

---

# WebSphere MQ Solution Designer certification exam 996 prep, Part 2: Installation and configuration

Skill Level: Intermediate

[Willy Farrell \(willyf@us.ibm.com\)](mailto:willyf@us.ibm.com)

Senior Software Engineer

IBM

03 Oct 2006

Prepare for the [IBM® Certification Test 996, WebSphere® MQ Solution Designer](#). This tutorial covers installation of WebSphere MQ and configuration of typical capabilities and features. It also explores troubleshooting techniques for problems that may occur. It is the second tutorial in a [series](#) of five tutorials on WebSphere MQ Solution Designer.

## Section 1. Before you start

### About this series

WebSphere MQ Version 6.0 connects applications in a consistent, reliable, and easy-to-manage way, providing a trustworthy foundation for cross department, enterprise wide integration. Providing reliable once-and-only-once delivery of important messages and transactions, WebSphere MQ handles the complexities of communication protocols and dynamically distributes messaging workload across available resources. This series of five tutorials helps you prepare to take the IBM certification Test 996, IBM WebSphere MQ V6.0, Solution Design. This certification targets intermediate level designers who understand the concepts of asynchronous messaging and can plan, architect, and design solutions based on WebSphere MQ.

## About this tutorial

This tutorial is the second in the series designed to help you prepare for the IBM Certification Test 996, WebSphere MQ V6.0, Solution Design. This tutorial addresses installation and configuration of WebSphere MQ. After you complete this tutorial, continue with the third tutorial, which covers distributed queue management.

## Objectives

After completing this tutorial, you will be familiar with:

- Planning an implementation of WebSphere MQ.
- WebSphere MQ administration interfaces.
- Configuring and administering a queue manager.
- Customization and administration tasks.
- Problem determination techniques.
- Restart/recovery procedures.

## Prerequisites

This tutorial is written for developers and architects with intermediate experience in application and solution design and implementation. It assumes intermediate knowledge and skills in:

- Transaction management and database products
- Systems management
- Basic programming concepts
- Data communications and networking
- Information technology security concepts

## System requirements

The examples in this tutorial were developed with [WebSphere MQ V6.0 for Windows®](#) and [Rational® Application Developer v6.0 for Windows](#).

The system requirements for the products used in the tutorial can be found through

the following links:

- [WebSphere MQ](#)
  - [Rational Application Developer](#)
- 

## Section 2. Planning a WebSphere MQ implementation

This section discusses what to consider, and decisions that must be made, when planning to implement WebSphere MQ.

### Naming WebSphere MQ objects

Naming the objects (queue managers, queues, and so forth) in a WebSphere MQ installation is one of the first things you should address in your planning activities. Agreement on names and naming conventions should be reached before beginning your implementation.

The limitations for WebSphere MQ names are:

- Only the following characters are allowed: A-Z, a-z, 0-9, . (period), /, \_ (underscore), %
- A maximum of 48 characters are allowed for the names of:
  - Queue managers
  - Queues
  - Processes
- A maximum of 20 characters are allowed for the names of channels.

The names of WebSphere MQ objects do not imply any structure to the object. All names in WebSphere MQ *are case-sensitive*.

### Queue manager

After installation, the first WebSphere MQ object to be created is a queue manager. Typically, you need to create only one queue manager per machine, but you can create others (for testing purposes, for example).

Each queue manager has a name that should be unique within a network of queue managers exchanging messages with each other. A queue manager uses the first 12 characters of its name as part of the unique message identifier when the queue manager generates the identifier.

Queue manager names are typically short. They are often given the same name as the TCP/IP host name, or the same name as the Windows system name, or the same name as an SNA LU alias, or some other name that signifies a relationship to a particular machine.

## Queues

There are some useful conventions for naming queues:

- The name of a queue should not contain an indication of its type or location. In this way, if a queue changes from being a local queue to a remote queue, for example, you can still use the same name for the queue and applications referencing the queue will require no change. Instead, the name of queue should describe its function.
- Using a common prefix for the names of related queues can aid administration. For example, allowing a search to find all queues related to a particular application.

## Special local queues

There are some local queues that have special purposes in WebSphere MQ:

### **Dead letter queue**

A designated queue upon which a queue manager will put messages that cannot otherwise be delivered. It is not mandatory for a queue manager to have a dead letter queue, but it is strongly recommended.

### **Initiation queue**

Used to implement triggering. You'll see more detail on [triggering](#) later.

### **Transmission queue**

As you saw in Part 1 of this series, a transmission queue works with message channels to enable queue manager-to-queue manager communication. The `Usage` attribute indicates that a local queue is used as a transmission queue.

### **Command queue**

Receives WebSphere MQ commands from an administration application running locally or remotely.

### Event queue

When a queue manager detects an instrumentation event, which is some significant occurrence in a queue manager, such as an error or a warning, it puts an event message describing the event on an event queue. An event queue can be monitored by a system management application that can get the event message and take appropriate action.

### Default queues

Identify the default values of attributes of any new queue that is created. There is one default queue for each of the four types of queues: local, alias, remote, and model. Thus, you only need to include in the definition of a queue those attributes whose values are different from the default values. You can change the default value of an attribute simply by redefining the appropriate default queue.

## Message channels

You learned in Part 1 that a *message channel* is a one way link between two queue managers for the transmission of messages. It consists of a *message channel agent* (MCA) at the sending end, an MCA at the receiving end, and a communications protocol between the two.

