encoding value should be specified in hex. Use the following syntax on the command line.

```
q -m QM1 -i Q1 -c 850:X'222'
```

If all that is required is to use the local code page and native encoding then simply use the following syntax on the command line.

```
q -m QM1 -i Q1 -c 0
```

If data conversion on the MQGET fails, you will receive an appropriate return code from IBM MQ which the Q program will report back to you. Here's an example.

```
q -m QM1 -iQ1 -c1234
MQGET on object 'Q1' returned 2115 Target CCSID error.
```

9.5.2. Conversion for Display

If you choose to browse or get your messages without using **`MQGMO_CONVERT`** as described above, then the Q program will attempt to display the output in an appropriate code page by doing data conversion into the code page for your system after the MQGET has finished. On Windows the Q program will discover the ANSI Code page for the O/S, on z/OS it will use IBM-1047 and on other platforms it will assume UTF-8. If you need to over-ride the code page for your console for output from the Q program, set the **`Q_CCSID`** environment variable.

If the value being used for the console output causes problems, you may see the following message to show that the Q program was unable to do the appropriate data conversion for display. To ignore these errors use the **`-dC`** flag.

```
* WARNING * Display to codepage 1234 is not supported, defaulting to 1208.
```

On Windows and Linux, the Q program uses wide-characters (**`WCHAR_T`**) to ensure that, no matter what the code page, it will be able to correctly display the data in your messages. On AIX and z/OS, you will be more dependant on your locale settings as these determine which characters are considered “printable”. The default locale in the UNIX shell, is the C locale, also known as the POSIX locale. This locale has a considerably smaller set of characters which are considered printable than most other locales. You can bypass the check for “printable” characters, by using the **`-du`** flag. However, bear in mind that this means truly non-printable characters will be written to your screen and may cause some weird behaviours.

Instead, if you set up your locale appropriately, you will see a larger set of characters are considered printable, and any variant characters, for example characters with accent marks, will be able to be viewed.

9.5.3. Codepoints example

The example that follows is a quick way to check whether your locale is set up to display all the characters you might need.

The Q program can very quickly produce a message with every code point from 0 – 255 (0xFF). You can do this by invoking Q as follows:-

```
q -m MQG1 -o Q1 -M#X
```

This will write a message with three bytes for each code point. The first two bytes are characters giving the index of the code point and the third byte is the code point itself. Here's a snippet to illustrate.

```
81a 82b 83c 84d 85e 86f 87g 88h 89i 8A. 8B. 8C. 8D. 8E. 8F. 90.
```

Here it shows that the hexadecimal code point value of 0x81 is the letter 'a' – this is an EBCDIC example. The dots shown for 0x8A-0x90 indicate that the locale considers these codepoints non-printable.

Changing the locale as follows:-

```
export LC_ALL=En_US.IBM-037
```

and browsing the same message again shows

```
81a 82b 83c 84d 85e 86f 87g 88h 89i 8A« 8B» 8Cð 8Dý 8Eþ 8F± 90°
```

The letters a-i are visible in both since those codepoints are invariant (the don't change), but we now have a few other characters that are considered to be printable.

9.6. Selection by SQL92 Selector

IBM MQ (V7 and later), provides a flexible selector string which can be used to control, to a fairly fine detail, the messages which are returned from an MQGET call or sent to a subscriber. The provided selection string is used on the MQOPEN call or MQSUB call respectively.

It is beyond the scope of this document to explain in detail the syntax and full capabilities of the SQL92 selector but we'll give a few examples. Essentially the SQL92 selector allows you to construct a boolean expression using mathematical operators which allows you to select against message properties as well as the standard message descriptor. It is therefore possible to write selectors of significant complexity which makes them very powerful.

For a full description of the SQL92 syntax please see IBM Knowledge Center: Message selector syntax.

This can, at first glance, seem a little daunting so let's look at a few examples.

- **Display only persistent messages**

  ```
  q -m QM1 -i Q1 -H "Root.MQMD.Persistence = 1"
  ```

- **Display only reply messages**

  ```
  q -m QM1 -i Q1 -H "Root.MQMD.MsgType = 2"
  ```

- **Display messages that were put by a particular user[3]**

  ```
  q -m QM1 -i Q1 -H "Root.MQMD.UserIdentifier = 'Paul        '"
  ```

  Notice that you need to pad the string field with blanks if you make an exact comparison.

[3]Here we have an example of a quoted string inside a quoted string. Note that the exact syntax may vary between Operating Systems and Shells. If you have problems consult your shell user guide.

- **Display messages that contain a user property value**

```
q -m QM1 -i Q1 -H "CustomerNumber = 123"
```

- **display messages with a particular Correlation ID value (Windows)**

```
q -m QM1 -i Q1 -H "Root.MQMD.CorrelId=""0x414D51204D51383034202020202020202020D973DF572000C274"""
```

display messages with a particular Correlation ID value (Linux - bash)

```
q -m QM1 -i Q1 -H "Root.MQMD.CorrelId=\"0x414D51204D51383034202020202020202020D973DF572000C274\""
```

The main problem in this example is the different ways to input the double-quote inside the command string in different shells.

