Queue Load / Unload Utility for IBM MQ
User Guide

Version 9.1.1

10th January 2023

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”

Twenty-seventh Edition, January 2024

This edition applies to Version 9.1.1 of Queue Load / Unload Utility for IBM MQ and to all subsequent releases and modifications until otherwise indicated in new editions.

Notices

Table of Contents

Tables

History

SupportPac

Ever since I released my MA01 (Q Utility) SupportPac I have had periodic requests to explain how it can be used to unload, and subsequently reload, messages from a queue. The answer has always been that this is not what the Q Utility is for and that surely there must be another tool available. Well, after sufficient numbers of these requests I looked for such a tool myself and didn't really find anything which fitted the bill. What was needed was a very simple, some would say unsophisticated, program which unloaded a queue into a text file. The notion of a text file was important because a number of users wanted the ability to change the file once it had been created. I also find that text based files are more portable and so this seemed useful if we want to unload a queue, say on Windows, and then load the messages again on an AIX machine. The disadvantage of this approach is that the file is larger than it would be in binary mode. Storing data using the hex representation of the character rather than the character itself essentially uses twice as much space. However, in general I do not envisage people using this program to unload vast amounts of message data but a few test messages or a few rogue messages on the dead letter queue which are then changed and reloaded elsewhere.

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).

Many people have requested that I continue to support and enhance QLOAD and sent in a number of their requirements. I have therefore decided to add QLOAD to the list of MQGem products and to support it while ever people find it useful. Future support and development of the program will therefore be made by MQGem Software and all support questions should be directed to support@mqgem.com.

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

Chapter 1. Queue Load / Unload Utility for IBM MQ

Overview

The Queue Load / Unload Utility for IBM MQ (QLOAD for short) allows the user to copy or move the contents of a queue, its messages, to a file. This file can be saved away as required and used at some later point to reload the messages back onto the queue. Messages can be copied or removed from a queue based on various selection criteria such as position in the queue, content of messages or message properties or the age of the message.

The unload file has a specific format understood by the utility, but is human-readable, so that it can be updated in an editor before being reloaded. Care should be taken not to change the format when editing fields within it. The utility will only reload a file with the correct format.

Changes from previous version

The main changes from the previous version the utility are:

1. **Non-QLOAD file can contain properties blocks**
   Enhancing the feature added in V9.1.0, when QLOAD reads a file that was not created by QLOAD but contains a set of user created messages (marked by delimiters), a second set of delimiters can indicate property blocks that are added to the message as an MQRFH2 folder.
   For more information please see 'Chapter 8. Reading non-QLOAD files' on page 26.

2. **Fully-qualified queue names can use a user-specified separator character**
   To match our Q Utility, provision of a user-specified separator character with the `-$` flag is now available. Without it, the characters '#','\','/', and comma, are looked for in queue names and treated as the queue vs queue manager name separator.

3. **QLOAD will move messages to the backout queue specified on an input queue**
   If you have a backout threshold and a backout requeue name specified on your input queue and QLOAD comes across messages with a Backout Count greater than the threshold, it will move those messages to the backout-requeue queue. See more details in 'Backout re-queue Queue' on page 46.

4. **QLOAD will show an application name of “MQGem Software QLOAD” on IBM MQ status output**

5. **Summary display extended**
   Previously only summarising message ages up to 1 day, and messages sizes up 1MB, the summary display (`-ds`) has been extended to summarise message ages up to 1 year and messages sizes up to 50MB. See more details in 'Display Summary' on page 52.

6. **Count the number of messages in an input source**
   Count the number of messages that match your filtering criteria, from an input queue or file.

7. **Update Put date and time**
   When putting messages to an output queue, copy all context fields, but update the put date and time to reflect the current time.

8. **QLOAD is available natively on z/OS**

9. **Allow generic files to be used with non-QLOAD files**
   So, for example, a command such as:
   ```
   qload -f c:\data\msg%n.txt -o Q1 -n1 -Cd
   ```

   would load the files msg1.txt, msg2.txt.....etc and load it to the queue Q1.

10. **New parameter -A allowing for overrides of certain message descriptor attributes**
    Please see 'Parameters Flags' on page 30 for more information.

11. **Summary display includes queue size counters**
    Summarising the depth, data amount, and queue file size in a table at the end of the summary output. See more details in 'Display Summary' on page 52.

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 z/OS Installation Instructions below for more details.

Linux, Unix and Windows

Once unzipped you should have the **qload** 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,

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 QLOAD would not run on an older version of Linux. I do, wherever possible, try to build QLOAD on older versions of the Operating System however it may not always be possible to go back far enough. If you have problems running QLOAD, particularly if it complains about GLIBC version, then please contact support@mqgem.com and we'll see what we can do.

z/OS Installation Instructions

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

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

```
receive inds(QLOAD.SEQ)
```

When prompted for a filename, reply

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

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

z/OS UNIX Installation Instructions

If you wish to run QLOAD 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(QLOAD)' '/u/gemuser/bin/qload' BIN
```

Chapter 2. Licensing

To access all the features (beyond any trial period) of QLOAD you will require a licence file. The licence file also entitles the user to email support. If you would like to try out QLOAD for free then a 1-month trial licence can be obtained by sending a note 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 QLOAD get more dedicated support**
  Resources can be targeted towards customers who have made a financial contribution to the development of QLOAD.

- **Purchasing decision is simpler**
  The QLOAD licence covers a period of time not a release. Any licence, will enable the user to run any version of QLOAD. 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 to collect a large group of features together to 'justify' a new release of QLOAD. 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	QLOAD is only supported on one machine using one userid.
Ruby	machine	QLOAD is supported by any number of users using it on the same machine.
Sapphire	userid	QLOAD is supported on any number of machines using the same userid.
Diamond	location	QLOAD 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	QLOAD 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.

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 QLOAD program with the following command on the machine you wish to run QLOAD on.

```
qload -Oi
```

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

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 QLOAD 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.

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.

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.

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

QLOAD can be useful for a wide variety of tasks. At it's core it is a pipeline for messages, moving or copying messages from a source to a target. The source and target can either be a queue (or set of queues) or a file (or set of files). During the transfer the messages can be filtered so that only some of the messages are transferred.

Uses

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

- **To unload messages from a queue to a file**
  The file can, if you wish, be edited to change the message and MQMD content before being re-loaded.
- **To re-load message from a file back to a queue**
  Messages can be reloaded at variable speed. You can even reload the messages at the same relative speed that they were put to the original queue.
- **To unload all queues, or a set of queues, from a Queue Manager to a set of files.**
  For example to take a back-up of your entire Queue Manager messages or to migrate from one Queue Manager instance to another.
- **To re-load a set of files to a Queue Manager**
  To restore a back-up or complete a Queue Manager migration
- **To copy or move messages between queues**
  These messages can be filtered in many different ways so only certain messages are copied or moved. See below for the types of filtering that can be applied.
- **To backup or remove messages from queue**
  Again, these messages can be filtered – e.g. removing all messages older than a certain age
- **To copy a single queue to a set of target queues**
- **To consolidate input from several queues to a single queue**
- **To load a set of user created messages to a queue**

Filtering

As messages are transferred between the source and target you can apply filtering based on various characteristics of the messages. For example, filtering can be based on:

- message position
- a string search
- message put timestamp
- message age
- message size
- message priority
- message id, correlation id or group id
- an SQL92 Selector

One of the best ways to learn how to use the program is to see a set of sample commands so please refer to “Chapter 4. Examples” on page 8 for examples of how to run the program. Full information about the program parameters can be found in “Chapter 9. Parameters“ on page 30.

Chapter 4. Examples

QLOAD can be useful for a number of tasks. As you will see, as you learn more about the features of QLOAD, the examples below are just a few of the things you can do with QLOAD. 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 9: Parameters” on page 30.

Example 1. Unload a Queue to a File

To save the messages that are on a queue, to a file, perhaps for archiving purposes and the possibility of later reload back onto a queue; use the following options on the command line.

```
qload -m QM1 -i Q1 -f c:\myfile
```

This takes a copy of the messages from the queue and saves them in the file specified. The format of this file is described in “Chapter 10: File Format” on page 59.

Example 2. Unload a Queue to a series of files

You can unload a queue to series of files by using an ‘insert’ character in the file name.

In this mode each message is written to a new file.

```
qload -m QM1 -i Q1 -f c:\myfile%n
```

This command will unload the queue to files, myfile1, myfile2, myfile3 etc.

You can control the name of the file in a number of ways, by adding date, time, or message number inserts to the file name. For a complete list of insert characters please refer to “File Insert Characters” on page 43.

By default each message will go to a different file when using the %n insert. However, by using the -L parameter the criteria for which a new file is created, can be change. For example, with the following options, a number of messages are written to each new file. The file limit provided causes a new file to be created once the current output file reaches 500 kilobytes in size.

```
qload -m QM1 -i Q1 -f c:\myfile%n -Ls500k
```

Example 3. Load a Queue from a File

To reload a queue with the messages you saved in “Example 1. Unload a Queue to a File”, use the following options on the command line. Note that the file passed to QLOAD must be a recognised format. The recognised formats are listed in “Recognised file formats” on page 60

```
qload -m QM1 -o Q1 -f c:\myfile
```

Example 4. Load a Queue from a series of files

You can load a queue from a series of files by using an 'insert' character in the file name.

```
qload -m QM1 -o Q1 -f c:\myfile%n
```

This command will load the queue files files, myfile1, myfile2, myfile3 etc. For a complete list of insert characters please refer to "File Insert Characters" on page 43.

Example 5. Copy the messages from one Queue to another Queue

The file parameter in "Example 1. Unload a Queue to a File" could be replaced with another queue name instead, allowing the messages from one queue to be copied to another queue, using the following options on the command line.

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

Example 6. Move messages from multiple Queues to another Queue

You can specify the `-i` and `-I` parameters as many times as you wish if you want to copy/move messages from multiple queues. In this case we are moving messages from three queues to a single target queue.

```
qload -m QM1 -I Q1 -I Q2 -I Q3 -o Q4
```

Example 7. Copy the first 100 messages from one Queue to another Queue

The file parameter in "Example 1. Unload a Queue to a File" could be replaced with another queue name instead, allowing the messages from one queue to be copied to another queue, using the following options on the command line.

```
qload -m QM1 -i Q1 -o Q2 -r#100
```

This example could be extended further to copy the messages to a queue on another queue manager, by using the following options on the command line.

```
qload -m QM1 -i Q1 -m QM2 -o Q2 -r#100
```

Please see "Transactions across Queue Managers" on page 47 for a discussion on the transactional implications of copying/moving messages between Queue Managers.

Example 8. Move the messages from one Queue to another Queue

A variation on "Example 3. Load a Queue from a File" would be to move the messages instead of copying them. This illustrates the distinction between using `-i` (lower case) which only browses a queue, and `-I` (upper case) which destructively gets from a queue. Use the following options on the command line.

```
qload -m QM1 -I Q1 -o Q2
```

This example could be extended further to copy the messages to a queue on another queue manager, by using the following options on the command line.

```
qload -m QM1 -I Q1 -m QM2 -o Q2
```

Please see "Transactions across Queue Managers" on page 47 for a discussion on the transactional implications of copying/moving messages between Queue Managers.

Example 9. Move messages older than one day from one Queue to another Queue

This example shows the use of age selection. Messages can be selected which are older than, younger than or within a range of ages. Use the following options on the command line.

```
qload -m QM1 -I Q1 -o Q2 -T1440
```

Example 10. Work with the file of messages

Having unloaded the message from your queue, as in "Example 1. Unload a Queue to a File", you may want to edit the file. You may also want to change the format of the file to use one of the display options that you did not specify at the time you unloaded the queue. You can use QLOAD to reprocess the file into the desired format even after the unload of the queue has taken place. Use the following options on the command line.