Each end of a message channel has a separate definition. Both definitions contain the name of the message channel. Among other things, the definition at each end indicates whether it is the sending or receiving end of the channel, and the communication protocol to be used.

A transmission queue is required for each message channel, but is actually located at the sending end of the channel. As a result, only the definition of the message channel at the sending end contains the name of the transmission queue. A common practice is to name the transmission queue the same name as the destination queue manager.

---

## Section 3. WebSphere MQ administration interfaces

You can administer WebSphere MQ through a variety of interfaces, described in this section.

### WebSphere MQ Script commands (MQSC)

MQSC provides a scripting interface for executing configuration commands against a queue manager. There are two modes of entering MQSC commands:

- Interactively, by typing an MQSC command at the keyboard and waiting for a result.
- Creating a file containing a sequence of MQSC commands and submitting the file for execution.

The WebSphere MQSCs are documented in the *WebSphere MQ Script (MQSC) Command Reference*, which can be found in the WebSphere MQ library (see [Resources](#)).

### **Programmable Command Format (PCF) commands**

An application can construct a message containing a PCF command and put it on a *command* queue of a queue manager. The message is retrieved by the *command server* of the queue manager, the PCF command in the message is executed, and the reply is put on the specified reply-to queue. The existence of a command queue and a command server in a network of queue managers enables each queue manager to be managed from just one system in the network, providing a *single point of control*.

PCF commands have a highly structured format and contain binary and character information. The structured format makes it easier for an application to generate the components of a PCF command dynamically. A reply to a PCF command has a similar format, which makes it easier for an application to parse.

PCF commands are documented in the *WebSphere MQ Programmable Command Formats and Administration Interface* manual, available in the WebSphere MQ library (see [Resources](#)).

### **Control commands**

Control commands are entered at a command prompt of the operating system, or they can be included in an operating system command file, such as a shell script on a UNIX system.

Control commands are described in the *WebSphere MQ System Administration Guide* in the WebSphere MQ library (see [Resources](#)).

### **OS/400 control language (CL) commands**

The WebSphere MQ CL commands are only supported on WebSphere MQ for iSeries, and can be entered anywhere an OS/400 CL command can be entered.

The WebSphere MQ CL commands are documented in the *WebSphere MQ for*

*iSeries V6.0 System Administration Guide* in the WebSphere MQ library (see [Resources](#)).

## Event messages

When a queue manager detects an instrumentation event during its operation, it puts an event message on an event queue. The event message contains information about the event that has occurred.

Event messages have a format similar to PCF commands and are therefore intended to be processed by applications. An event queue can be monitored by a system management application, which can get each message put on the queue and take the appropriate action. For example, by putting a message containing a PCF command on the command queue of the queue manager.

## WebSphere MQ Explorer

WebSphere MQ provides a graphical user interface (GUI) for administrative tasks on Windows and Linux platforms called the WebSphere MQ Explorer. This application is built on the Eclipse platform, and is provided as an alternative to control commands or MQSC commands. The *WebSphere MQ System Administration Guide* has details on using WebSphere MQ Explorer (see [Resources](#)).

## Installation

Installing WebSphere MQ on a particular platform is like installing any other software on the same platform. Always follow the instructions in the *Quick Beginnings Guide* for the platform or the *System Setup Guide* for z/OS.

Pay particular attention to instructions about what to do before installation. Some systems require certain user ID and user groups to be created. Also pay attention to instructions about what to do before using WebSphere MQ. Some systems require certain system parameters to be changed to support WebSphere MQ.

---

## Section 4. Configuring a queue manager

Once you have installed WebSphere MQ, create, configure, and control a queue manager.

### Creating a queue manager

The control command to create a queue manager is `crtmqm`. The name of the queue manager is a required parameter. Some optional parameters of the `crtmqm` command are described in Table 1.

**Table 1. Optional parameters of `crtmqm`**

<code>-q</code>	Specifies that this queue manager is to be made the default queue manager.
<code>-lc</code>	Circular logging is to be used. This is the default logging method.
<code>-ll</code>	Linear logging is to be used. Linear logging is needed for recovery from media failures.
<code>-lf LogFileSize</code>	The size of each of the log files expressed as a multiple of 4KB.
<code>-ld LogPath</code>	The directory to be used to hold the log files.

All system and default objects are created for the queue manager when you execute the `crtmqm` command.

The control command to delete a queue manager is:

```
dltmqm QMgrName
```

where *QMgrName* is the name of the queue manager to be deleted.

You must have the proper authority to create or delete a queue manager.

## Starting a queue manager

A queue manager has to be started before applications can connect to it and before any commands can be issued to it. To start a queue manager use the control command:

```
strmqm QMgrName
```

## WebSphere MQ MQSC commands

Before looking at MQSC commands in detail, you should know some of the more important syntax rules for writing MQSC commands:



- Each command starts on a new line.
- The names of the commands and their keywords are not case sensitive.
- A blank line, and a line starting with an asterisk (\*), are ignored.
- If the last non-blank character on a line is:
  - A minus sign (-), the command is continued from the start of the next line.
  - A plus sign (+), the command is continued from the first non-blank character in the next line.
- An MQSC command may contain a string of characters. The rules for using strings are:
  - A string containing blanks, lower case characters, or special characters other than those valid in the name of a WebSphere MQ object must be enclosed in single quotation marks. Lower case characters not enclosed in single quotation marks are converted to upper case.
  - A string containing no characters is not valid.