Of course, you could also do this using the following command and avoid using the SQL92 selector altogether, but it does make a useful illustration of using multiple sets of double quotes in a command.

```
q -m QM1 -i Q1 -gxc:414D51204D51383034202020202020202020D973DF572000C274
```

9.7. Using Message Properties

If your messages contain message properties, the Q program can also deal with those.

9.7.1. Displaying message properties

You have a few choices about what to do with the message properties in your messages when you are displaying them with the Q program.

You can display them using the following command.

```
q -m QM1 -iQ1 -da
```

Which will show them displayed like this

```
  Telephone   :800123123
  ProductName :'Q'
A message from Q with some message properties
```

You can force them to be generated as an MQRFH2 header (**-dH**), and the display the formatted MQRFH2 header (**-df**), using the following command.

```
q -m QM1 -iQ1 -dHf
```

Which will show them displayed like this

```
[  165 bytes] Message Content
<usr>
  <Telephone dt='i4'>
    800123123
  </Telephone>
  <ProductName>
    Q
  </ProductName>
</usr>
A message from Q with some message properties
```

Or you can choose not to display them at all, using the following command.

```
q -m Qm1 -iQ1 -dn
```

9.7.2. Setting message properties

If you want to add message properties to a message, there is a small sub-menu that is shown if you type **#p** at the message input prompt.

This will then display the following sub-menu.

```
i. Add Integer
c. Clear Properties
d. Delete Properties
D. Delete Property handle
l. List Properties
s. Add String
q. Quit
```

Typing either '**i**' or '**s**' on this menu (to add an integer or string property respectively) will prompt you for a name and then a value for that property.

```
i
Enter name
Telephone
Enter value
800123123
```

Typing '**l**' will list the properties currently on your message property handle:-

```
l
 Telephone   :800123123
 ProductName :'Q'
```

You can use '**d**' to delete properties by name (prompted), or '**c**' to clear them all completely (without further prompting).

When you type '**q**' to quit this sub-menu, the properties on your message handle will be used for the messages you subsequently type into the message input prompt.

9.8. Message Truncation

When messages are long and you don't want to see the whole message, you can reduce the size of the message buffer that the Q program uses on its MQGET. You have two choices in this situation, you can accept the truncation of your messages, useful for browsing, or you can fail if the message is truncated, so no message data is lost.

9.8.1. Accept Truncation

Use a Q command like the following – this one is browsing the queue.

```
q -m QM1 -iQ1 -=t100
```

The letter '**t**' after the **-=** flag is what indicates you are willing to truncate.

9.8.2. Fail on Truncation

Use a Q command like the following – this one is destructively getting from the queue and writing to a file.

```
q -m QM1 -IQ1 -=100 > output.txt
```

Any messages that are greater than the 100 bytes specified will stop the program and report:-

```
Message (500 bytes) larger than Max size (100 bytes)
```

No further messages will be processed.

9.9. Stripping Headers

Messages can have a variety of structures and it is fairly common to have one or more standard headers before the 'meat' of the message itself. Examples of these might be the Transmission Queue header (added when a message is put to a transmission queue) or perhaps the Dead Letter Queue header (added when the message is put to the Dead Letter Queue).These headers usually provide additional information about the message and are very useful if the message is to be sent to a program which understands these headers. However, there are times when you might want to re-direct these messages to an alternative target and in those cases you might well want to strip these headers from the front of the message.

Q allows you to strip the headers[4] by giving it's format name on the command line using the **`-dU`** parameter. For example the following command:

```
q -m QM1 -I SYSTEM.DEAD.LETTER.QUEUE -o Q1 -dU:MQDEAD
```

This command will move all the messages on the Dead Letter Queue to a queue called Q1 and remove any Dead Letter Queue headers it finds.

Q allows the following headers to be stripped from the message.

Format Name	Header Type
MQCICS	CICS Header
MQDEAD	Dead Letter Queue Header
MQHDIST	Distribution List Header
MQHREF	Reference Message Header
MQHRF	Rules and Formatting Header
MQHRF2	Rules and Formatting Header (Version 2)
MQHWIH	Work Information Header
MQIMS	IMS Header
MQXMIT	Transmission Queue Header

The **`-dU`** parameter is actually a little more flexible than just simply providing the name. Firstly you are allowed to use wildcards. So, for example the command:

```
q -m QM1 -I Q1 -o Q2 -dU:MQHRF*
```

will move messages from Q1 to Q2 and remove any Rules and Formatting Headers it finds, whether version or version 2. We could even extend this with the following:

```
q -m QM1 -I Q1 -o Q2 -dU:*
```

which asks Q to remove **any** of the known headers from the front of message leaving only the base message.

[4]Sometimes known as 'Unchaining' headers since the standard headers can be considered a chain where each header references the format, codepage and encoding of the next.

The second extension to the `-dU` parameter is that you can be more specific about the order headers should be found by specifying a comma separated list of names. The processing of the comma separated list is as follows:

```
for each name in the list
{
  while(topmost header matches current name)
  {
    strip header from message-data
  }
  go to next name in list
}
```

It follows therefore that the order of headers in the message is important. The list of names is not just a list of all headers which will be stripped but the *sequence* they must be found. Consider the following command:

```
q -m QM1 -I Q1 -o Q2 -dU:MQDEAD,MQXMIT
```

The effect of this command would depend on which order these headers were in the messages. If the message contained a Dead Letter Queue header followed by a Transmission Queue Header then both headers would be removed. However, if the message contained a Transmission Queue Header followed by a Dead Letter Queue Header then only the Transmission Queue Header is removed.