```
qload -f c:\oldfile -f c:\newfile -dA
```

Example 11. Display the ages of messages currently on a Queue

Use the following options on the command line.

```
qload -m QM1 -i Q1 -f stdout -dT
```

Example 12. Put messages to more than one queue

You can target more than one queue for the messages to be sent to using the following options. This can include using a fully qualified remote queue - in this case Q3 on QM2.

```
qload -m QM1 -I Q1 -o Q2 -o QM2/Q3
```

Example 13. Put messages to a list of queues in a file

With a file containing one queue name per line, you can use that file to specify the output queues.

```
qload -m QM1 -I Q1 -o file:c:\QueueList.txt
```

Example 14. Generic Unload

Unload all the non-empty queues to files in the current directory.

```
qload -m QM1 -i * -f *
```

For more information please refer to 'Generic Unload' on page 13.

Example 15. Generic load

Re-load all the unloaded files to their appropriate queues.

```
qload -m QM1 -o * -f *
```

For more information please refer to 'Generic Load' on page 17.

Example 16. Required Rate Processing

Copy messages from a source queue to a target queue at a predefined rate.

```
qload -m QM1 -i SOURCE -o TARGET -R1000:*
```

This command will copy messages from SOURCE to TARGET at 1000 messages per second. This is useful for emulating a workload. Pre-load the SOURCE queue with your test messages and then inject them into your processing application at the prescribed speed.

For more information please refer to 'Required Rate Processing' on page 19.

Example 17. Required Rate Indefinite Processing

Copy messages from a source queue to a target queue as fast as possible.

```
qload -m QM1 -i SOURCE -o TARGET -R*:*
```

This command will copy messages from SOURCE to TARGET as fast as they can be processed by the application processing the TARGET queue. This can be useful to get a rough idea of the processing capability of your system.

For more information please refer to 'Required Rate Processing' on page 19.

Example 18. Loading from a non-QLOAD file

If you have a file with message data, but not in the QLOAD format, you can still load it onto a queue as long as you can describe the delimiters of the messages within the file.

For example, with one line per message:-

```
qload -m QM1 -oQ1 -f messages.txt -Cd -nn
```

Or if you have some kind of tag that surrounds the messages:-

```
qload -m QM1 -oQ1 -f messages.txt -Cd -nns:"<msg>" -nne:"</msg>"
```

This file can also contain blocks of data representing message properties (or in fact any MQRFH2 folder). Again, you just need to describe the delimiters of both the message block and the property block.

```
qload -oQ1 -f msgs.txt -Cd -ns:"<msg>" -ne:"</msg>" -nip:"<usr>" -niq:"</usr>"
```

For more information please see 'Chapter 8. Reading non-QLOAD files' on page 26.

Chapter 5. Generic Unload and Load

Sometimes it can be useful to unload all the messages from a Queue Manager and then have the ability to re-load them back again. This could be, for example, for back-up purposes or Queue Manager migration. Clearly we could unload each queue manually with a separate QLOAD command but that would be awkward and error prone. We would need to use some form of scripting language to first retrieve the set of queues to unload. It would be nice to have QLOAD do all the hard work for us. And, of course, it can. The basic commands are very simple.

Generic Unload

Suppose we wish to unload all of the Queues on a Queue Manager. This is achieved very simply by issuing the following command:

```
qload -m QM1 -i* -f*
```

It is the '*' in the file name that tells QLOAD that this is a generic unload. Because the file name is not explicitly given it is derived from the queue name. If we were to issue this command QLOAD would unload all non-empty queues to a different file in the local directory[1]. The output of the program would show the progress it was making. An example of the output would be:

```
Q1                                                       44  Done.
Q2                                                        3  Done.
SALES                                                     2  Done.
SAMPLES                                                  10  Done.
SYSTEM.ADMIN.ACCOUNTING.QUEUE                          2480  Done.
SYSTEM.ADMIN.CONFIG.EVENT                                10  Done.
SYSTEM.ADMIN.PERFM.EVENT                                 34  Changing.
SYSTEM.ADMIN.QMGR.EVENT                                   1  Done.
SYSTEM.ADMIN.TRACE.ACTIVITY.QUEUE                      1490  37%
```

QLOAD will output a line per for each queue it unloads. During the unload process itself it will display a progress indicator to show how far through the unload process it is.

Before it starts the unload QLOAD will query all the non-empty queues in Queue Manager. It remembers the depth and it compares that to how many messages were actually unloaded. If they are different when the unload is complete it will output the words 'Changing' rather than 'Done'. This is to give you an indication that this queue appears to be active, In the case of the SYSTEM.ADMIN.TRACE.ACTIVITY.QUEUE above this is perhaps not surprising if activity trace is active. So, in this case, it would not be of concern. However, for some application queues it might be. It could be an indication that application activity has not yet ended and that the unload should be re-attempted when all applications and transactions have completed.

[1]Of course it is important to run this command under a userid which has the authority to view the queues you are interested in.

Of course you don't have to unload every queue. The -i parameter tells QLOAD which queues you are interested in using normal wildcard semantics. For example, we could have specified the following command.

```
qload -m QM1 -iAPP* -f*
```

This will, as you might expect, unload only queues that start with the characters 'APP'. By default the -i parameter is case insensitive though. If you wish to ensure that QLOAD matches the case of the supplied queue name then you should surround the name with single quotes. For example:

```
qload -m QM1 -i'APP*' -f*
```

Once the unload is complete QLOAD will output summary lines. For example:

```
Total : 24 Queues, 393 Messages
 plus : 15 System Queues, 5522 Messages
```

The reason the summary is split into two totals is that most system queues will not be uploaded by a generic upload. Essentially the first line tells you how many messages and queues would be uploaded by a generic upload. The second line tells you how many other system queues were unloaded.

QLOAD will unload **all** SYSTEM queues in a generic unload except the following:

- **SYSTEM.ADMIN.COMMAND.QUEUE**
- **SYSTEM.COMMAND.INPUT**

So, you will have a copy of most SYSTEM queues for back-up purposes.

However, most SYSTEM queues will not be loaded by a generic load operation. This is because most SYSTEM queues are intrinsically in-use by the SYSTEM and changing their content could cause a problem for the queue manager. The only SYSTEM queues which will be loaded on a generic load are:

- **SYSTEM.CLUSTER.TRANSMIT.QUEUE**
- **SYSTEM.DEAD.LETTER.QUEUE**

Of course if you really, really want to upload the content of a different SYSTEM queue then you can do so using the normal non-generic QLOAD command but in general it is not a good idea. If they truly are SYSTEM queues they are 'owned' by the Queue Manager and their content is particular to the current Queue Manager operation and generally speaking the messages on them should not be transferred between Queue Managers.

File location

When you issue a command such as this:

```
qload -m QM1 -i* -f*
```

The queues are unloaded into files in the current directory. However, it is possible to tell QLOAD to unload the files to a different directory.

For example:

```
qload -m QM1 -i* -fc:\unloads\*
```

Generally speaking it is sensible to unload the queues into an empty directory, or at least one that does not already contain unload queue files. This is to avoid one set of unloaded queues being contaminated by another. To further help this QLOAD will output a message if any unload files are detected in the directory and ask whether you are happy to delete them.

There are actually two such messages. The first covers files which match the queues about to be unloaded. The files must be deleted in order to continue. The second message covers files which have been discovered in the directory but which won't actually be impacted by the current unload since we are unloading a different set of files. For this second message QLOAD will ask whether you want to delete these files or leave them and continue. Bear in mind that if you do say continue your directory will contain files that are the sum of more than one unload operation.

If you use the -F, instead of the -f parameter, which signifies overwrite, the first of these messages will not be displayed and any existing files will be deleted and overwritten by the new download.

File Location on z/OS

While QLOAD will allow you to do individual queue unloads and loads using DD names to refer to the files, when using generic unload and load, the files created will be z/OS UNIX files and the pattern representing them, specified with the -f parameter, must be provided directly, and not using a DD name.

When you issue a command such as this, when running QLOAD in z/OS UNIX:

```
qload -m MQG1 -i* -f*
```

the queues are unloaded into files in the current directory. If you use the same command from TSO, or when running from JCL, the queues are unloaded into the home directory of the TSO user.

Alternatively, you can provide an explicit HFS directory to be used instead of the home directory:

```
qload -m MQG1 -i* -f/u/gemuser/unloaded/*
```

If you run the follow command from JCL, you must use double quotes round it to ensure that the PARM string does not pick up the '/' character.

```
//QLOAD    EXEC PGM=QLOAD,
//             PARM=('-m MQG1 -i* -f"/u/gemuser/qload/generic/*"')
//STEPLIB  DD   DSN=GEMUSER.USER.LOAD,DISP=SHR
//         DD   DSN=IBM.MQ.SCSQAUTH,DISP=SHR
//         DD   DSN=IBM.MQ.SCSQANLE,DISP=SHR
//SYSPRINT DD   SYSOUT=*
//SYSOUT   DD   SYSOUT=*
```

Generic File Name Format

By default the file name used for an unloaded queue is ***<QUEUENAME>.qld***

However, as is so often the case, it is not quite as simple as this.

File limits

The first consideration is the file size. If the queue is really deep and/or the messages are very large then it is possible that the unloaded file could itself be very large. Large files are unwieldy and are themselves limited in size by the file system. It therefore makes sense to ensure that files don't grow too large. By default QLOAD will set a maximum file size of 500MB for a generically unloaded file. You can, if you wish change this using the -Ls flag described in "Parameters Flags" on page 30. Any of the file limit parameters may be used. For example you could decide that no more than 5,000 messages should be stored in a single file. By default, generic unload does not limit the number of messages which can be stored in a single file, the limit is purely on size.

So, if we limit the file size this means that we may have to unload a queue to two, three or many different files. For example, if we were to unload a large queue called TEST we would end up with files such as:

```
TEST$1.qld
TEST$2.qld
TEST$3.qld
TEST$4.qld
```

A monotonically increasing index number is put on the end of the file name after a "$' character.

All sequences of files in this manner must start at index 1. There must be no gaps to the largest number. However, there are no limits on the number of files which can be stored.

Case sensitivity

As I'm sure you know MQ Object names are case sensitive. This means that it is perfectly legitimate to have queues called **SALES**, **Sales** and **SaLeS**. Note that we say that this is a legal thing to do, we are not saying it is a sensible thing to do. In fact, we would argue against it. However, it is possible so QLOAD has to be able to deal with it. If we left things as they are we would end up with files such as:

```
SALES.qld
Sales.qld
```

```
SaLeS.qld
```

And this would be fine on a Unix system where files are case sensitive anyway. However, on Windows, despite the file system knowing about 'case', these files would usually be treated as the same file. To avoid this QLOAD adds a unique qualifier to each file. So, the files actually used would be:

```
SALES.qld
SaLeS#1.qld
Sales#2.qld
```

So, the case of the Queue Name is stored in the file name itself but a unique qualifier is added to the file name to ensure that it will not clash with any other file.

The / character

Rightly or wrongly a Queue Name can contain a forward slash ('/') character. File names containing this character are either not allowed or not a good idea. So QLOAD will replace any '/' characters with an exclamation mark ('!').

File Type

Until now we have always assumed that the files have a file type of .qld. This is default but if you prefer you can change it. It is simply a matter of using the file type you want on the QLOAD command. For example in the following command:

```
qload -i* -f*.txt
```

This command will give all the files the file type of .txt.

Naturally you must remember to specify the same file type on the upload command, such as:

```
qload -o* -f*.txt
```

Of course you have to be careful that you don't have other .txt files lying around in that directory that QLOAD might also try to use. For that reason if you do decide to use your own file type it is best to use something that is firstly meaningful and secondly not in common use on your system.

Generic Load

As you might expect the generic load command is very similar to a generic unload. If you haven't already read the description of “Generic Unload“ on page 13 I suggest you do so now. The basic command for a generic load is very simple.