## Running MQSC commands

The administration interface for entering MQSC commands is the control command `runmqsc`. The input to `runmqsc` is zero, one, or more MQSC commands, and the output is the results of executing those commands, including operator and error messages.

`runmqsc` reads from the *standard input device*, also called *stdin*, and writes to the *standard output device*, also known as *stdout*. Typically, the standard input device is the keyboard and the standard output device is the display. However, by using redirection operators, input can be taken from a file and output can be directed to a file.

In this way, you can enter MQSC commands interactively at the keyboard and see the results on the display. Alternatively, you can create a file containing a sequence of MQSC commands and then have them executed with the results directed to a file. You can even mix the two approaches.

It is useful to maintain MQSC command files:

- For replicating a queue manager configuration on multiple systems

- To recover a queue manager configuration
- When going through a number of iterations in testing a queue manager configuration

## Defining a local queue

The MQSC command to define a local queue is `DEFINE QLOCAL`. You can also use the synonym `DEF QL`. Let's look at an example:

```
DEFINE QLOCAL(MY_QUEUE) +
DESCR("This is a test queue")
REPLACE +
PUT(ENABLED) +
GET(ENABLED) +
MAXDEPTH(1000)
```

This command defines a local queue named `MY_QUEUE`, with a description of "This is a test queue" that is enabled for both `PUT`ting and `GET`ting messages, and that can contain a maximum of 1000 messages. The value of any attributes not explicitly defined in the definition of the queue are taken from the corresponding attributes of the default local queue, `SYSTEM.DEFAULT.LOCAL.QUEUE`.

The `REPLACE` keyword indicates that if this queue already exists, then the definition of the existing queue is replaced with this new definition. Any messages on the queue are retained. Let's look at another example:

```
DEF QL(ANOTHER_QUEUE) LIKE(MY_QUEUE)
```

This command uses the synonym to define a local queue. The `LIKE` keyword indicates that the default attributes for the queue should come from `MY_QUEUE` rather than the default local queue.

## Displaying attributes

The MQSC command to display the attributes of a queue is `DISPLAY QUEUE`. The synonym is `DIS Q`. `DISPLAY QUEUE` applies to all types of queues: local, alias, remote, and model.

The command:

```
DISPLAY QUEUE(MY_QUEUE)
```

---

displays all of the attributes of MY\_QUEUE. Use the ALL keyword to display all attributes.

To indicate only those attributes you want displayed:

```
DIS Q(ANOTHER_QUEUE) DESCR MAXDEPTH
```

DESCR and MAXDEPTH displays the attributes of ANOTHER\_QUEUE.

A trailing asterisk (\*) can be used as a wild card character. For example:

```
DIS Q(SYSTEM*)
```

displays the attributes of all queues beginning with SYSTEM. An asterisk on its own specifies all queues.

The MQSC command to display the attributes of a queue manager is DISPLAY QMGR. Its synonym is DIS QMGR.

## Defining other queue types

### Alias queues

An *alias* queue is a WebSphere MQ object that is used to refer indirectly to another queue. The MQSC command to create an alias queue is DEFINE QALIAS. Its synonym is DEF QA. This command has a TARGQ keyword that indicates the name of the queue to which the alias queue resolves.

The command:

```
DEFINE QALIAS(ALIAS_QUEUE) TARGQ(MY_QUEUE)
```

defines an alias queue named ALIAS\_QUEUE that resolves to the queue named MY\_QUEUE.

### Local definition of a remote queue

A *local definition of a remote queue*, or remote queue, is a WebSphere MQ object owned by one queue manager that refers to a queue owned by another queue manager. The MQSC command to create a local definition of a remote queue is DEFINE QREMOTE. Its synonym is DEF QR. This command has a keyword RNAME

that specifies the name of the queue on the remote queue manager. Another keyword, `RQMNAME`, specifies the name of the remote queue manager.

The command:

\

```
DEFINE QREMOTE(REMOTE_QUEUE) RNAME(A_QUEUE) RQMNAME(QM2)
```

defines a remote queue named `REMOTE_QUEUE` that refers to a queue named `A_QUEUE` on a queue manager named `QM2`.

## Model queues

A *model* queue is a WebSphere MQ object whose attributes are used as a template for creating a *dynamic* queue. When an application opens a model queue, the queue manager creates a dynamic queue. The MQSC command to create a model queue is `DEFINE QMODEL`. Its synonym is `DEF QM`. This command has a keyword `DEFTYPE` that specifies whether a dynamic queue created from the model queue is:

- A *temporary* dynamic queue, which is deleted when it is closed and does not survive a queue manager restart (`TEMPDYN`), or
- A *permanent* dynamic queue, whose deletion upon being closed is optional and which does survive a queue manager restart (`PERMDYN`).

The command:

```
DEFINE QMODEL(MY_MODEL_QUEUE) DEFTYPE(TEMPDYN)
```

creates a model queue named `MY_MODEL_QUEUE` which, when opened by an application, creates a temporary dynamic queue.