9.10. Check-summing messages

When using the # message processing, you can ask for the messages to be checksummed.

```
q -m QM1 -o Q1

Connecting ...connected to 'QM1'.
>#c1/2000
[Chk.-4783794]
```

The above example command entered at the message input prompt, will ask the Q program to generate one message of 2000 bytes in length, containing random characters as the message content.

The program will randomly set the characters of the message. It will then calculate a checksum by adding up the numerical values of each byte in the message multiplied by its offset in the message and print out that checksum value to the screen. Upon receipt of a checksummed message, the Q program will regenerate the checksum using the same calculation and print out the result. As the user, you can then compare the two checksum values to determine if they match.

This is useful to ensure that messages are being received complete and without corruption.

In order to see the checksum value at get time, you must be doing a simple get. Any of the following display options will turn off the display of the checksum on get; **`-df -dh -dN`**.

Just issue a command like this:-

```
q -m QM1 -i Q1
```

and you'll see output like this:-

```
(2000 bytes) [Chk.-4783794] #c1/2000)#¥äßlÍ«RÉI±±.ÚÙ.ª.<ç.>Ö$^...ÀGÌ
```

N.B. This output is truncated to 40 bytes on the screen, since the random characters may not even be printable. You can over-ride this truncation by adding **`-ax`** to your command.

9.11. Indexing messages

Sometimes it can be useful to have the text of the message describing a series of messages, and showing where you currently are in the series. For example when demonstrating an aspect of messaging, such as the round-robin nature of clustering, or assured delivery of messages.

This feature allows you to quickly create messages such as the following:-

```
Msg 1 of 5: Source Q1/QM1
Msg 2 of 5: Source Q1/QM1
Msg 3 of 5: Source Q1/QM1
Msg 4 of 5: Source Q1/QM1
Msg 5 of 5: Source Q1/QM1
```

By typing the following in at the message input prompt.

```
#i5
```

Alternatively if you use capital '`I`', you will get an incrementing number, spelled out, as your message text.

```
#I5
```

would give you

```
One
Two
Three
Four
Five
```

Or you can build your own message using either `#i` or `#I` using the following inserts

Insert	Meaning
`%i`	Number
`%I`	Number in words, e.g. One, Two, Three etc
`%l`	Total number of messages, allowing “one of 3” for example.
`%L`	As `%l`, but with number in words, allowing “one of three” for example
`%q`	Queue name
`%m`	Queue manager name

Here are some examples.

```
#!i5 Msgs for Queue:%q on Qmgr: %m - %I of %L
```

gives

```
Msgs for Queue:Q1 on Queue Manager: MQG1 - One of Five
Msgs for Queue:Q1 on Queue Manager: MQG1 - Two of Five
Msgs for Queue:Q1 on Queue Manager: MQG1 - Three of Five
Msgs for Queue:Q1 on Queue Manager: MQG1 - Four of Five
Msgs for Queue:Q1 on Queue Manager: MQG1 - Five of Five
```

The '`!`' character instructs the Q program to omit the # instruction string from the message text when it puts the message.

9.12. Output from the Q program

The Q program prints out data to both **`stdout`** and **`stderr`**.

Message contents, formatted headers and so on, are sent to **`stdout`**.

Other information, such as licensing details, connection status, and other errors are sent to **`stderr`**.

So, for example, you can send the message contents information to an output file without having all the other error and information message included in that file.

```
q -m QM1 -i Q1 -df3 > Q1.txt
```

Chapter 10. Formatting Messages for display

The Q program recognises and can format out a number of message formats. All the built-in MQ headers are automatically recognised and can be formatted out using the **-df** flag. These built-in MQ headers are recognised because the MQMD Format field has the appropriate value in it.

Additionally, various industry standard string formats are supported, and also your own message formats can be displayed if the shape of the message is described in an MQGem User Format file.

10.1. Display Message Formats

In addition to the built in MQ headers, there are a number of string based message formats (those which just have the MQMD Format field set to MQSTR) which the Q program can also format out using the **-dF:** flag, by naming the format in question.

Format	Description
AUTO	The Q program will detect which of the following types the data is, and choose an appropriate formatted display accordingly. This value is used by default if you omit the **-dF:** flag and just specify **-df**.
CSV	The data consists of many fields of data separated by commas. Each comma separated value is displayed on a new line. This format is also useful for displaying the Open Financial Standard (OFS) messages.
FIX	This formatter is designed for FIX (Financial Information eXchange) messages. Essentially these are message where the delimiter between fields is the SOH (0x01) character.
JSON	The data should be treated as JSON data and formatted according to the bracket nesting.
TEXT	The data is displayed as just plain text characters.
XML	Q will parse the data into the various XML tags.
EDIFACT	The data should be interpreted according to the EDIFACT standard.