```
qload -m QM1 -o* -f*
```

This command tells QLOAD to look for any .qld file in the current directory and upload the contents of that file to the equivalently named queue on Queue Manager QM1. If we issue the command we might see something like the following:

```
Q1                                              Not empty.
SALES                                           RC(2085) Unknown object name.

There are potential problems with these queues.
Are you sure you want to continue?
```

Hopefully you won't see this output but it demonstrates the checks that QLOAD makes. Firstly QLOAD expects all the queues about to be uploaded to be currently empty. If they aren't it will warn you. This is to avoid unintentionally contaminating one set of messages with another. Clearly if you were restoring a backup or migrating a set of queues from one Queue Manager to another you would not expect the queues to already contain messages. However, if this is not a concern to you then you can say 'continue'.

QLOAD will also check at this point that you do have access to the required queues. Firstly this checks that the queue actually exists but it also checks that you have sufficient authority to put messages on the queue. If you choose not to continue at this point then none of your queues have been changed. If you do decide to continue then you might be displayed something like this:

```
Q1                                              44    Done.
SALES                                           RC(2085) Unknown object name.
SYSTEM.CLUSTER.TRANSMIT.QUEUE                   23%
```

Note that QLOAD went ahead and loaded the messages onto Q1. So, Q1 will now contain 44 additional messages to whatever it had before. Note we also try again to upload the 'SALES' queue. This means that when you get the first message telling you the queue does not exist you can, if you like, define the queue before saying continue.

In this display we can also see that QLOAD will display a progress indicator[2] while it is uploading a queue. In this case it is the SYSTEM.CLUSTER.TRANSMIT.QUEUE[3] and we are almost a quarter of the way through the upload.

As in the case of unload QLOAD will output a summary line once the upload is complete.

For example:

```
Total : 22 Queues, 265 Messages
```

If the upload completes normally then you would expect these numbers to match the first set of numbers in the generic unload. We know that in this case they won't match since we know at least one queue, 'SALES' which doesn't exist on this new Queue Manager.

[2]The progress indicator will not be shown on z/OS when running in batch or TSO. It will only be shown in z/OS UNIX.
[3]Remember that a generic upload will not upload most SYSTEM queues. SYSTEM.CLUSTER.TRANSMIT.QUEUE and SYSTEM.DEAD.LETTER.QUEUE are the exceptions.

Chapter 6. Required Rate Processing

There are times when you want to copy or move messages but only at a controlled rate rather than as fast as possible. The main reasons you might wish to do this are:

1. You would like to test that a server application can cope with a particular message rate.
2. You would like to know what message rate your server application can achieve.
3. You would like to know how many messages per second you can move across your channel.
4. You wish to inject lots of messages into your production system but you don't want to risk upsetting any running applications so you would like to 'trickle feed' in the messages.

QLOAD allows you to do all these things using the -R parameter in combination with the other standard QLOAD parameters. The simplest form of the command will just copy messages from a source queue to a target queue at a predefined rate. The main use of this command is test that an applications server can cope with a particular message load. First you create some test messages of the form you want your server application to process and put then on the 'SOURCE' queue and then you issue a command such as the following:

```
qload -i SOURCE -o TARGET -R500
```

This command will copy messages from the source queue to the target queue at a rate of 500 messages per second. In an ideal world the program would copy each message and then wait 1/500th of a second between each message. However, this is fraught with hazard. Firstly this method would only work if the MQPUT time took no time at all. Secondly Operating Systems do not do well with sleeps of small amounts of time because the process is continually giving up its time-slice[4]. So, instead QLOAD will put all messages as fast as it can and then sleep for the remainder of the second. Once QLOAD has copied all of the messages from the queue it will stop.

Of course the intention of this command is that you have an application processing the messages as they arrive at TARGET. This allows you to verify that the application is capable of processing messages at the given rate. Clearly if the application is not capable then the depth of queue TARGET will grow and grow.

One potential problem with this simple command is that it may be necessary to generate large numbers of test messages in order to run the command and see the effect. A slight variance on this command is to specify a duration for the test. This is specified by following the message per second value with another value which is the number of seconds for the test. For example:

```
qload -i SOURCE -o TARGET -R500:300
```

This command instructs QLOAD to continue copying messages for 5 minutes. The other key difference about this command though is that if QLOAD reaches the end of the source queue then it will just start again at the first message. It is possible, therefore, to run a test, putting 500 messages per second to a target queue with they only actually being a single message on the source queue. The same message is put to the target queue over and over again. In reality though it is recommended that you have at least 10 source messages to reduce the overhead of returning back to the start of the queue.

[4]If you really want to issue a sleep before each MQGET call you can do so using the -D parameter. Bear in mind though that, depending on the OS, it will not work well with really small sleeps.

The next variant of the command is to specify that you want the test to run indefinitely. This is done merely by specifying an asterisk (*) as the required duration. So, we get the following command:

```
qload -i SOURCE -o TARGET -R500:*
```

This command will run forever, or at least until there is a failure of some sort. For example, the Queue Manager ends or a queue fills up. Once the test has run long enough for you to gather your test information you will need to end the QLOAD program manually. This means something like pressing Ctrl-C in the command window or killing the process (using kill or task manager), or cancelling the job in SDSF, or disabling the target queue for put using an MQSC command like:

```
ALTER QLOCAL(TARGET) PUT(DISABLED)
```

Of course we don't have to *copy* the messages, we can choose to *move* them between the queues. The command to do this would simply be:

```
qload -I SOURCE -o TARGET -R50
```

This command is useful for trickle feeding in a set of messages into a running system where you don't wish to cause any issues to any running applications. For example, suppose you had 10,000 messages you wish to process. You could just put all 10,000 messages on to the TARGET queue but it is likely that this would affect the response time of any running applications. Of course the actual affect depends on a number of factors, principally how fast each message is to process. However, trickling the messages in is always going to be a safer option. The command above trickles the messages in at a rate of 50 messages per second which means that our 10,000 messages would take just over 3 minutes to process. Of course the rate you choose is entirely up to you and would depend on the type of request.

When you are moving messages, as opposed to copying them, you can not specify a duration. This is because there is no concept of going back to the beginning of the queue. QLOAD will run until it runs out of messages. That is not to say though that a command of this type can not run for a long time. All we need to do is add another parameter. Consider the following command:

```
qload -I SOURCE -o TARGET -R50 -w 86400
```

By adding the -w parameter we have told QLOAD to wait for up to 24 hours if it runs out of messages. So, now we can use QLOAD as a constant buffer between two queues. QLOAD will sit there waiting for messages to arrive on the SOURCE queue. When they do arrive they will be trickle-fed to the TARGET queue. So let's say, for example, you had an application that produces messages throughout the day and at the end of every day it dumped its entire operating log to a queue. We suppose that this sudden influx of messages causes problems, perhaps by clogging up your sender/receiver channels, you could use this command to spread out the workload.

Of course bear in mind that if, for some reason, no messages arrive on the source queue for an entire day then QLOAD will end. This can be mitigated by increasing the wait time, having QLOAD triggered in some way, or having QLOAD started automatically as a Cron job or similar.

Unlimited Rate Processing

The final variant of the -R parameter is where you don't want to specify the message rate but instead you are interested in what rate can be achieved. The command for this is very simple:

```
qload -i SOURCE -o TARGET -R*:*
```

Here we have specified an asterisk (*) for the message per second rate. This tells QLOAD to put messages to the target queue as fast as they are processed. If I issue the command on my system I get the following output.

```
qload -i SOURCE -o TARGET -R*:*

Queue Load/Unload Utility for IBM MQ
  Version V9.1.1 Build date:Aug  7 2019
  (C) Copyright MQGem Software 2015,2019. (http://www.mqgem.com)

Monitoring depth of queue 'TARGET'

Message Rate   802 msgs/sec
Message Rate  5200 msgs/sec
Message Rate  5548 msgs/sec
Message Rate  5516 msgs/sec
Message Rate  5409 msgs/sec
Message Rate  5566 msgs/sec
Message Rate  5503 msgs/sec
^C
```

Roughly every second QLOAD will output how many messages it has managed to put to the queue. It does this by adjusting the copy rate until a balance is achieved. You can, therefore, ignore the first few numbers as QLOAD tries to find the rate at which the processing speed remains reasonably consistent[5].

Bear in mind that the key difference between this speed test and others you may have seen is that in this case QLOAD is monitoring the target queue to ensure that the messages are being processed at the same rate that they are being put to the queue. It is not just simply trying to put messages to the queue as fast as possible. In this test to 'process' the messages I just used a simple sink program using another instance of QLOAD with the following command:

```
qload -I TARGET -f null -w600
```

In a real world example however you would have your target application actually processing the test messages and therefore there would be some 'think' time.

Now, as you can see MQ is capable of processing many messages a second. In fact on many systems you will see a number far higher than this. As a consequence it is important to have your maximum depth values on your queue set to a large number. By default the maximum depth of a queue is 5,000 which is likely to be too small. As you can see MQ can easily fill a 5,000 message queue in under a second. We recommend that if you are going to try a test of this sort that you would set your maximum depths to at least 20,000 and possibly far more. It all depends on the type and size of messages and the speed of your system.

[5]Of course with any test of this nature you are at the mercy of what ever else if happening on the system so complete consistency can be hard to achieve. The purpose of this type of test is not to give an exact number but to give an approximation, to try and determine the kind of numbers which can be achieved.

The next thing to say is that the message per second rate output by QLOAD should merely be taken as a ballpark figure. There are a huge number of things which can affect performance of a messaging system. As we have discussed before just the way the Operating System schedules processes can have a very significant effect. But we must also bear in mind that QLOAD is not 'doing much'; there is no 'think' time in the putting application so perhaps represents the best that can be achieved. Having said that there are down-sides to the QLOAD processing also which adversely affects performance. Most notably the fact that QLOAD has to put all the messages at the start of the second, rather than spreading them out during the second. This means that during that second that the queue depth will increase significantly and then be drained by the processing application. The target queue is therefore growing and shrinking rather than the more ideal situation of it staying roughly the same size. The first problem caused by this is that there is little chance for MQ to be efficient – what is often referred to a 'put to waiting getter'. The notion that the message can just be passed from one application to another. Instead, the vast majority of the messages will actually have to be put to the queue. The other problem is that the queue depth grows beyond what is most efficient. If a queues depth goes beyond the message buffer areas then the messages have to be spilled to disk which is not ideal and hurts performance.

Of course it is also possible to run all the Required Rate commands across a channel. For the Unlimited Rate case this raises another couple of issues.

So, to run this test across a sender/receiver channel we would issue a command something like:

```
qload -m QMA -i SOURCE -o QMB/TARGET -R*:*
```

Here we are reading from queue SOURCE on Queue Manager QMA and sending to a queue called TARGET on Queue Manager QMB. The key point to realise here is that QLOAD will be monitoring the depth of the transmission queue on QMA not the depth of queue TARGET on QMB. The assumption therefore is that the target application can keep up with the message rate. If it can't then the queue will become full and the receiving channel will go into 'paused' state which will essentially nullify the test. However, assuming that the target application can keep up, this test is useful because it will give you an indication of how fast messages of this type and size can be sent down the channel.

Chapter 7. Recovering Messages from the IBM MQ log

IBM MQ provides a command for formatting the IBM MQ log files on distributed platforms, called DMPMQLOG. This command will show you all the activity to the log files. One of the most important records amongst these are clearly the messages written to the queues. It follows therefore that, if persistent messages are deleted for some reason, it should be possible to use this output to reconstruct the messages themselves and put them back on a queue.

As with all recovery technology, such as recovering accidentally deleted files, there are very few guarantees. This facility in QLOAD is provided as a 'best can do' feature which may not work in your specific environment. One of the reasons for this is that although IBM provides the DMPMQLOG log formatter program they only go half way. They don't actually document the full format of the records themselves. So, we are in the slightly strange situation of having a supplied formatting program which doesn't actually fully format the data. However, the data format is fairly easy to determine and as such it is possible for us to add this feature to QLOAD to post process the DMPMQLOG output. Of course, one thing which is worth stating up front is that since the data format is not fully documented, we could find that the record format changes with a new IBM MQ release. As we said before there are no guarantees with this feature but if you do find that QLOAD is incapable of recovering a message then if you send us the log file in question we will do our best to modify QLOAD to accept the new format.

So, that being said, what other conditions must be met for message recovery

- **The message must be in the log file**
  This may seem obvious but whether an MQ message is actually written to the log file depends on a number of factors. Your best chance of finding the message in the log file is when it is a persistent message written to a Queue Manager running with a linear log file. Message recovery will work from either a circular or linear log file. However with a circular log clearly there is a considerable risk that the message has been over-written by subsequent messages.
- **Sufficient Authority**
  You need sufficient authority to issue the DMPMQLOG command. Usually this means that you need to be in the 'mqm' group.

General recovery syntax

Message recovery follows the same pattern as other uses of QLOAD. First you have an 'input' which, in this case, is the DMPMQLOG output and then you have an output which can either be a queue or a file. Between the two you can have a whole variety of filters to control which of the input messages get written to the output location.

Now, in terms of input you can actually have two modes of operation. QLOAD can either parse a previously generated DMPMQLOG output file, or it can invoke the DMPMQLOG command in the background and process it's output directly. The key parameter which dictates whether QLOAD is expecting to process a DMPMQLOG file is the existence of the **-j** parameter. The two basic versions of the command might look something like this:

QLOAD processing a DMPMQLOG output file

This format of the command would be used if you had already used the DMPMQLOG command to generate an output file and just wanted QLOAD to process the already created file.

```
qload -j c:\myfiles\dmpmqlog.out -i Q1 -f recover_Q1.qld
```

QLOAD will attempt to process the file directly if a file name is included in the **-j** parameter. Note that you can not use a file name with a file type of **.log**. This is avoid confusion with the actual MQ log files. QLOAD can not read the log files directly, it can only process the output of the DMPMQLOG command.

Clearly you need to arrange for the input file to contain the messages you are interested in. The first stage of this process would be to select the linear log files that are for the time interval you are interested in. These IBM MQ log files are then passed to the DMPMQLOG command. In addition, the DMPMQLOG command contains a number of options which allow you to control how much or little of the log files are processed.

The command above will process the input file and recover all messages found on queue Q1 and put the messages into file **recover_Q1.qld** This file can then be edited, emailed or loaded onto a queue as required.

If you wish you can put the messages directly to a queue using a command such as:

```
qload -j c:\myfiles\dmpmqlog.out -i Q1 -m MYQM -o RECOVERQ1
```

QLOAD issuing the DMPMQLOG command

This format of the command is used when you would like the messages recovered directly from the log file. The DMPMQLOG command is still issued but it is issued behind the scenes with no need to create an intermediary file.

```
qload -m MYQM -j* -i Q1 -f recover_Q1.qld
```

The asterisk[6] in parameter **-j** will look in the default location for the log files for Queue Manager MYQM and process the resultant DMPMQLOG output and put any message found to the output file.

Note that the DMPMQLOG command requires that the Queue Manager generating the log files is not currently running.

Again, if we wish, we can put the found messages directly to an MQ queue. However, note that, since we know MYQM must not be running, we need to use a different Queue Manager as the target for the messages.

```
qload -m MYQM -j* -i Q1 -m MYQM2 -o RECOVERQ1
```

You can, if you wish, be more explicit with the location of the log files. In this example you specify a path to the log files just as you would on the DMPMQLOG command itself.

```
qload -j c:\mylogfiles -i Q1 -f recover_Q1.qld
```

[6]Some Unix shells may interpret the asterisk. In this case do not put a space between -j and the asterisk. Alternatively, put quotes round the parameter.

As mentioned before the DMPMQLOG command allows you to specify some filtering fields. You can specify a start and end LSN. QLOAD allows you to specify these after the log path so your **-j** parameter would be *LogFilePath(StartLSN - EndLSN)*. For example,

```
qload -j c:\mylogfiles(0:0:0:0-0:0:72:51707) -iQ1 -f recover_Q1.qld
```

Either the start or end LSN can be omitted if required. If only the end LSN is specified then just precede the value with a '-' (minus) sign.

Performance

Clearly log files can be large. It follows therefore that any program reading the log files can have a lot of work to do and may take a considerable time to do its task. Sometimes this doesn't matter and the program can be run over-night, or whatever, to parse through the entire log. However, if performance is a factor then try to ensure that the input file is as targeted as it can be, perhaps by using the date of the MQ log files.

If you plan on recovering more than one queue, then running the DMPMQLOG command first and having QLOAD process the resultant file will use less CPU since you only need to format the MQ logs once.

In all cases QLOAD will output a regular status line so that you can see that it is processing and have some idea of the speed it is running. The status line looks like the following:

```
DMPMQLOG Records : Line:498090 Records:11420 Messages (2670, 10, 5)
                        A              B                C      D   E
```

The inserts to the status line are as follows:

- **A** The line number currently being processed
- **B** How many log records have been found so far
- **C** How many messages have been found so far
- **D** How many messages for the required input queue name have been found
- **E** How many messages have been written to the output location (this may differ from **D** if you are filtering the messages).

Chapter 8. Reading non-QLOAD files

Generally speaking QLOAD will be used to unload queues to files and then reload those files back to either the same or different queues. However, there are times when it might be useful to start from a message in a file and be able to load those messages to a queue. This could be, for example, when you are creating some test messages or perhaps you wish to save sequences of messages for a message processor.

In order to tell QLOAD to read a non-QLOAD file you are given various flavours of the '`-n`' flag. The existence of any of the `-n` flags tells QLOAD that any file parameter should be treated as a non-QLOAD file.

There are essentially two ways to process a non-QLOAD file. Either the file should be treated as a single message, or the file should be parsed looking for multiple messages (and optionally blocks of property data). These are both discussed below.

Loading an entire file as a single message

This is clearly the simplest way of reading a non-QLOAD file. The command flag to indicate that the entire file is a single message is `-n1`. So, for example the command to load a file 'mymessage.txt' and put it as a single message would just be the following:

```
qload -oQ1 -f mymessage.txt -Cd -n1
```

In this case the entire message will be read and put to the queue as a single message. You may be wondering what the `-Cd` parameter is doing. Well, by default QLOAD will try to use 'set all context' so that the Message Descriptor values match the incoming source values. However, in this case there aren't any message descriptor values incoming. So, we need to tell QLOAD to use the default context values, hence `-Cd`.

Loading multiple messages from a single file

Now, if our file may contain multiple messages it follows that QLOAD needs to be told how to parse the file. Essentially what are the delimiters for the messages? The simplest scheme one could imagine is that each line of the file is a separate message and this is indeed the default. The following command will read a file and put each line of the file as a separate message.

```
qload -oQ1 -f messages.txt -Cd -nn
```

Here we have the `-n` parameter to tell QLOAD that the file is not an actual QLOAD file and we have the single flag '`n`' which says that the delimiters should not be included in the message (which is the default).

Issue this command and the first thing you will see is a message something like the following:

```
Loading 'messages.txt' : 3 messages found
Do you want to continue? (y/n)
```

This can be a useful double-check that the file contains what you were expecting. For example if you mistyped the delimiter values then you might find that the interpreted file contains a significantly different set of messages. If you enter 'n' at this point then QLOAD will end having done nothing. However, if you enter 'y' then QLOAD will re-parse the file and put the appropriate messages to the queue.

Similarly you can load multiple files with the command:

```
qload -oQ1 -f messages%n.txt -Cd -nn
```

This will load all the files it finds in the pattern message1.txt, messages2.txt etc.In this case you might see the following confirmation message:

```
Loading 'messages1.txt' : 5 messages found
Loading 'messages2.txt' : 8 messages found
Loading 'messages3.txt' : 10 messages found
Loading 'messages4.txt' : 4 messages found
Do you want to continue? (y/n)
```

If you don't want or need this initial confirmation message (for example if you are running in a script) then you can bypass it by using the '`-nF`'. So, the command would be:

```
qload -oQ1 -f messages.txt -Cd -nF
```

Note that we don't need the '`-nn`' flag since not including the delimiters is the default anyway.

Of course having each line of the file being a message can be a little restrictive. Sometimes it would be useful to be able to specify delimiters for the messages. Let's consider the case where we want to use '<' and '>' to delimit the messages. So, in the file we have something like this:

```
<Message 1>
<Message 2>
<Message 3>
```

To parse this command we need only enter a command such as the following:

```
qload -oQ1 -f messages.txt -Cd -n"s:<" -n"e:>"
```

Unfortunately the '<' and '>' characters have special meanings in most shells so we have to enclose them in quotes to make sure the shell doesn't think we are trying to redirect the output of the program of anything like that. The exact way you do this may vary by shell but the given command should work in most cases.

If your start delimiter and end delimiter characters have the same number of characters, as in this case, then you can use a shorthand by specifying them in the same parameter with the `-nd` flag instead of the `-ns` and `-ne` flags, as shown in the following:

```
qload -oQ1 -f messages.txt -Cd -n"d:<>"
```

These commands have not included the delimiters themselves as part of the message content, but if you want to include them, you can use the `-ni` flag as shown in the following command:

```
qload -oQ1 -f messages.txt -Cd -n"id:<>"
```

The delimiters themselves can be up to 100 characters. So, they could, for example, be XML tags or similar.

The logical amongst you might realise that the file may now contain characters which do not fall between these delimiters and wonder what happens to them. Well, the answer is nothing, these characters can be treated as comments. This allows you to have a file something like the following:

```
Switch on the porch light
<light 23 on>
Open the driveway gate
<device 14 on>
```

Such words allow you to annotate your message files which can be useful for documenting the work of the file or the tests that you are running.

Loading message properties from a delimited file

In addition to supplying delimiters for message content in a non-QLOAD file, you can also have sections of the file containing message property content. You can supply an additional pair of delimiters and QLOAD will add the message property content to the message as it is put to the output queue.

Message property content found before each message will be associated with the message when put to the queue. As an example look at the file content example below:

```
<usr><Colour>Pink</Colour></usr>
<msg>This is a pink message</msg>
<usr><Colour>Blue</Colour></usr>
<msg>This is a blue message</msg>
```

This file could be parsed with the following command, where the <usr> tags must be included so that the MQRFH2 folder is treated as message properties, and in this example, the <msg> tags are not included and are only present to allow the message text to be correctly parsed in the file, but your text file may be different.

```
qload -o Q1 -f msg.txt -Cd -n"ns:<msg>" -n"ne:</msg>" -n"ip:<usr>" -n"iq:</usr>"
```

Issue the above command and you will see output something like this, helping to confirm that you have used, and QLOAD has found, the correct delimiters and blocks of data in your file.