## More MQSC commands

### Altering attributes

To change the attributes of a queue, a form of the `ALTER` command is used:

- `ALTER QALIAS` (synonym `ALT QA`) alters the attributes of an alias queue
- `ALTER QLOCAL` (synonym `ALT QL`) alters the attributes of a local queue
- `ALTER QMODEL` (synonym `ALT QM`) alters the attributes of a model queue

- `ALTER QREMOTE` (synonym `ALT QR`) alters the attributes of a remote queue

To change the attributes of a queue manager, the `ALTER QMGR` is used. Its synonym is `ALT QMGR`.

Only the attributes specified to be changed in the `ALTER` command are modified; all other attributes remain unchanged.

## Deleting a queue

To delete a queue, a form of the `DELETE` command is used:

- `DELETE QALIAS` (synonym `DELETE QA`) deletes an alias queue
- `DELETE QLOCAL` (synonym `DELETE QL`) deletes a local queue
- `DELETE QMODEL` (synonym `DELETE QM`) deletes a model queue
- `DELETE QREMOTE` (synonym `DELETE QR`) deletes a remote queue

A queue cannot be deleted that is still in use (open by an application).

## Delete all messages on a queue

To clear all of the messages from a local queue that is not in use, the `CLEAR LOCAL` command is used. Its synonym is `CLEAR QL`.

The command:

```
CLEAR LOCAL(MY_QUEUE)
```

clears all of the messages from the queue `MY_QUEUE`, given that the queue is not in use.

## Sample programs

WebSphere MQ ships with a number of sample programs, written in C, to accomplish various simple tasks. These programs are quite useful for testing the configuration of your WebSphere MQ installation.

The sample programs most commonly used are:

<code>amqsput</code>	Reads lines of text from the standard input device, converts them to messages, and
----------------------	--

	puts the messages on the named queue.
amqsget	Gets messages from the named queue, and writes the text within each message to the standard output device.
amgsbcg	Browse the messages on the named queue and writes their contents, in both hex and character format, to the standard output device. It also displays, in a readable format, the message descriptor of each message.

The sample programs take a queue name as the first mandatory parameter, followed by a second optional parameter naming the queue manager.

## Stopping a queue manager

The control command to end the operation of a queue manager is `endmqm`. The command typically takes two parameters: the first parameter indicates the mode of shutdown, and the second indicates the queue manager to be stopped.

The shutdown modes are:

### **Controlled (or quiesced) -c**

The queue manager stops only after all applications have disconnected. All new requests to connect to the queue manager fail. This is the default mode.

### **Immediate -i**

The queue manager stops after it has completed all the MQI calls currently being processed. Any MQI calls issued after the command has been entered fail. Any incomplete units of work are rolled back when the queue manager is next started.

### **Controlled with wait -w**

Stops the queue manager in the same manner as the controlled option, but the command prompt does not return until the queue manager has ended.

### **Preemptive -p**

The queue manager stops without waiting for applications to disconnect or for MQI calls to complete. Using this mode can lead to unpredictable results.

The normal mode of stopping a queue manager is the controlled, or quiesced, mode. In this mode, applications can continue to do work, but *well-behaved* applications will disconnect as soon as convenient. To detect that a queue manager is quiescing, a well-behaved application should use the *fail if quiescing* option on the MQOPEN, MQPUT, MQPUT1, and MQGET calls. The use of this option means that the call

fails if the queue manager is quiescing.

Once an application has detected that a queue manager is quiescing, it should terminate cleanly. In doing this, it may possibly issue further calls that ignore the quiescing state. It may either complete and commit a unit of work in progress, or it may back it out, before terminating.

The immediate mode of stopping a queue manager should only be used if you need to stop it quickly with predictable results.

The preemptive mode should only be used as a last resort.

## Working with queues (a hands-on exercise)

If you are going to perform these examples on your own system, install WebSphere MQ (see [Resources](#) for a link to a trial version of WebSphere MQ) but do not use the default install directory. Instead, perform a custom install and change the install directory to C:\WSMQ\. If you install to a directory other than C:\WSMQ\, replace your install directory with C:\WSMQ\ in the instructions in this tutorial. Also, do **NOT** set up the default configuration when prompted, once install is complete.

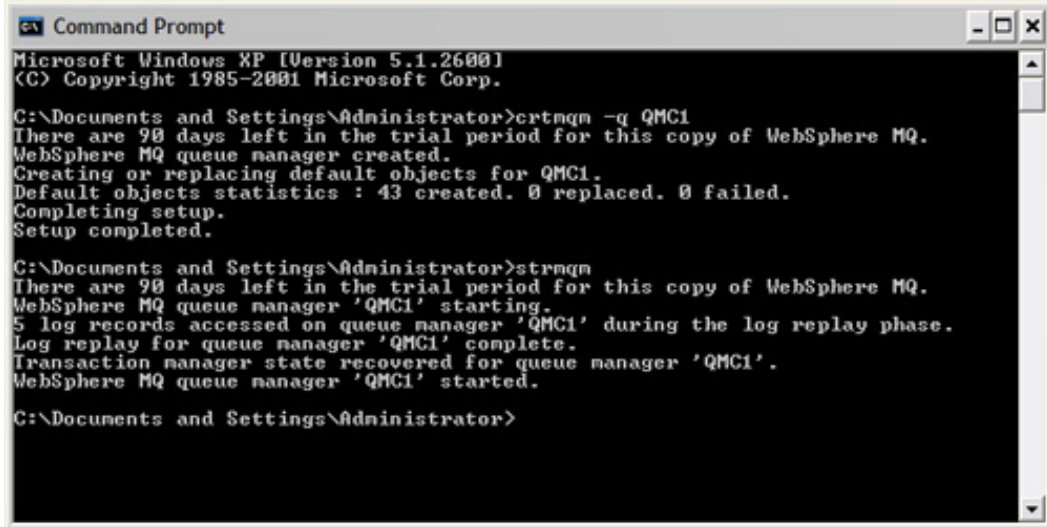
If you are using WebSphere MQ on another platform, consult the documentation for your specific platform for corresponding commands to those given here.