For example, use a command like the following to force a message to be formatted as JSON:-

```
q -m QM1 -iQ1 -dF:JSON
```

which will display a message like this:-

```
[  144 bytes] Message Content
{
  "firstName": "Paul",
  "lastName": "Clarke",
  "address": {
    "streetAddress": "1600 Pennsylvania Avenue",
    "state": "Washington DC",
    "country": "U.S.A."
  }
}
```

10.2. User Format Messages

In addition to the above, the Q program can format out your own message formats by reading an MQGem User Format file that describes the shape of the message. The description of MQGem User Formats is detailed in a separate user guide. If you downloaded it into the same directory as this user guide, then this link will open it.

10.2.1. Locating the User Format file

The Q program will look for your user format file in the following location by default.

Platform	Location
Windows and Linux	Same directory as the Q program – file called `mqgem.fmt`
AIX and z/OS UNIX	Current directory – file called `mqgem.fmt`
z/OS	DD:MQGEMFMT

Alternatively, you can set environment variable **`MQGEM_USERFORMAT_FILE`** to point to the full path and file name of your user format file. If you are uncertain whether Q is correctly picking up your user format file location, use the **`-vu`** parameter to have it report the file it is using.

This environment variable can also be used to ensure no attempt is made to open a format file, by setting it to the value "**`none`**".

10.2.2. Associating a User Format structure with your message

The MQGem User Format file allows you to associate a user format structure you have defined in the file with a queue name or a Message Descriptor format, or both. This is all defined in the user format file itself. You can learn more about this in the separate "MQGem User Format User Guide".

In addition, you can force Q to format out your message using a specifically named structure, over-riding any queue name or format name association that might be in the file, or just supplying one because the association is not in the file, by using the **`-dF:`*`<struct-name>`*** parameter.

For example, with the following format file contents:

```
struct EMPLOYEE; Employee Record
{
  char            Name[30]: summary; Full Name
  char            Title[3];
  int             Age;
}
struct PERSON; Person Record
{
  char            Name[30]: summary; Person Name
  char            Job[3];
  int             Age;
}
queue ADD.USER.RECORD
{
  struct EMPLOYEE;
}
```

the messages on queue **`ADD.USER.RECORD`** would always be formatted using the structure **`EMPLOYEE`**. With the following command parameters, you could instruct the Q program to instead use the structure **`PERSON`** on this invocation.

```
q -m QM1 -i ADD.USER.RECORD -dF:PERSON
```

Chapter 11. Triggering

The Q program can be used as a triggered application. In order to demonstrate this, the example below will use the Q program to read a message from an input queue (**`MQGEM.INPUTQ`**), and then echo (**`-e`**) that message back to the ReplyToQ recorded in the message.

11.1. 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)
```

11.2. Describe the triggered application

The triggered application will need an input queue, a reply queue, and a process object showing where the Q program executable lives. Make sure your **`DEFINE PROCESS`** details the location for your machine. The parameters to the Q program can be provided in the queue's **`TRIGDATA`** field and/or in the process object's **`USERDATA`** field. You can use one, other or both of these fields depending on your requirements.

The Q program command that we want to run when our input queue is triggered is the following:-

```
q -m QM1 -I MQGEM.INPUTQ -e
```

However, we don't need to pass in the queue manager name flag (**`-m`**) or the input queue flag (**`-I`**), as those pieces of information are already in the trigger message, so all we need to pass is the echo flag (**`-e`**).

```
DEFINE PROCESS(MQGEM.Q.PROC) DESCR('Q Program') APPLICID('c:\MQGem\Q\q.exe')
```

```
DEFINE QLOCAL(MQGEM.REPLYQ) DESCR('ReplyQ used for Q Triggered demo')
```

```
DEFINE QLOCAL(MQGEM.INPUTQ) DESCR('Triggered InputQ for Q Triggered demo')
INITQ(MQGEM.INITQ) PROCESS(MQGEM.Q.PROC) TRIGGER TRIGDATA('-e') TRIGTYPE(FIRST)
```

11.3. Trigger the Q program

Now that you have the triggered application configured, put a few messages to the triggered queue, with their ReplyToQ filled in (**`-r`**), and wait for the replies to come back(**`-r+`**). Normally those replies would not be written to **`stdout`**, so we force them to **`stdout`** (**`-s`**) and we add a small get-wait (**`-w`**) as well. Here is the command:-

```
q -m QM1 -o MQGEM.INPUTQ -M”#!I5 Hello %I” -r+MQGEM.REPLYQ -s -w30
```

and here is the output you'll see, showing the messages echoed back to you

```
Connecting ...connected to 'QM1'.
Hello One
Hello Two
Hello Three
Hello Four
Hello Five
```

Chapter 12. Time Reporting

You can use the Q program to report some primitive timings to test the performance of your system. To do this, just use the -t flag with the rest of your Q command to see output such as the following.

12.1. Measuring puts

This command puts 10000 messages to the queue Q1 and prints out the timings. For more details on the # message format see ***"9.1. Q message operation syntax"*** on page 30

```
q -m QM1 -o Q1 -M"#10000" -t
```

This will print out something like the following.

```
Connecting ...connected to 'QM1'.
10000 Iterations in 1.52s, Average = 152.06us or 6576.4 per second
```

12.2. Measuring gets

This command gets messages from a queue, and every 5000 messages (or environment variable `Q_PERF_LIMIT` messages if you have over-ridden the default), it prints out the timings. This example command also uses the quiet flag (`-q`) as printing to the screen can be an expensive business.