```
Loading 'msg.txt' : 2 messages found (2 with properties)
Do you want to continue? (y/n)
```

Continuing will result in two messages being found and put on queue Q1, with each message containing a single property named 'Colour' with a different value in each case.

QLOAD takes the data between the tags you supply and adds it to the front of the message in an MQRFH2 folder. If the data is a <usr> folder, then IBM MQ will interpret this as message properties. If any other folder, then it is just an MQRFH2 folder, so you can use this technique to add any MQRFH2 folder to the front of your message data as required.

NOTE: The Name Value Data in an MQRFH2 header must always be in ASCII, specifically a Unicode code page. Please remember this when viewing messages created in this way on z/OS. The text in the Name Value Data section of the MQRFH2 header has been converted to 1208 before being written to the message.

Often you may be copying property data from output that has been formatted. This means you gain a lot of insignificant white space, which QLOAD will strip by default. In case QLOAD is over-zealous in this removal, you can use the `-nW` flag to keep all the white space that is in your input file.

Special Characters

It is possible that your delimiters could contain characters which are not easily given on the command line. The newline character is one such example. You can, therefore, specify special characters:

• \n	Newline characteristics
• \\	Backslash character
• \<decimal number>	Character code

For example, you could specify your delimiter such as the following:

```
qload -oQ1 -f messages.txt -Cd -n"s:\14" -n"e:\15\16"
```

Chapter 9. Parameters

There are a number of switch parameters that can be passed to QLOAD to control the behaviour you need. These are detailed in this chapter.

Parameters Flags

The following parameters can be passed to the program as flags on the command line.

Flag	Meaning
`-a`	Controls whether the file is opened in binary or append mode **-aa** Append **-ab** Binary
`-A`	Allowed the user to override certain message attributes from the original source. **-Ac:<ccsid>** Message Codepage **-Ae:<expiry>** Message Expiry (in tenths of a second) **-Af:<format>** Message Format **-Ap:<priority>** Message Priority **-Ar:<ReplyQ>** Message Reply Queue and optionally Reply Queue Manager. **-Ar:REPLYQ** **-Ar:QM1/REPLYQ** Use of either will set Msg Type to **REQUEST** **-At:<Msg Type>** Message Type: **REQUEST**, **REPLY**, **DATAGRAM**, **REPORT**
`-b`	Initial message buffer size in Kilobytes (Kb)
`-c`	Controls whether messages taken from a queue are converted -c < CCSID> [: X 'Encoding'] For example ***-c850:111***
`-C`	Context options **-CA** Set all context (Default) **-CI** Set identity context **-Ca** Pass all context **-Ci** Pass identity context **-Ct** Set all context and update times **-Cd** Default context **-Cn** No context Use of the 'pass' context options is restricted to cases where the source messages are consumed from a queue. For a fuller description of context options please see 'Context Options' on page 48.

`-d`	Display options. In many cases more than one flag can be specified. For example, `-dsCM` **-da** — Add ASCII columns to the hex output in the file to aid readability **-dA** — Write ASCII lines of data wherever possible **-dc** — Output ApplicationOriginData and ApplicationIdentityData as characters **-dC** — Display Correlation Id in queue summary **-dH** — Don't write file header[7] **-di** — Include the message index in the output **-dk** — Count the messages in the input source (queue or file) **-dK** — Count the messages in the input source (queue or file) and output only the resultant number. **-dM** — Display Message Id in queue summary **-dN** — Don't write out the message descriptor content, only the message payload. **-dp** — Printable character output format[8] **-dr** — Write message content as raw bytes Note that files output in raw mode are not loadable since the message data is not encoded in any way. **-ds[-]** — Write a simple summary of the messages found on input Add '-' after the 's' to suppress the use of Inquire Queue Status in this output. **-dt** — Text line output format[8] **-dT** — Display the time the message has been on the queue **-dw<Length>** — Set the data width for the output

[7]Note that files created with this option will not be loadable by QLOAD since it will not recognise the file format. However, if necessary an appropriate header can be manually added using an editor and then the file will be loadable.

[8] The `-dt` output format is only suitable for a small range of messages. In particular it is not codepage safe. It is not recommended that you use this format if you intend to re-load the file to a queue or if your messages contain non-printable characters.

`-D`	Add delay, expressed in milliseconds, before writing message to output destination. For really short delays consider using the `-R` parameter. **-D <+ve value>** Add a fixed delay before putting message `-D500` would put each message half a second apart **-D <-ve value>** Add a random delay up to the specified value before putting message `-D-10000` adds a random delay of up to 10 seconds before putting message **-D r <value>** Replays the messages at a percentage of their original put speed **-Dr** Replays messages at original speed **-Dr50** Replays messages at twice original speed **-Dr200** Replays messages at half original speed
`-e`	This parameter allows message selection based on the content of the message. For a complete description of message selection please see 'By Search String' on page 54. This parameter may be specified up to 3 times; see Multiple Search Strings for a description.
`-E`	This parameter allows message selection based on the content of the message. For a complete description of message selection please see 'By Search String' on page 54. This parameter may be specified up to 3 times; see Multiple Search Strings for a description.
`-f`	Specifies either the source or target file name. For a complete description of the filename format please see 'File Use' on page 42.
`-F`	Specifies either the source or target file name. For a target file it forces output to file if it already exists. The program will not ask whether the file should be overwritten. For a complete description of the filename format please see 'File Use' on page 42.
`-g`	Filter by Message Id, Correlation Id or Group Id **-gc<Value>** Get by character Correlation Id **-gm<Value>** Get by character Message Id **-gg<Value>** Get by character Group Id **-gxc<Value>** Get by hex Correlation Id **-gxm<Value>** Get by hex Message Id **-gxg<Value>** Get by hex Group Id

`-G`	Get filter options This parameter adds additional filtering capability. The flag is followed by a letter indicating the type of filter followed by ':' and possibly a value. The `-G` flag can be used multiple times **-Gi:** — Inverse selection. This can be used to inverse the current selection criteria. This can only be used in cases where QLOAD is doing the filtering. For example it can not be used to inverse the SQL92 Selector. **-Gr:<Reason Code>** — Will filter only messages with a Dead Letter Queue header containing the given reason code. **-Gt:[TargetQm/][TargetQ]** — Will filter messages which are targeted at the given Queue Manager and Queue Name. If a Queue Manager or Queue name is not given then a blank value is matched. The Queue Manager and Queue names can contain wildcards, e.g. `-Gt:*/APP*`
`-h`	Strip headers Any Dead Letter Queue header (MQDLH) or Transmission Queue header (MQXQH) will be removed from the message before being written.
`-H`	Specifies the SQL92 Message Selector For example **`-H "Value > 100"`** For more information please see “Selection by SQL92 Selector“ on page 57.
`-i`	Specifies the input queue to browse. The parameter can be specified as many times as you wish to browse messages from multiple queues. For example **`-iQ1 -iQ2 -iQ3`** Please see “Generic Unload“ on page 13 to see how multiple queues can be unloaded at once.
`-I`	Specified an input queue to consume messages from. The parameter can be specified as many times as you wish to consume messages from multiple queues. For example **`-IQ1 -IQ2 -IQ3`**
-j	Specifies the MQ log recovery location to process. This parameter does not apply on z/OS. For information on the format of this parameter please see Chapter 7 Recovering Messages from the IBM MQ log on page 23.

Sub-table of `-G`:

-Gi:	Inverse selection This can be used to inverse the current selection criteria. This can only be used in cases where QLOAD is doing the filtering. For example it can not be used to inverse the SQL92 Selector.
-Gr:<Reason Code>	Will filter only messages with a Dead Letter Queue header containing the given reason code.
-Gt:[TargetQm/][TargetQ]	Will filter messages which are targeted at the given Queue Manager and Queue Name. If a Queue Manager or Queue name is not given then a blank value is matched. The Queue Manager and Queue names can contain wildcards, e.g. `-Gt:*/APP*`

`-l`	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. **-lmqm** Local Application **-lmqic** Client Application 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`	Specifies the output file limits. This parameter controls the point at which a new file is generated by QLOAD when unloading a queue to multiple files. **-La<Age>s|m|h** Limit the file age by specifying a maximum file age in seconds (s), minutes (m) or hours (h). For example, the file is used for no more than 2 hours: **-La2h** **-Ls<Size>b|k|m|g** Limit the file size by specifying a maximum file size in bytes (b), kilobytes (k), megabytes (m) or gigabytes (g). For example, the file is limited to 500 kilobytes: **-Ls500k** Note there is no guarantee that all files will be below this size. It specifies a preferred maximum size. For example, it could be possible that a single unloaded message causes a file to be created that is larger than the provided limit. In which case, a file will be created that contains just this one message. **-Lm<Number>** Limit the file size by specifying a maximum number of messages per file. For example, the file contains at most 200 messages: **-Lm200** These limits can be combined so that the first limit reached causes a new file to be used. Use of this parameter requires the use of file index inserts. For a complete description of the filename format please see ‘File Use’ on page 42.

-m	Specifies the name of the Queue Manager to connect to 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 QLOAD. For example ***-m QM1 -i INPUT.Q -m QM2 -o OUTPUT.Q*** Similarly, parameters which qualify the queue manager connection, such as `-l` (to qualify it as a client connection) or `-u` (to provide a connection time user id and password), 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***
-n	Specifies that the input file is a non-QLOAD file. Multiple flags can be specified in the same parameter e.g.: `-nnFd:<>` For a fuller description of these options please see ‘Chapter 8. Reading non-QLOAD files’ on page 26.

-n1	Read the entire file as a single message
-nd:<value>	Specifies both start and end message delimiters. eg `-nd:<>`
-ne:<value>	Specifies the end message delimiter. E.g. `-ne:>`
-nF	Specifies than no preliminary message count is needed
-ni	Include the delimiters in the message
-nn	Do not included the delimiters in the message
-np:<value>	Specifies the start property delimiter. E.g. `-np:"<usr>"`
-nq:<value>	Specifies the end property delimiter. E.g. `-nq:"</usr>"`
-ns:<value>	Specifies the start message delimiter. Eg `-ns:<`
-nW	Don't strip white space from properties
-n!	Allows the start and end delimiters to be the same

`-o`	Output Queue Name For example ***-o Q2*** It is also possible to qualify the queue name with the name of the Queue Manager using any of the characters '/', '\', '#' and ',' as separators, or by specifying your own separator character using the `-$` flag. For example ***-o QM2/Q2*** The `-o` parameter can be specified as many times as you like and QLOAD will put messages to all the destinations. For example ***-o Q1 -o Q2 -oQ3*** will put the same messages to Q1, Q2 and Q3. In place of a queue name, you may also specify a destination file which contains the list of queues you wish to put to. For example ***-o file:mydestinations.dst*** For more information please see 'Destination File' on page 49. Wild cards can be used in the Queue name if you are doing a generic load. For more information see “Generic Load“ on page 17.
`-O`	General Option Flags **-Oa** Use Asynchronous put when loading message on to a queue. This option can yield significant performance benefits when running over an MQ Client. **-OB** By default QLOAD will MQINQ the input queue to discover if there is a backout threshold and a backout re-queue name defined. Use this flag to suppress the MQINQ and thus any backout processing by QLOAD. **-Oe** Open the input queue for exclusive use. **-OF** Do not use 'Force MQRFH2' option[9] **-Oi** Print machine and userid information to the screen. Use this command to ensure you supply the correct information when buying a licence. **-Or** Use Read Ahead when getting messages from a queue. This option can yield significant performance benefits when running over an MQ Client. **-OI** Use individual queues instead of distribution lists for multiple destinations.[10]
`-p`	If set this option will cause the source queue to be purged of messages as they are copied to the target destination.

[9]This option is automatically applied if the connected Queue Manager is before Version 7.0
[10]This option is automatically applied if QLOAD is running on z/OS

`-P`	Message Priority Message Selection (lowest,highest) **-P 4** Select message with priority >= 4 **-P 4,4** Select only messages that are priority 4 **-P 2,6** Select message with priority >= 2 and priority <= 6 **-P ,6** Select messages with priority <= 6
`-q`	Sets quiet mode. If this flag is set the program will not output it's usual summary of activity.
`-r`	Sets the applicable message range **-r x** Just the 'x'th message For example `-r10` **-r x..y** From message 'x' to message 'y' For example `-r 10..20` **-r x#y** Output 'y' messages starting at message 'x' `-r 100#10` **-r#x** The first 'x' messages For example `-r#100`
`-R`	This parameter controls the Require Rate processing. **`-R <Message Per Second> [ : Duration in seconds ]`** Required Rate is useful if you would like to inject messages into a queue at a controlled rate. The messages themselves can be copied or moved depending on whether -i or -I is used. However, you can not specify a duration if you are moving the messages. **-R 500** Copy messages to target queue at 500 messages per second until end of queue **-R 500:600** Copy messages to target queue at 500 messages per second for 10 minutes **-R *:600** Copy messages to target queue as fast as they are being processed for 10 minutes **-R *:*** Copy messages to target queue as fast as they are being processed indefinitely. For more information please refer to "Chapter 6. Required Rate Processing on page 19.

`-s`	This parameter allows message selection based on the content of the message. For a complete description of message selection please see ‘By Search String’ on page 54. This parameter may be specified up to 3 times; see Multiple Search Strings for a description.
`-S`	This parameter allows message selection based on the content of the message. For a complete description of message selection please see ‘By Search String’ on page 54. This parameter may be specified up to 3 times; see Multiple Search Strings for a description.
`-t`	Sets the transaction message limit. For a description of the use of transactions please see ‘Transactions’ on page 47
`-T`	This parameter allows message selection based on time on queue. For a complete description of the options available please see 'Selection By Time on Queue' on page 55.
`-u`	The userid that should be used for the connection. If a password is not provided, using the `-U` parameter, then QLOAD will prompt for the password. See Passing in a Userid & Password on the connection on page 41 for more details. On z/OS, QLOAD will only prompt for the password when running in z/OS UNIX, but not when running in TSO or in Batch from JCL.
`-U`	The password that goes with the userid, `-u`, parameter.
`-v`	Verbose flags. Flags which controls what additional information QLOAD should output. **-va** Display MQI usage **-vd** Display the set of output destinations. This can useful when the destinations themselves have been read from a file. **-vs** Display periodic status (based on messages) The 's' character can be followed by a number. The number indicates after how many messages the status should be written. For example, to write a status record after every 100 message use the flag **-vs100** The default number of messages is 1000. **-vt** Display periodic status (based on time) The 't' character can be followed by a number. The number indicates after how many seconds the status should be written. For example, to write a status record every minute use the flag **-vs60** Note that the status is only written when a message is sent. If QLOAD is not moving messages it will not continually be writing out status messages. The default number of messages is 300 (5 minutes) Multiple verbose flags can be specified at the same time. For example **`-vads10`**

`-w`	Wait Interval, in seconds, for consuming messages. If specified the program will wait for messages to arrive, for the specified period, before ending. A value of -1 can be used to indicate an indefinite wait.
`-x`	This parameter allows message selection based on the content of the message. For a complete description of message selection please see 'By Search String' on page 54. This parameter may be specified up to 3 times; see Multiple Search Strings for a description.
`-X`	This parameter allows message selection based on the content of the message. For a complete description of message selection please see 'By Search String' on page 54. This parameter may be specified up to 3 times; see Multiple Search Strings for a description.
`-z`	Message Size Message Selection (smaller,larger) Message sizes can be specified in bytes, KB or MB **-z 10** Select message of size >= 10 bytes **-z 10K** Select message of size >= 10KB **-z 10240,10241** Select message of exactly size 10KB **-z 1K,1M** Select messages of size >= 1KB and < 1MB **-z ,10MB** Select message of size < 10MB
`-$`	This parameter allow you to specify which character will be used as a separator between Queue Manager name and Queue name when providing a fully-qualified queue name for use as the input or output queue.

Getting help from the command

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

- **All Parameter overview**
  ```
  qload
  ```
  By entering the command on it's own QLOAD will output the parameters it accepts

- **All parameters**
  ```
  qload -?
  ```
  This command will output all the parameters and information about the options for each parameter.

- **Parameter information**
  ```
  qload -d?
  ```
  This command will give information about the -d parameter

- **Parameter search**
  ```
  qload -?file
  ```
  This command will output any parameter usage text containing the string 'file'

Connection Methods

You have two connection methods available to you with QLOAD. 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.

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' parameter 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:-