### Create and start a queue manager

We'll create a default queue manager and start its operation.

1. Open a command prompt from the Windows Start menu.
2. Type `crtmqm -q QMC1` (the queue manager name is case sensitive), and press **Enter**.
3. Type `strmqm` (because it is the default queue manager, there is no need to supply the queue manager name), and press **Enter**. Your screen should look like Figure 1.

**Figure 1. Creating and starting a queue manager**



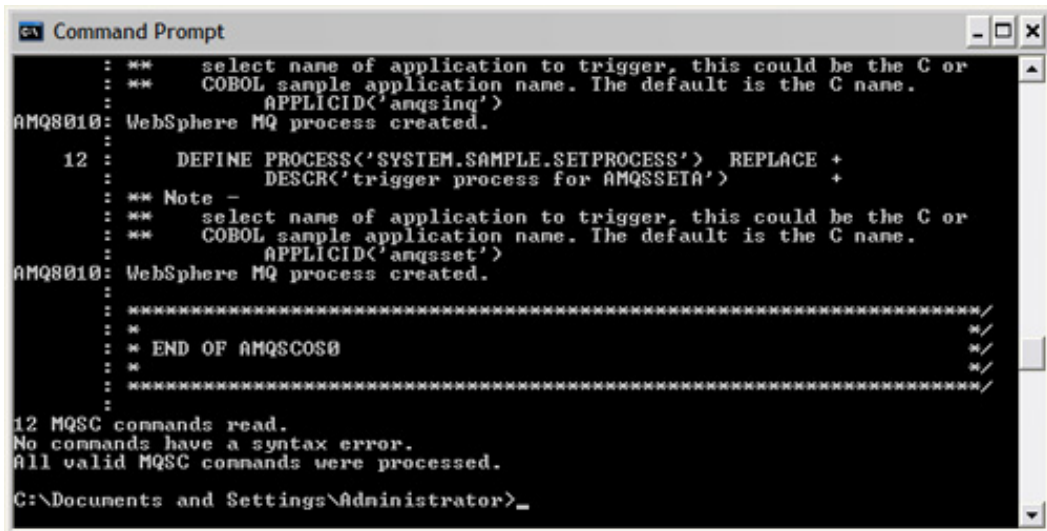
4. Type `cls` and press **Enter** to clear the screen.
5. Type

```
runmqsc QMC1 < \WSMQ\tools\mqsc\samples\amqscos0.tst
```

and press **Enter**. This creates sample WebSphere MQ objects.

6. Make sure all twelve commands completed successfully, as shown in Figure 2.

**Figure 2. Creating sample objects**



7. Type `cls` and press **Enter** to clear the screen.

### Use MQSC commands interactively



Now we can use MQSC commands to display, create, and alter WebSphere MQ objects.

1. Type `runmqsc` and press **Enter**. This puts you in interactive mode.
2. Type `display qmgr` and press **Enter** to display all of the attributes of the queue manager. Your screen should look like Figure 3.

**Figure 3. Displaying queue manager attributes**

```

Command Prompt - runmqsc
CLWLLEN(100)
CLWLSEQ(LOCAL)
COMMANDQ(SYSTEM.ADMIN.COMMAND.QUEUE)
CRTIME(08.24.28)
DEFXMITQ( )
DISTL(YES)
IPADDRU(IPU4)
LOGGEREU(DISABLED)
MAXMSGL(4194304)
MAXUMSGS(10000)
MONCHL(OFF)
PERFMEU(DISABLED)
QMI D(QMCI_2006-09-17_08.24.28)
REPOS( )
ROUTEREC(MSG)
SCMSERU(QMGR)
SSLCRYP( )
SSLFIPS(NO)
SSLKEYR(C:\Program Files\IBM\WebSphere MQ\qmgrs\QMCI\ssl\key)
SSLKEYC(0)
STATCHL(OFF)
STATMQ(OFF)
STRSTPEU(ENABLED)
TRIGINT(999999999)
CLWLMRUC(999999999)
CMDLEVEL(600)
CRDATE(2006-09-17)
DEADQ( )
DESCR( )
INHIBTEU(DISABLED)
LOCALEU(DISABLED)
MAXHANDS(256)
MAXPRIV(9)
MONACLS(QMGR)
MONQ(OFF)
PLATFORM(WINDOWSNT)
REMOTEEU(DISABLED)
REPOSNL( )
SCHINIT(QMGR)
SSLCRLNL( )
SSLEU(DISABLED)
STATACL(QMGR)
STATINT(1800)
STATQ(OFF)
SYNCP
  
```

3. Type `display q(SYSTEM*)` and press **Enter**. This displays all of the queues beginning with "SYSTEM." Check your screen against Figure 4.