```
q -m QM1 -I Q1 -q -t
```

This will print out something like the following.

```
Connecting ...connected to 'QM1'.
5000 Iterations in 960.54ms, Average = 192.11us or 5205.4 per second
5000 Iterations in 728.51ms, Average = 145.70us or 6863.3 per second
No more messages.
```

This option allows you to easily compare the performance of various options in the MQI, such as:-

- Message size
- Message persistence
- Client connection (e.g. TLS / compression / exits)
- Connection type (e.g. Fastpath / Standard / Isolated)
- Read-ahead
- … and many others

Chapter 13. z/OS Specifics

This user manual applies to the Q program when running on z/OS as well. In some z/OS environments there are some quirks that it is helpful to know about and this chapter pulls those together.

13.1. z/OS File name format

Specifying the file name on z/OS can be done in a number of different ways. Examples of the most common are shown here.

To provide the name of a dataset, whether sequential file or partitioned dataset, directly, use the following file name format (note that this can be fully qualified as in the first example, or will have your TSO user ID (e.g. GEMUSER) pre-pended to the name in the second example, depending on whether you use the single quotes or not):

```
q -m MQG1 -o Q1 -f"//'GEMUSER.MY.FILE'"
q -m MQG1 -o Q1 -f"//MY.FILE"
```

You can provide the name of a member of a partitioned dataset, using the following file name format (bearing in mind the use of single quotes mentioned above):

```
q -m MQG1 -o Q1 -f"//'GEMUSER.MY.PDS(MEMBER)'"
```

Take care with this single-quoted syntax in JCL jobs (see ***13.2 Use of the slash (/) character in JCL PARM strings*** below).

You can also use z/OS UNIX files, as shown in this example.

```
q -m QM1 -o Q1 -f "/u/gemuser/messages.txt"
```

You can also use DD cards to identify the data set you wish to use. For example, if you have a DD card similar to one of the following examples:

```
//INFILE DD DSN=GEMUSER.MY.FILE,DISP=SHR
//INFILE DD DSN=GEMUSER.MY.PDS(MEMBER),DISP=SHR
//INFILE DD PATH='/u/gemuser/messages.txt',PATHOPTS=(ORDONLY)
```

you can then use the following:

```
q -m MQG1 -o Q1 -fDD:INFILE
```

For more details see “Performing OS I/O Operations” in “z/OS XL C/C++ Programming Guide”.

13.2. Use of the slash (/) character in JCL PARM strings

There are a number of places where you might use the slash (/) character when invoking the Q program. If you are running Q from a JCL job, then you need to remember that a slash (/) character in the JCL PARM string is treated as a separator between the runtime options and the program parameters and if, as is likely, you did not intend this meaning, then you need to put quotes around the argument containing the slash ('/'). Here are some examples.

When referencing a queue on a remote queue manager:

```
EXEC PGM=Q,PARM=('-m QM1 -o "QM2/Q1"')
```

When publishing or subscribing to a topic that contains multiple levels:

```
EXEC PGM=Q,PARM=('-m QM1 -S"s:Price/Fruit/#" -w60')
```

When referring to a z/OS UNIX file name:

```
EXEC PGM=Q,PARM=('-m QM1 -o Q1 -f "/u/gemuser/messages.txt"')
```

When referring to an MVS file name, you have to deal with both the slash ('/') character and the single quotes when using this in a JCL PARM string. You must double up the single quotes as shown here:

```
EXEC PGM=Q,PARM=('-m MQG1 -o Q1 -f"//''GEMUSER.MY.PDS(MEMBER)''"')
```

Chapter 14. Environment variables

There are a few environment variables that change the behaviour of the Q program.

Environment Variable	
`MQGEML`	Location of licence file (if not in same directory as executable). For more details see ***"2.2. Licence File Location"*** on page 5.
`MQACCESS_DEBUG`	Set this environment variable to any value to get additional helpful messages if you are having problems finding the MQ DLLs to connect to the queue manager.
`MQ_INSTALLATION_PATH`	This IBM MQ provided environment variable is used by the Q program on distributed platforms, to locate the MQ DLLs.
`MQIC_DLL_PATH`	Set this environment variable to over-ride MQ_INSTALLATION_PATH for the location of your client (mqic) DLL.
`MQM_DLL_PATH`	Set this environment variable to over-ride MQ_INSTALLATION_PATH for the location of your server (mqm) DLL.
`Q_SCREENHEIGHT`	The Q program will attempt to find the number of lines in your command window, to appropriately pause the help text if more than one screen full would be displayed. To over-ride what the O/S says, or to set a value in environments where the screen height cannot be determined, set this environment variable to a number.
`Q_PERF_LIMIT`	When using the `-t` flag to print out timings, the Q program will, by default, print out said timings after 5000 messages have been got from the queue. Set this environment variable to a number to over-ride the 5000.
`Q_CCSID`	If you need to over-ride the code page used for output from the Q program, set this environment variable. For more details see ***"9.5. Data Conversion"*** on page 36.
`MQGEM_USERFORMAT_FILE`	Location of your MQGem User Format file. For more details see "***10.2. User Format Messages***" on page 46.

Chapter 15. MQ Options

This sections acts as a reverse mapping from the various different MQ options, to the flags available in the Q program.

15.1. Connection Options

The following connection options are available through the Q program.

Connection Option		Q flag
Accounting options	MQCNO_ACCOUNTING_MQI_ENABLED	
	MQCNO_ACCOUNTING_MQI_DISABLED	
	MQCNO_ACCOUNTING_Q_ENABLED	
	MQCNO_ACCOUNTING_Q_DISABLED	
Binding options	MQCNO_STANDARD_BINDING	`-xs`
	MQCNO_FASTPATH_BINDING	`-xf`
	MQCNO_SHARED_BINDING	
	MQCNO_ISOLATED_BINDING	`-xi`
	MQCNO_CLIENT_BINDING	`Use -l mqic`
	MQCNO_LOCAL_BINDING	`Use -l mqm`
Connection-tag options	MQCNO_SERIALIZE_CONN_TAG_Q_MGR	`-xt`
	MQCNO_SERIALIZE_CONN_TAG_QSG	
	MQCNO_RESTRICT_CONN_TAG_Q_MGR	
	MQCNO_RESTRICT_CONN_TAG_QSG	
Handle-sharing options	MQCNO_HANDLE_SHARE_BLOCK	`-xb`
	MQCNO_HANDLE_SHARE_NO_BLOCK	`-xn`
Reconnection options	MQCNO_RECONNECT_AS_DEF	`Omit others`
	MQCNO_RECONNECT	`-xr`
	MQCNO_RECONNECT_DISABLED	`-xR`
	MQCNO_RECONNECT_Q_MGR	`-xm`
Conversation-sharing options	MQCNO_NO_CONV_SHARING	`-xS`
	MQCNO_ALL_CONVS_SHARE	
Channel definition options	MQCNO_CD_FOR_OUTPUT_ONLY	
	MQCNO_USE_CD_SELECTION	

15.2. Open Options

The following open options are available through the Q program.

Open Option		**Q flag**
Access options	MQOO_INPUT_AS_Q_DEF	
	MQOO_INPUT_SHARED	`-I`
	MQOO_INPUT_EXCLUSIVE	`-aX`
	MQOO_OUTPUT	`-o`
	MQOO_BROWSE	`-i`
	MQOO_CO_OP	
	MQOO_INQUIRE	
	MQOO_SET	
Binding options	MQOO_BIND_ON_OPEN	`-O`
	MQOO_BIND_NOT_FIXED	`-o`
	MQOO_BIND_ON_GROUP	
	MQOO_BIND_AS_Q_DEF	
Context options	MQOO_SAVE_ALL_CONTEXT	`Used as needed`
	MQOO_PASS_IDENTITY_CONTEXT	`-Ci`
	MQOO_PASS_ALL_CONTEXT	`-Ca`
	MQOO_SET_IDENTITY_CONTEXT	`-CI`
	MQOO_SET_ALL_CONTEXT	`-CA`
Read ahead options	MQOO_NO_READ_AHEAD	`-aA`
	MQOO_READ_AHEAD	`-aa`
	MQOO_READ_AHEAD_AS_Q_DEF	`Omit others`
Other options	MQOO_ALTERNATE_USER_AUTHORITY	`-au:`
	MQOO_FAIL_IF_QUIESCING	Always on. Remove with `-xq`
	MQOO_RESOLVE_LOCAL_Q	`-xl`
	MQOO_RESOLVE_LOCAL_TOPIC	
	MQOO_NO_MULTICAST	

15.3. Get Message Options

The following get messages options are available through the Q program.

Get Message Options		**Q Flag**
Wait options	MQGMO_WAIT	`-w`
	MQGMO_NO_WAIT	
	MQGMO_SET_SIGNAL	
	MQGMO_FAIL_IF_QUIESCING	Always on. Remove with `-xq`
Sync point options	MQGMO_SYNCPOINT	`Use -p`
	MQGMO_SYNCPOINT_IF_PERSISTENT	`Use -p (-ve)`
	MQGMO_NO_SYNCPOINT	`Omit -p`
	MQGMO_MARK_SKIP_BACKOUT	
Browse options	MQGMO_BROWSE_FIRST	`-i`
	MQGMO_BROWSE_NEXT	
	MQGMO_BROWSE_MSG_UNDER_CURSOR	
	MQGMO_MSG_UNDER_CURSOR	
	MQGMO_MARK_BROWSE_HANDLE	
	MQGMO_MARK_BROWSE_CO_OP	
	MQGMO_UNMARKED_BROWSE_MSG	
	MQGMO_UNMARK_BROWSE_CO_OP	
	MQGMO_UNMARK_BROWSE_HANDLE	
Lock options	MQGMO_LOCK	`-k`
	MQGMO_UNLOCK	
Message-data options	MQGMO_ACCEPT_TRUNCATED_MSG	`-=t`
	MQGMO_CONVERT	`-c`
Group and segment options	MQGMO_LOGICAL_ORDER	
	MQGMO_COMPLETE_MSG	`-ac`
	MQGMO_ALL_MSGS_AVAILABLE	
	MQGMO_ALL_SEGMENTS_AVAILABLE	
Property options	MQGMO_PROPERTIES_AS_Q_DEF	
	MQGMO_PROPERTIES_IN_HANDLE	`-da`