```
qload -m QM1 -iQ1 -fstdout
```

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

```
qload -m QM1 -iQ1 -fstdout -l mqic
```

Passing in a Userid & Password on the connection

QLOAD supports the userid/password feature provided by IBM MQ in Version 8.0. 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:

```
qload -m QM1 -iQ1 -fstdout -u myuserid -U mypassword
```

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

```
qload -m QM1 -iQ1 -fstdout -u myuserid

Queue Load/Unload Utility for IBM MQ
  Version V9.1.1 Build date:Aug  7 2019
  (C) Copyright MQGem Software 2015,2019. (http://www.mqgem.com)

Please enter password for QM(NTPGC1) User(myuserid) >****
```

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

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

File Use

As already seen in “Chapter 4. Examples” on page 8, the `-f` flag is used to indicate the file name.

If the file name contains an asterisk ('*') character it signifies that you wish to do a generic unload or load. Please see “Chapter 5. Generic Unload and Load“ on page 13 for a description of this feature.

Otherwise messages are targeting a particular file (or group of files). This file may already exist in which case the program will ask whether you wish to overwrite it or not. If you select not to overwrite the file, no messages will be unloaded and the utility will end. You can select to overwrite the file when asked this question, or you can specify that the file to be used should be overwritten if it exists by using the `-F` (upper case) option as shown, on the command line.

```
qload -m QM1 -i Q1 -F c:\myfile
```

If you wish to combine the messages from two queues into one file, you can use your operating system services to concatenate the two files together and the format will still be acceptable to QLOAD.

You can also use the `-f` flag to specify **`stdout`** as the output source instead of a file name. This may be useful if you simply wish to display the messages on the queue to the screen, or pipe the output another program.

```
qload -m QM1 -i Q1 -f stdout
```

Another special value recognised by QLOAD for use in the `-f` flag is **`null`**. This is used to discard the output and if messages were destructively got from the queue with the `-I` flag, this would result in them being discarded from the queue.

```
qload -m QM1 -I Q1 -f null
```

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):

```
qload -m MQG1 -i Q1 -f"//'GEMUSER.MY.FILE'"
qload -m MQG1 -i 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):

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

However, this does not work from within a PARM string in your JCL because of the single quotes, so when using this from JCL, you must use the following:

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

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

```
qload -m MQG1 -i Q1 -f"/u/gemuser/Q1.qld"
```

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:

```
//OUTFILE DD DSN=GEMUSER.MY.FILE,DISP=SHR
//OUTFILE DD DSN=GEMUSER.MY.PDS(MEMBER),DISP=SHR
//OUTFILE DD PATH='/u/gemuser/Q1.qld',PATHOPTS=(ORDWR,OCREAT)
```

you can then use the following:

```
qload -m MQG1 -i Q1 -fDD:OUTFILE
```

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

File Insert Characters

QLOAD is capable of writing and reading to multiple files in a single command. This can be useful if you wish to unload a queue where each message, or number of messages, goes to a separate file; or if you want to load multiple files to a queue. To facilitate this a special insert characters can be placed in the file name which will be replaced with a value when the program runs. Please note that these inserts are not supported, nor necessary if you are doing a Generic Unload.

Inserts are specified by preceding the character with a ‘%’ character in the file name. For example, using the command **qload -iQ1 -fQ1_Unload_%c** will actually write to the file ‘Q1_Unload_061014’. In other words it will append the date to the file name.

The full list of inserts is as follows:-

Index Inserts	
i	Input message number
I	Input message number padded to 5 characters eg. 00001
o	Output message number
O	Output message number padded to 5 characters eg.00001
n	Current File number
N	Current File number padded to 5 characters eg.000001

Date Inserts	
c	Current date in YYMMDD format This format is used to enable the files to be sorted alphabetically. Alternatively a different date format can be constructed using the inserts below.
C	Current date in YYYYMMDD format
d	Two digit day of month
dd	Day of month including suffix eg.1st, 2nd, 3rd …
j	Zero based Julian day of year – January 1st equal '000'
J	One based Julian day of year – January 1st equal to '001'
m	Three character month name eg.Jan,Feb,Mar….
mm	Two digit month
mmm	Full month name eg. January, February,March…
y	Four digit year
yy	Two digit year
D	Three character day of week eg.Mon,Tue,Wed…
DD	Full character day of week eg. Monday, Tuesday, Wednesday…
Time Inserts	
t	Simple time format eg. 181403
H	Two digit hour (24 hour clock)
HH	Hour (24 hour clock)
h	Two digit hour (12 hour clock)
hh	Hour (12 hour clock)
M	Two digit minutes
S	Two digit seconds
P	AM/PM
p	am/pm
Other Inserts	
%	The '%' character

Table 1: File insert characters

File Insert Examples

In the following examples assume that we have a queue called Q1 which contains 10 messages where each message contains the string which is their message number. Eg. “One”, “Two”,”Three” etc etc

- **Adding the current date and time to an output file**

  ```
  qload -m QM1 -iQ1 -f c:\myfile_%c_%t
  ```

  This will help to ensure you don't over-write a previously created file.

- **Writing each message on a queue to a separate file**

  ```
  qload -m QM1 -iQ1 -f c:\myfile%n
  ```

  Will write each message in Q1 to a separate file myfile1, myfile2, myfile3….

- **Writing five messages to each file**

  ```
  qload -m QM1 -iQ1 -f c:\myfile%n -Lm5
  ```

  Will write each of five messages from Q1 to a separate file myfile1, and myfile2.

- **Loading a series of files onto a queue**

  ```
  qload -m QM1 -oQ1 -f c:\myfile%n
  ```

  Will load the files myfile1, myfile2, myfile3….The loading will stop as soon as a file does not exist.

- **Selectively reading messages from a queue**

  ```
  qload -m QM1 -iQ1 -se -f c:\myfile%o
  ```

  The -se parameter says that only messages containing the character ‘e’ should be written.
  Will write each message in Q1 to a separate file myfile1, myfile2, myfile3….

- **Selectively reading messages from a queue**

  ```
  qload -m QM1 -iQ1 -se -f c:\myfile%i
  ```

  The -se parameter says that only messages containing the character ‘e’ should be written.
  Will write each message in Q1 to a separate file myfile1, myfile3, myfile5,myfile7….
  Note that the %i character is the index of the **input** message. Consequently the file names correspond to the message index which caused the file to be written. In this example, therefore you get the file indexes of the numbers which contain the letter ‘e’.

- **Reformatting a selection of files**

  ```
  qload -m QM1 -fInFile%n -da -f -fOutFile%n
  ```

  This command will read the files, InFile1, InFile2 etc and write them reformatted to files OutFile1,OutFile2 etc. The program will stop as soon as an input filename is reached which does not exist. The number of output files will match the number of input files.

- **Reformatting a selection of files containing multiple messages**

  ```
  qload -m QM1 -fInFile%n -da -f -fOutFile%o
  ```

  This command will read the files, InFile1, InFile2 etc and write them reformatted to files OutFile1,OutFile2 etc. The program will stop as soon as an input filename is reached which does not exist. The number of output files will match the number of input messages – that is some of the input files may have had more than one message contained within them.

Queue access options

As already seen in "Chapter 4. Examples" on page 8, the `-o` flag is used to indicate the output queue, that is the queue to which messages are put; the `-i` and `-I` flags are used to indicate the input queue, that is the queue from which messages are browsed or destructively got.

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

```
qload -m QM1 -i Q1 -f c:\myfile -c 850:X'222'
```

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

```
qload -m QM1 -i Q1 -f c:\myfile -c 0
```

Backout re-queue Queue

If you define your input queue with a backout re-queue name (`BOQNAME`) and a non-zero backout threshold (`BOTHRESH`), QLOAD will make use of these and copy or move (depending on your input mode) any messages found on the input queue that have an `MQMD.BackoutCount` at or above the threshold to the nominated re-queue queue. In order to do this, QLOAD will inquire the details of the input queue which means the user id running QLOAD must have authority to inquire the queue. If it does not, QLOAD will report that backout processing is disabled and continue. If you wish to suppress this inquire authority requirement, or stop backout processing from happening for any other reason, you can disable it by adding the `-OB` flag to your command.