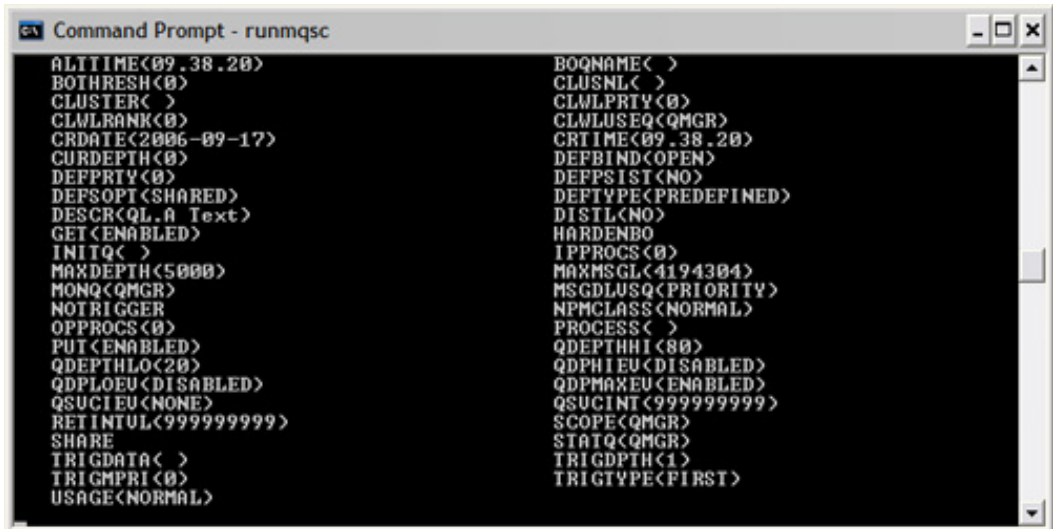
**Figure 4. Displaying all SYSTEM queues**

```

Command Prompt - runmqsc
AMQ8409: Display Queue details.
  QUEUE(SYSTEM.DEFAULT.REMOTE.QUEUE)      TYPE(QREMOTE)
AMQ8409: Display Queue details.
  QUEUE(SYSTEM.MQEXPLORER.REPLY.MODEL)     TYPE(QMODEL)
AMQ8409: Display Queue details.
  QUEUE(SYSTEM.MQSC.REPLY.QUEUE)          TYPE(QMODEL)
AMQ8409: Display Queue details.
  QUEUE(SYSTEM.PENDING.DATA.QUEUE)        TYPE(QLOCAL)
AMQ8409: Display Queue details.
  QUEUE(SYSTEM.SAMPLE.ALIAS)              TYPE(QALIAS)
AMQ8409: Display Queue details.
  QUEUE(SYSTEM.SAMPLE.ECHO)               TYPE(QLOCAL)
AMQ8409: Display Queue details.
  QUEUE(SYSTEM.SAMPLE.INQ)                TYPE(QLOCAL)
AMQ8409: Display Queue details.
  QUEUE(SYSTEM.SAMPLE.LOCAL)              TYPE(QLOCAL)
AMQ8409: Display Queue details.
  QUEUE(SYSTEM.SAMPLE.REMOTE)             TYPE(QREMOTE)
AMQ8409: Display Queue details.
  QUEUE(SYSTEM.SAMPLE.REPLY)              TYPE(QMODEL)
AMQ8409: Display Queue details.
  QUEUE(SYSTEM.SAMPLE.SET)                TYPE(QLOCAL)
AMQ8409: Display Queue details.
  QUEUE(SYSTEM.SAMPLE.TRIGGER)            TYPE(QLOCAL)
  
```

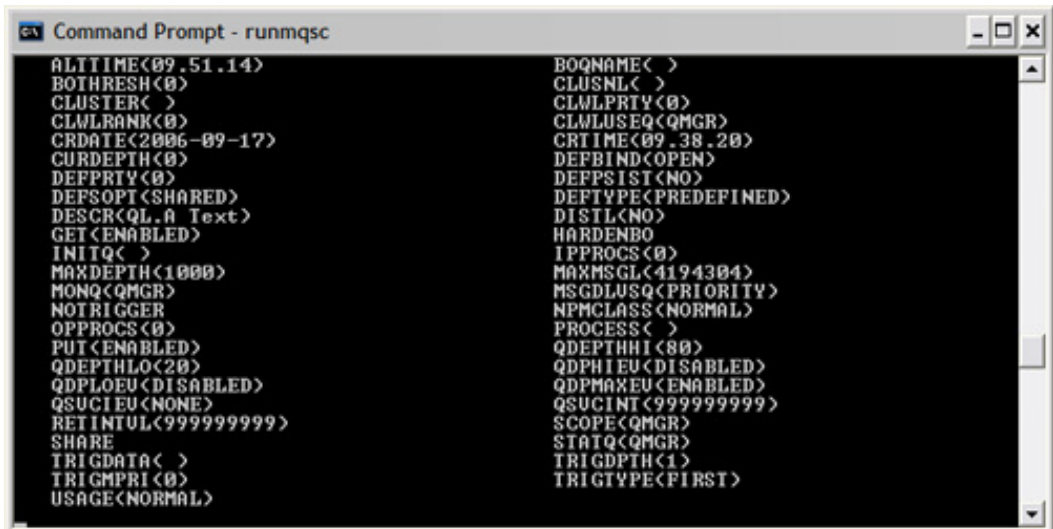
4. Type `define ql(QL.A) descr('QL.A Text')` and press **Enter**. This creates a local queue named QL.A with a text description.