	MQGMO_NO_PROPERTIES	`-dn`
	MQGMO_PROPERTIES_FORCE_MQRFH2	`-dH`
	MQGMO_PROPERTIES_COMPATIBILITY	

15.4. Put Message Options

The following put messages options are available through the Q program.

Put Message Options		Q Flag
Scope option	MQPMO_SCOPE_QMGR	
Publishing options	MQPMO_SUPPRESS_REPLYTO	`-Tp`
	MQPMO_RETAIN	`-Tr`
	MQPMO_NOT_OWN_SUBS	`-Tn`
	MQPMO_WARN_IF_NO_SUBS_MATCHED	`-Tw`
Syncpoint options	MQPMO_SYNCPOINT	`Use -p`
	MQPMO_NO_SYNCPOINT	`Omit -p`
Message-identifier and correlation-identifier options	MQPMO_NEW_MSG_ID	`Use -az`
	MQPMO_NEW_CORREL_ID	
Group and segment options	MQPMO_LOGICAL_ORDER	
Context options	MQPMO_NO_CONTEXT	`-Cn`
	MQPMO_DEFAULT_CONTEXT	`Omit others`
	MQPMO_PASS_IDENTITY_CONTEXT	`-Ci`
	MQPMO_PASS_ALL_CONTEXT	`-Ca`
	MQPMO_SET_IDENTITY_CONTEXT	`-CI`
	MQPMO_SET_ALL_CONTEXT	`-CA`
Property options	MQPMO_MD_FOR_OUTPUT_ONLY	
Put response options	MQPMO_ASYNC_RESPONSE	`-af`
	MQPMO_SYNC_RESPONSE	`-aF`
	MQPMO_RESPONSE_AS_Q_DEF	
Other options	MQPMO_ALTERNATE_USER_AUTHORITY	`-au:`
	MQPMO_FAIL_IF_QUIESCING	Always on. **Remove with** `-xq`
	MQPMO_RESOLVE_LOCAL_Q	`-xl`

15.5. Subscription Options

The following subscription options are available through the Q program.

Subscription Options		**Q Flag**
Access or creation options	MQSO_CREATE	`-Sc`
	MQSO_RESUME	`-Sr`
	MQSO_ALTER	`-Sa`
Durability options	MQSO_DURABLE	`-Sd`
	MQSO_NON_DURABLE	`Omit -Sd`
Destination options	MQSO_MANAGED	`Omit -i & -I`
	MQSO_NO_MULTICAST	
Scope Option	MQSO_SCOPE_QMGR	
Registration options	MQSO_GROUP_SUB	`-Sg`
	MQSO_ANY_USERID	`-Sv`
	MQSO_FIXED_USERID	`-Sf`
Publication options	MQSO_NOT_OWN_PUBS	
	MQSO_NEW_PUBLICATIONS_ONLY	`-SN`
	MQSO_PUBLICATIONS_ON_REQUEST	`-SR`
Read ahead options	MQSO_READ_AHEAD_AS_Q_DEF	
	MQSO_NO_READ_AHEAD	`-aa`
	MQSO_READ_AHEAD	`-aA`
Wildcard options	MQSO_WILDCARD_CHAR	`-SC`
	MQSO_WILDCARD_TOPIC	`-ST`
Other options	MQSO_ALTERNATE_USER_AUTHORITY	`-au:`
	MQSO_SET_CORREL_ID	`-gc`
	MQSO_SET_IDENTITY_CONTEXT	`-CI`
	MQSO_FAIL_IF_QUIESCING	Always on. **Remove with** `-xq`

Chapter 16. Changes made in previous versions

16.1. Changes in V9.2.0

1. **Message Display Formatting**
   This version of Q now has a new formatting engine based on wide-characters. This means that Q has better display support for DBCS, MBCS and variant characters.

2. **New message summary display**
   Option `-dy` will tell Q to output just a one line summary of the messages on the queue.

3. **New formatted Put Date/Time option in ISO 8601 format**
   The new puttime is displayed based on the `-dz` and `-dZ` option which display the time in UTC or local time respectively.

4. **Message limiting**
   The message limit parameter `-L` has been enhanced to contain both an optional start and end limit. For example `-L3:` will display just the 3rd message. `-L5:7` will display messages 5,6 and 7. As before `-L10` will display the first 10 messages.

5. **XML formatting**
   By default XML leaf tags will be displayed on a single line. The previous method of displayed them on three lines can be obtained using option `-dL`

6. **PCF Groups**
   By default Q will now display the boundaries of PCF groups. This can be suppressed using the `-dB` option.

7. **Output of newline character in raw (non-formatted) mode**
   This version will output 'newline' characters as a newline rather than the previous '.'

8. **Command Level 924 Support**

9. **z/OS Support**
   Q is now available to run locally on z/OS. To enable this you require a z/OS specific licence. A distributed platform licence will not enable Q on z/OS to run.

16.2. Changes in V9.1.0

1. **IBM MQ Multi-version Support**
   Q will load the MQ libraries from the place identified by `setmqenv`.

2. **New help features**
   To make it easier to find the option you are looking for – see ***“5.3. Getting help from the command“*** on page 21.

3. **Message formatters added**
   Formatters for JSON, EDIFACT, CSV and FIX message formats have been added – see ***"10.1. Display Message Formats"*** on page 45.

4. **64-bit executable**
   The Q program is now 64-bit across all platforms.

5. **Minor flag enhancements**

   a) The special message format string which starts with the '#' character previously had a 40 character limit. This has been lifted. For more on this special format string, see ***"9.1. Q message operation syntax"*** on page 30.

   b) `MQRO_PASS_CORREL_ID` has been added to the confirm options on the `-n` flag as `-n[passc]`.

   c) You can use `-dt` to print out the offsets of a message

   d) The prompt menu used by `-xc` to set various client connection channel settings such as TLS and exits has been updated to only request valid exits for a client channel, and to use a more modern CipherSpec by default.