The re-queued message will be put with `MQPMO_SET_ALL_CONTEXT` to ensure that all the fields in the MQMD are retained. Please ensure your user id running QLOAD has authority to put to the re-queue queue with "set all context". Processing will halt with a failure if it does not.

Transactions

By default QLOAD will try to do a group of messages in a single transaction in order to improve performance. However, there is no guarantee that the WHOLE operation will be done in a single transaction. By default the transaction will be limited to 200 messages. To change this you need to use the `-t` parameter to say how many messages can be done in a single transaction.

Option	Meaning
`-t 0`	Switch off transactions all together
`-t-1`	All messages will be done in a single transaction
`-t n`	The message operations will be split into groups of n messages, for example `-t1000` would deal with 1000 messages in a single transaction.

Table 2: Transaction flag values

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.

Transactions across Queue Managers

QLOAD 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 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 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.

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 QLOAD is to set all the context information in the MQMD to that which was saved in the file being loaded. This requires the user ID under which the utility is running to have appropriate authority to set all the context fields.

The context fields can be manipulated in other ways using the various other options with the `-C` flag. "Table 3: Context options used by QLOAD" details the options that can be used. The main difference to note is that the first three options are applicable when loading from a QLOAD format file, and the next two options are applicable when loading one queue from another queue.

Option	Meaning
A	Set all the context fields in the MQMD to that which was saved in the file.
t	Set all the context fields in the MQMD to that which was saved in the file apart from `PutDate` and `PutTime` which are set to values reflecting the current time.
I	Set only the identity context fields in the MQMD to that which was saved in the file.
a	Pass all the context fields in the MQMD from the messages on the input queue to the messages on the output queue.
i	Pass only the identity context fields in the MQMD from the messages on the input queue to the messages on the output queue.
d	Context fields in the MQMD of messages on the output queue will represent the QLOAD program
n	There will be no information in the Context fields in the MQMD of messages on the output queue.

Table 3: Context options used by QLOAD

Destination File

QLOAD supports multiple destinations by allowing repeated specification of the -o parameter. Alternatively, If you want to put messages to a large number of destinations, then it may be more convenient to specify the destinations in a file and specify the filename on the -o parameter such as the following:

```
qload -m QM1 -iQ1 -o file:mydestinations.dst
```

On z/OS, the file name provided can also be an MVS file name, by using the following format:

```
qload -m MQG1 -iQ1 -o file:"//'GEMUSER.USER.TEXT(QLIST)'"
```

Or a DD name, by using the following format:

```
qload -m MQG1 -iQ1 -o file:DD:QLIST
```

If the destination file includes further destination files, these can also use the same format.

If you want to keep all the file names in a single piece of JCL, rather than listing them as repeated -o parameters, consider putting them into an inline DD card as follows:

```
//PUTMANY  EXEC PGM=QLOAD,PARM=('-m MQG1 -i Q1 -o "file:DD:QLIST1"')
//QLIST1   DD   *
Q2
Q3
Q4
/*
```

Destination File Contents

The format of the destination file is very straight forward. For example:

```
* The file contains queue destinations used by QLOAD
QM1/Q1
QM2/Q2
QM3/Q3

* This is a comment line
MY_LOCAL_QUEUE

* You can even load other files
file:otherdests.dst
```

As you can see, the destination file can invoke further destination files.

Blank lines and any line starting with an '*' (asterisk) character are ignored. The destinations themselves are specified one per line.

Display Options

The default way that your message data is represented in the file is hex data, as follows:

```
X 000000AE080000001000000004000000044000000DF0700000000
X 000030000000004E54314D414820202020202020202020202020
```

When unloading messages from a queue to a file, or when manipulating a file later, you can specify one of the display options for your hex and ASCII data. What follows are three example excerpts from the file produced when using each of these options. The full format of the file is discussed in “Chapter 10: File Format” on page 59. The examples are all shown as post-processing of the file, but the options can also be used when unloading a queue to a file.

Combination of Hex and ASCII data

Use the following options on the command line.

```
qload -f c:\oldfile -f c:\newfile -da
```

This will produce a file that shows the hex values of your message data on the left and any displayable characters on the right.

```
X 000000AE080000001000000004000000044000000DF0700000000
<..............D.........>
X 000030000000004E54314D414820202020202020202020202020  <..0...NT1MAH
>
```

Interleaved Hex and ASCII data

Use the following options on the command line.

```
qload -f c:\oldfile -f c:\newfile -dA
```

This will produce a file that shows ASCII text whenever the data is displayable and hex values otherwise.

```
X 000000AE08000000100000000400000

S "D"
X 000000DF070000000000000

S "0"
X 000000

S "NT1MAH                                          "
```

Message Index

Use the following options on the command line.

```
qload -m QM1 -i Q1 -f c:\myfile -di
```

This will add a comment line at the beginning of each message to show the index of the message.

```
* Index 1
A RPT 0
```

This is just a comment and is useful for determining which message you are looking at in the file. If you concatenate two files together, or edit the file to remove some of the messages, then the indexing will no longer be correct. Use the following options on the command line to re-index a file after it has been edited.

```
qload -f c:\oldfile -f c:\newfile -di
```

Message Age

Use the following options on the command line.

```
qload -m QM1 -i Q1 -f stdout -dT
```

This will display the ages of the messages on the queue.

```
1. MsgId:414D51204E5450474331202020202020 06A937452000A905
           6 hours 16 minutes 51 seconds
2. MsgId:414D51204E545047433120202020202006A937452000D204
           35 minutes 26 seconds
```

If required message selection can be performed based on the ages of the messages. Please see “Message Selection : Multiple Search Strings” on *Page 54*.

Display Summary

You can obtain a summary of the contents of your queue, by using the `-ds` option. This will provide you with details of the ages of messages on your queue, including the age of the youngest and oldest message, and the average message age. It will also provide you with details of the sizes of messages on your queue, along with the size of the smallest and biggest message, and the average message size. Here is some example output to illustrate.

```
Message Ages
============

< 2  secs < 10 secs < 1   min < 1  hour < 1   day < 1  week
--------- --------- --------- --------- --------- ---------
        0         0         0         0         0        27

< 2 weeks < 4 weeks < 3 month < 6 month < 1  year > 1  year
--------- --------- --------- --------- --------- ---------
        0         0         0         0         0         2

Youngest Message : 3 Days 10 Hrs 9 Mins 1 Sec
Oldest   Message : 224 Weeks 5 Days 3 Hrs 54 Mins 58 Secs
Average  Message : 15 Weeks 6 Days 16 Hrs 20 Mins 27 Secs

Message Sizes
=============

< 100B <  1KB <  4KB <100KB <  1MB < 10MB < 50MB > 50MB
------ ------ ------ ------ ------ ------ ------ ------
     0     18     11      0      0      0      0      0

Smallest Message : 408 bytes
Biggest  Message : 2.0 KBs
Average  Message : 924 bytes

Queue Size Summary
==================
                     Current     Maximum     Full
                    ---------   ---------   -------
Messages                   29   999999999    0.0 %
Message Bytes       26.19 KBs    1.99 TBs    0.0 %
Data Bytes          36.50 KBs    1.99 TBs    0.0 %
Queue File Size          1 MB    1.99 TBs    0.0 %
```

You can add `-dC` and `-dM` to this command to see the correlation ID and message ID of the messages at the extreme ends of age and size in this report.

In the “Queue Size Summary”, if your queue manager does not support the Current Queue File Size parameter in Inquire Queue Status, either due to platform or version, or your user ID does not have the authority to issue the Inquire Queue Status command, then the last two rows will be omitted, and the Maximum value in the “Message Bytes” row will be calculated as Max Depth x Max Msg Length.

Counting messages

If you have need to count the numbers of messages that meet criteria not covered by the above summary display, you can using the counting option `-dk` (or `-DK`). You can use any filters, search strings or time based criteria that QLOAD offers, and then instead of moving/copying the messages that match to a queue or file, QLOAD will instead simply count them and tell you how many. This can run in two modes, `-dk` which is verbose enough to show you the selection criteria in effect and the summary output at the end, and `-dK` which very simply writes the count to `stdout` and outputs nothing else to the screen.

Summary Output

Once QLOAD has completed executing the command with the options you have specified, it will output a summary of what it did, for example:

```
Read    - Files    1  Messages:     1  Bytes:        15
Written - Files    0  Messages:     1  Bytes:        15
```

To suppress this summary, use the quiet option, `-q`, as follows:-

```
qload -m QM1 -o Q1 -f c:\myfile -q
```

Message Selection

When using QLOAD to move or copy messages, or even just to manipulate the file that was produced from by unloading a queue, it may be only necessary to work with some of the messages. This can be done by selecting specific messages to process in two different ways. You can either select messages by a numeric range, or you can select messages that contain a specific string.

Selection By Message Range

Indexing the file as described in "Message Index" on page 51, may be useful when working with ranges. You can choose to process a certain range of messages from the input source (this may be a queue or a file) in a number of different ways, which are shown in "Table 4: Message range options used by QLOAD".

Option	Meaning
`x`	Process message number x only.
`x..y`	Process message number x through y only
`x#y`	Process y messages starting from message x
`#y`	Process y messages starting from the beginning. **Note:** This is the same as `1#y`

Table 4: Message range options used by QLOAD

Here are examples of some of the options. To illustrate the difference between `x..y` and `x#y`, we have a file with 10 messages in it. In order to load messages 2 through 7 inclusive onto our queue we could use either of the following options on the command line.

```
qload -m QM1 -o Q1 -f c:\myfile -r2..7
```

```
qload -m QM1 -o Q1 -f c:\myfile -r2#6
```

In both cases QLOAD will let you know which messages you are processing with output similar to the following.

```
Queue Load/Unload Utility for IBM MQ
Message Selection Active
  Message indexes 2 -> 7.
```

Selection By Search String

You can choose to process only messages that contain a certain string. This string can be in ASCII, EBCDIC or hex.

Use the following options on the command line to only process those messages that contain the ASCII string 'SALES'.

```
qload -m QM1 -i Q1 -f c:\SalesFile -s SALES
```

You can also do the opposite search - that is for messages that do not contain a certain string. This can be done using the upper case version of the option, for example, to only process those messages that do not contain the EBCDIC string 'SALES'.

```
qload -m QM1 -i Q1 -f c:\SalesFile -E SALES
```

The table below lists all the search options that can be used with QLOAD.

	Search containing this string	**Search not containing this string**
ASCII Search Options	`s (lower case)`	`S (upper case)`
EBCDIC Search Options	`e (lower case)`	`E (upper case)`
Hex Search Options	`x (lower case)`	`X (upper case)`

Table 5: Search string options used by QLOAD

You can use any combination of the search string options together. If more than one option is used, all search strings must match for the message to be processed. Use the following options on the command line to only process those messages that contain the ASCII string 'SALES' and do not contain the hex string 'F0F1F2'.

```
qload -m QM1 -i Q1 -f c:\SalesFile -s SALES -X F0F1F2
```

On ASCII platforms (Linux, UNIX and Windows) use the `-s` option to search for a natively encoded string; on EBCDIC platforms (z/OS) use the -e option to search for a natively encoded string.

Multiple Search Strings

Up to 3 of each search string may be specified. If multiple strings are used then they are treated as follows:

- **Positive search strings**
  When multiple positive strings are used then all strings must be present for the search to match. For example the command ***qload -iMATCH -s LIVERPOOL -s CHELSEA*** will only return messages which contain both strings.

- **Negative search strings**
  When multiple negative strings are used then none of the strings must be present for the search to match. For example the command ***qload -IMATCH -S HOME -S DRAW*** will only return messages which contain neither string.