5. Type `display ql(QL.A)` and press **Enter** to display all of the attributes of the queue. Compare your screen to Figure 5.
- Figure 5. Displaying queue attributes**



6. Type `alter ql(QL.A) maxdepth(1000)` and press **Enter** to change the maximum number of messages allowed on the queue to 1000.
7. Type `display ql(QL.A)` and press **Enter** to display the queue attributes again to ensure the modification was successful. Notice that no other attributes were changed, as shown in Figure 6.

**Figure 6. Displaying queue attributes after alter**



8. Type `define ql(QL.B) descr('QL.B Text')` and press **Enter** to define a second queue with a text description.

9. Type `display ql(QL.B)` and press **Enter** to display the attributes of this second queue.
10. Type `define ql(QL.B) replace maxdepth(2000)` and press **Enter** to change the maximum number of messages allowed on the queue to 2000.
11. Type `display ql(QL.B)` and press **Enter** to display the attributes again. Notice that by replacing the old definition, the `DESCR` attribute was changed to its default value (empty string), as shown in Figure 7.

**Figure 7. Displaying queue attributes after replace**

```

Command Prompt - runmqsc
ALTIME<10.00.20>          BOQNAME< >
BOTHRESH<0>              CLUSNL< >
CLUSTER< >               CLWLPRTY<0>
CLWLANK<0>               CLWLUSEQ<QMGR>
CRDATE<2006-09-17>      CRTIME<09.59.56>
CURDEPTH<0>             DEFBIND<OPEN>
DEFPRTY<0>              DEFPST<NO>
DEFSOPT<SHARED>        DEFTYPE<PREDEFINED>
DESCR< >                 DISTL<NO>
GET<ENABLED>           HARDENBO
INITQ< >                 IPPROCS<0>
MAXDEPTH<2000>         MAXMSGL<4194304>
MONQ<QMGR>             MSGDLUSQ<PRIORITY>
NOTRIGGER               NPMCLASS<NORMAL>
OPPOCS<0>              PROCESS< >
PUT<ENABLED>           QDEPTHHI<80>
QDEPTHLO<20>          QDPHIEU<DISABLED>
QDPLOEU<DISABLED>     QDPMAXEU<ENABLED>
QSUCIEU<NONE>         QSUCINT<999999999>
RETINTUL<999999999>   SCOPE<QMGR>
SHARE                  STATQ<QMGR>
TRIGDATA< >           TRIGDPTH<1>
TRIGMPRI<0>          TRIGTYPE<FIRST>
USAGE<NORMAL>

```

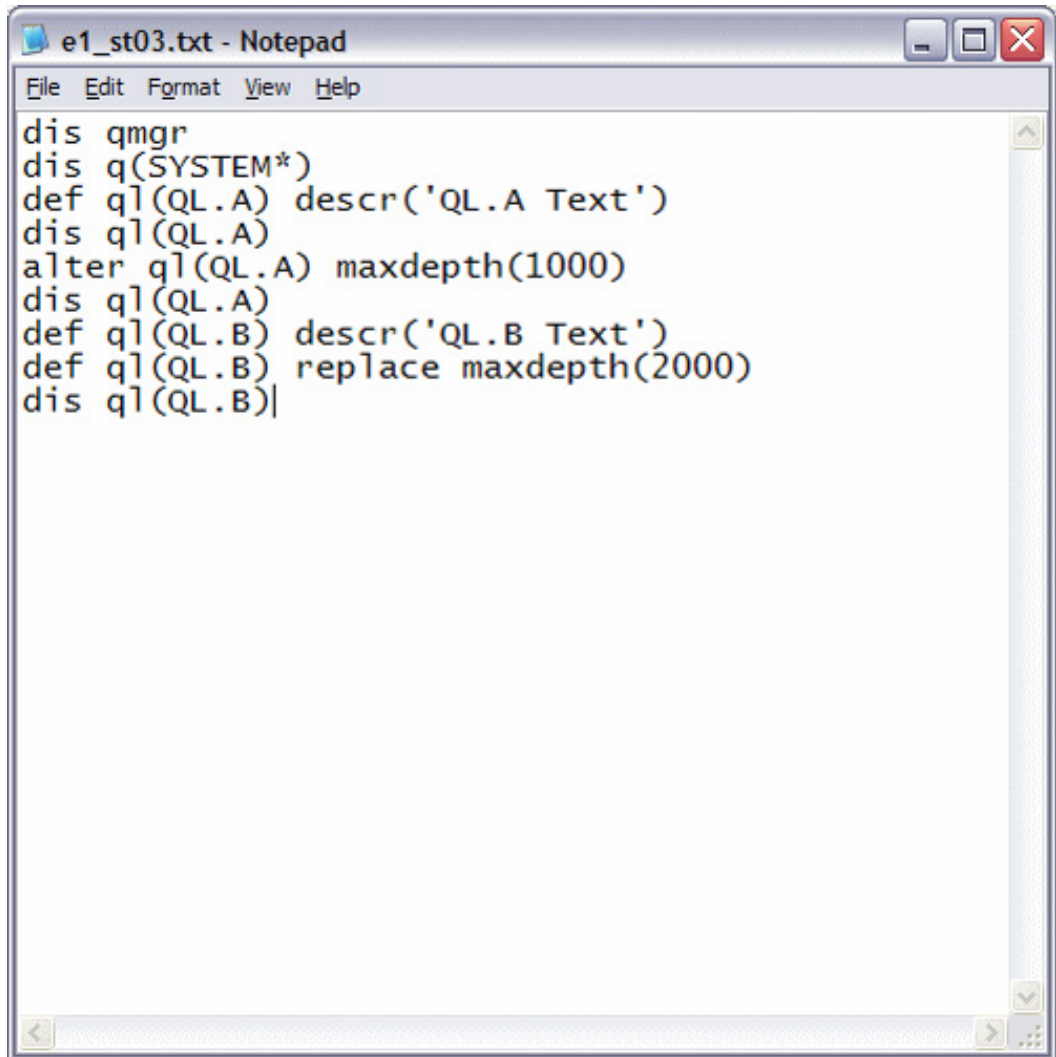
12. Type `delete ql(QL.A)` and press **Enter** to delete the QL.A queue.
13. Type `delete ql(QL.B)` and press **Enter** to delete the QL.B queue.
14. Type `end` and press **Enter** to exit runmqsc interactive mode.
15. Type `cls` and press **Enter** to clear the screen.