   e) The use of truncation on an MQGET has been made safer by requring the user explicitly select `MQGMO_ACCEPT_TRUNCATED_MESSAGE` when using the `-=` flag by using the optional '`t`' flag, to give `-=t<length>`.

   f) The Commit interval (-p) flag has been extended to also take an optional commit interval after which an incomplete transaction, that is one that has not reached the requested total number of messages, will be committed anyway.

   g) The subscribe call created with the `-S` flag can be additionally configured to use `MQSO_SET_CORREL_ID` by using the `-gc:CorrelId` flag.

6. **Flags renamed**
   Some minor flags have been renamed to improve usability and consistency – see ***"17.3. Migrating from the MA01 IBM Support Pac"*** on page 61. The majority of the flags are exactly the same as they were in the Support Pac.

Chapter 17. Migrating from previous versions

We always try to ensure that, as each version is shipped, all the features that were working in the previous versions remain intact. When installing a new version we always recommend you backup the previous Q program. 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.

17.1. Migrating from Version 9.4.0

The following changes may affect your operation of the program

- **-n parameter**
  This parameter allows you to specify the type of report options you want added to the message. Previous versions of Q allowed for a free format parameter whereas now the parameter is more prescriptive to allow for the negative flavours of each flag. It may be that a previous command using the -n flag is now rejected. The parameter must only contain valid parameters or '!', '*' or ',' characters.

17.2. Migrating from Version 9.1.0

The following changes may affect your operation of the program

- **New display engine**
  The engine for formatting and displaying the contents of messages has been changed to wide-character format. This means that Q has better support for DBCS and MBCS character sets.

- **Removal of -dV option**
  Q used to be able to convert the data from EBCDIC to ASCII just before display. This option allowed you to suppress this behaviour. However, now the data is always converted to wide-characters. As such this option no longer applies.

- **XML leaf nodes**
  The XML formatter used to display leaf nodes on three separate lines. However, leaf nodes will now be displayed in format <tag>Value</tag> on a single line. If you wish to go back to the three line format then you can do so with the **`-dL`** option.

- **PCF Groups**
  When formatting PCF groups Q will now show you where the start and end of the groups are. If you wish to suppress this behaviour you can use the **`-dB`** option.

17.3. Migrating from the MA01 IBM Support Pac

Changes to various flags

In this first version of Q as a supported product by MQGem Software, we have used this as the one opportunity to make some changes to flags to tidy them up into more appropriate groupings from those that were used in SupportPac MA01. This section details the changes in those flags, so if you have scripts that used the MA01 SupportPac and you are now moving to using the MQGem version, please check whether you are affected by any of these changes.

- **Browsing messages**
  The -b flag which was long deprecated, has been completely removed. Use -i instead.

- **Setting the priority of a message**
  `-P<priority>` becomes `-aP:<priority>`

- **Setting the format of a message**
  `-J<format>` becomes `-aJ:<format>`

- **Setting the expiry of a message**
  `-y<expiry>` becomes `-ay:<expiry>`

- **Zeroing out the message id**
  `-z` becomes `-az`

- **Setting the user identifier in the message**
  `-U<userid>` becomes `-Au:<userid>`

- **Using Alternate User ID**
  `-U+<userid>` becomes `-aU:<userid>`

- **Request message properties as MQRFH2 header**
  There were two options for this action, `-a2` and `-dF`. `-a2` has been removed and `-dF` has been changed to `-dH`.

- **Providing user id and password on connect**
  `-xu` has been replaced with `-u` and `-U` to match QLOAD

- **Accept Truncation is no longer the default**
  -= has been changed to require accept truncation to be explicitly requested (with the addition of the letter 't') since it is a hazardous option.

- **Second level flags which provide a value require a colon between the flag and the value**
  Therefore the following flags have changed:
  `-dw<width>` becomes **`-dw:<width>`**
  `-Ai<appl-identity-data>` becomes **`-Ai:<appl-identity-data>`**
  `-Ao<appl-origin-data>` becomes **`-Ao:<appl-origin-data>`**
  `-Aa<acct-token>` becomes **`-Aa:<acct-token>`**
  `-g[p][x][C][m|c|g]<identifier>` becomes **`-g[p][x][C][m|c|g]:<identifier>`**
  `-Sl<sub-level>` becomes **`-Sl:<sub-level>`**