Selection By Time on Queue

QLOAD supports two different ways of specifying a message selection based on the time a message has been on the queue. These are 'the message age' and the 'put time-stamp'.

We will discuss these separately in the next two sub-sections.

Selection by Message Age

You can choose to process only messages older than a certain time interval using the **-T** flag. Time interval can be specified in Days, Hours and Minutes. The general format being **[days:]hours:]minutes**. The parameter can take one or two times, **-T [OlderThanTime][,YoungerThanTime].** This is clearly very useful if you want to select messages based on a relative, rather than absolute, time.

For example, here are a number of commands and their effect :-

- **Display messages older than 5 minutes.**
```
qload -m QM1 -i Q1 -fstdout -T5
```
- **Display messages younger than 5 minutes.**
```
qload -m QM1 -i Q1 -fstdout -T,5
```
- **Display messages older than 1 day but younger than 2 days.**
```
qload -m QM1 -i Q1 -fstdout -T1:0:0,2:0:0
```
- **The following command will copy messages older than 1 hour from Q1 to Q2.**
```
qload -m QM1 -i Q1 -o Q2 -T1:0
```
- **The following command will move messages older than 1 week. from Q1 to Q2**
```
qload -m QM1 -I Q1 -o Q2 -T7:0:0
```

The **-T** flag can be combined with the **-dT** flag to determine whether there are messages older than a certain elapsed time on the queue without actually printing the contents of the message.

Selection by put time-stamp

If the relative time provided by message age selection is not suitable you can also specify absolute timestamps. You can specify a 'from' time and a 'to' time. The 'from' time selects any message put after that time. The 'to' time selects messages put before that time. Each time-stamp uses a different -T parameter. The time-stamp itself can consist of just the date portion, just the time portion or both. If the date is not specified then today's date is assumed. If the time is not specified then mid-night is assumed. The time-stamp defaults to local time but can be specified in UTC time if required.

The full format of the time-stamp is : `[f | t][ u ][DD/MM/YYYY][_][HH:MM | HH:MM:SS]`

Characters	Meaning
f	Specifies a 'from' date or put at or after this time
t	Specifies a 'to' date or put before this time
u	Specifies that the time-stamp should be treated as UTC
DD/MM/YYYY	Specifies the date. If not set the today is assumed
HH:MM	Specifies the time. If not set then mid-night is assumed
HH:MM:SS	Specifies the time including seconds. If not set then mid-night is assumed

Here are a few examples of using this parameter.

- **display messages that were put today**

  ```
  qload -m QM1 -i Q1 -fstdout -Tf00:00
  ```

- **display messages that were put on New Years Day 2015**

  ```
  qload -m QM1 -i Q1 -fstdout -Tf01/01/2015 -Tt02/01/2015
  ```

- **display messages that were put before New Years Day 2015**

  ```
  qload -m QM1 -i Q1 -fstdout -Tt01/01/2015
  ```

- **display messages that were put after a particular UTC time**

  ```
  qload -m QM1 -i Q1 -fstdout -Tfu04/07/2015_10:45:33
  ```

Selection by SQL92 Selector

IBM MQ 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[11]. 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**

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

- **display only reply messages**

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

- **display messages that were put by a particular user[12]**

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

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

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

  ```
  qload -m QM1 -i Q1 -fstdout -H "CustomerNumber = 123"
  ```

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

  ```
  qload -m QM1 -i Q1 -fstdout
  -H "Root.MQMD.CorrelId=""0x414D51204D5138303420202020202020D973DF572000C274"""
  ```

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

  ```
  qload -m QM1 -i Q1 -fstdout
  -H "Root.MQMD.CorrelId=\"0x414D51204D5138303420202020202020D973DF572000C274\""
  ```

  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.

  ```
  qload -m QM1 -i Q1 -fstdout
  -gxc414D51204D5138303420202020202020D973DF572000C274
  ```

[11]Only available when running against a version of IBM MQ which supports selectors.
[12]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.

Purge non-selected messages

In addition to using message selection, as described above, to control which messages get copied or moved, you may also need to purge those messages that do not match the message selection criteria when moving several messages[13]. This is best illustrated by an example.

Let's start with queue Q1 containing messages with the following message data, and Q2, which is empty.

```
one potato
two potato
three potato
four
five potato
six potato
seven potato
more
```

We use the following command:

```
qload -m QM1 -I Q1 -o Q2 -s potato
```

This would leave Q1 containing the following messages

```
four
more
```

and would move the following messages to Q2

```
one potato
two potato
three potato
five potato
six potato
seven potato
```

However, if we want to ensure that when moving those messages to Q2, any messages that do not match are also removed from the source queue, Q1, then we must use the following command:

```
qload -m QM1 -I Q1 -o Q2 -s potato -p
```

[13]This option is not supported when using SQL92 Selector string since the selection is being done in the Queue Manager not in the QLOAD program.

Chapter 10. File Format

The format of the file that QLOAD uses is deliberately human-readable to allow a user to update the file to easily make changes to the messages before loading them onto a queue, perhaps on a different system. After introducing the basic format of the file, this chapter gives an example to illustrate a possible field that you may wish to update, and then provides a reference to the full file format.

The file has a free format. Spaces and blank lines are ignored unless between quotes. QLOAD will add spaces to make the file more readable, but they do not affect the message format stored in that file.

The first column contains the key for each line. This can have a number of different values, some of which we have seen already. These are listed in "Table 6: Meaning of column one symbol in file format".

Column 1 value	Meaning
S	The text shown on this line is ASCII string text
X	The text shown on this line is hex
A	The text shown on this line is an attributed in the Message Descriptor (MQMD)
N	This line represents a 'missing' MQMD definition. Is it output when -dN option is used
T	The text shown on this line is ASCII string text and is followed by a newline character
*	The text on this line is a comment and will be ignored.

Table 6: Meaning of column one symbol in file format

The second column should contain a blank; this makes the file more readable.

If the first column contained the letter 'A', then the next three characters will represent the field that is being shown. For example, 'FMT' shows that this line displays the format of the message. The full set of these field labels is listed at the end of this chapter.

Example - Changing the user ID

The message descriptor (MQMD) contains a user ID which, depending on your security settings, may be used for access control when putting a message onto a queue. You may wish to change this user ID before reloading the messages onto a queue on a different system because the IDs on that system use a different naming convention. Before loading the queue from your file of messages, you would edit the file changing the field identified as USR to your desired user ID.

```
:
A RTM MQ24
A USR HUGHSON
A ACC 1A0FD4D8F2F4C3C8C9D5F1F9C6F7C1C3F3F00019F7AC30000000000000000000
:
```

Attribute Format Reference

The fields in the Message Descriptor (MQMD) are formatted in the file with a three character string representing the attribute name. "Table 7: Message descriptor attribute representations" lists the full set of these strings and which field they represent.

Message Descriptor attribute	File format representation
Report	RPT
MsgType	MST
Expiry	EXP
Feedback	FBD
Encoding	ENC
CodedCharSetId	CCS
Format	FMT
Priority	PRI
Persistence	PER
MsgId	MSI
CorrelId	COI
BackoutCount	BOC
ReplyToQ	RTQ
ReplyToQMgr	RTM
UserIdentifier	USR
AccountingToken	ACC
ApplIdentityData	AID
ApplIdentityData (HEX)	AIX
PutApplType	PAT
PutApplName	PAN
PutDate	PTD
PutTime	PTT
ApplOriginData	AOD
ApplOriginData (Hex)	AOX

Table 7: Message descriptor attribute representations

Recognised file formats

QLOAD recognises the above described file format, but also recognises the file format used as output from the sample browse program AMQSBCG.

Chapter 11. Migrating from previous versions

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

Changes made in Version 9.1.0

The main changes from the previous version the utility were:

1. **Reading non-QLOAD files**
   QLOAD is now capable of reading a file that was not created by QLOAD but contains a set of user created messages.

   For more information please see 'Chapter 8. Reading non-QLOAD files' on page 26.

Changes made in Version 9.0.3

The main changes from the previous version the utility were:

1. **Recovering messages from IBM MQ log files**
   QLOAD is now capable of piecing together messages from the output of the DMPMQLOG command and putting the resulting messages to an MQ queue.

   For more information please see 'Chapter 7. Recovering Messages from the IBM MQ log on page 23.

Changes made in Version 9.0.2

The main changes from the previous version the utility were:

1. **Required Rate Processing**
   QLOAD can now be used to move or inject a workload into your system at a defined rate using the -R parameter. For example **qload -iSOURCE -oTARGET -R500** will copy messages from SOURCE to TARGET at a rate of 500 messages per second.

   For more information please see 'Chapter 6. Required Rate Processing' on page 19.

Changes made in Version 9.0.1

1. **Generic unload/reload**
   It is now possible to unload all messages from a Queue Manager with a single command. For example, the command **qload -i* -f*** will unload all queues and write the messages to files, one file per queue. Similarly the command **qload -o* -f*** will reload the messages back again.

   For more information please refer to “Chapter 5. Generic Unload and Load“ on page 13.

2. **Extra filtering capability**
   You can ask QLOAD to filter based on the target queue found in a Dead Letter Queue or Transmission Header.

Changes made in Version 9.0.0

1. **File Limits**
   You can unload a queue to multiple files, specifying limits to control the size of each of the multiple files. These limits can be specified as the maximum age of the file; the maximum file size; or the maximum number of messages in the file. These limits can be combined so that the first limit reached causes a new file to be used.

2. **Multiple queue manager connections**
   If you need to move or copy messages from one queue manager to another, you can specify the -m parameter more than once, telling QLOAD to make a connection to each queue manager.

3. **Consuming from multiple queues**
   It is not possible to specify the -i and -I parameters multiple times if you wish to copy/move messages off multiple queues.

4. **Null file destination**
   QLOAD can unload a queue to a null file destination, thus discarding the messages from the queue**.**

Changes made in Version 8.0.2

1. **Multiple Target Queues**
   It is now possible to target multiple MQ queues and these can now be qualified by Queue Manager name. This means that QLOAD can now be used as a queue replicator.

2. **Destination File**
   If you have a large number of queues you wish to distribute messages to it can be more convenient to put the names of the queues in a file and just refer to that file. This is now possible in QLOAD with the Destination file.

3. **New Verbose Options**
   QLOAD can now print out periodic status messages to inform the user how far it has got through processing. Other verbose options include printing out the MQI verbs being used, and the list of destinations.

Changes made in Version 8.0.1

The main changes from the previous version the utility were:

1. **Built on older versions of Unix**
   It should now be possible to run the latest version of QLOAD, with all it's features, on older versions of Unix.

2. **Multi-version Support**
   QLOAD will load the MQ libraries from the place identified by **setmqenv**.

3. **Allow Message Selection based on a time-stamp**
   For a long time QLOAD has enabled you to select messages based on age. However, now you can select based on an absolute time-stamp. So, you can now easily select those messages put 'last week', or yesterday.

4. **Allow Message Selection based on Message Size**
   Have you ever wanted to get rid of those 'large' messages? Well now it's easy.

5. **Allow Message Selection based on Message Priority**
   You can now easily remove all the low or high priority messages from a queue.

6. **Allow Message Selection based on SQL92 Selection String**
   The IBM MQ product allows messages to be got based on an SQL92 selection string that allows you to write sophisticated selection criteria. QLOAD now allows you to use these selection strings.

7. **Client Performance options**
   You can now explicitly state that you wish QLOAD to use Read Ahead or Async Put. This can increase the speed of QLOAD quite significantly when run over client connections.

8. **New help features**
   To make it easier to find the option you are looking for

9. **General Bug fixes**
   Sadly software bugs are a fact of life, this version has fixed all those reported to me.

End of document

MQGem Software Limited

www.mqgem.com