### Use MQSC commands with a command file

Now you'll create a command file with the same commands you entered interactively above. You can then run the command file and view the results in an output file.

1. User a text editor to create a file with the list of commands entered above. Name the file `e1_st03.txt`. I used Notepad, and my file looks like Figure 8.

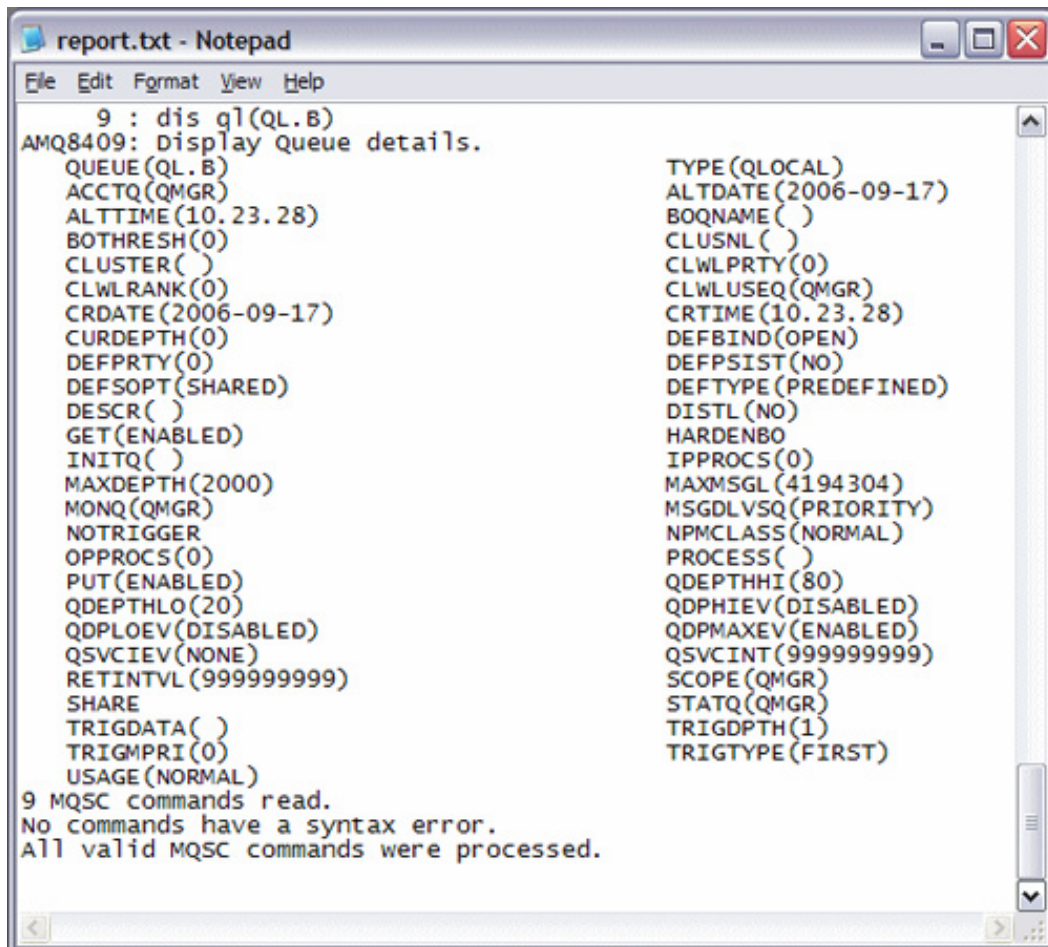
**Figure 8. MQSC command file e1\_st03.txt**

A screenshot of a Notepad window titled "e1\_st03.txt - Notepad". The window has a menu bar with "File", "Edit", "Format", "View", and "Help". The text content of the file is as follows:

```
dis qmgr
dis q(SYSTEM*)
def ql(QL.A) descr('QL.A Text')
dis ql(QL.A)
alter ql(QL.A) maxdepth(1000)
dis ql(QL.A)
def ql(QL.B) descr('QL.B Text')
def ql(QL.B) replace maxdepth(2000)
dis ql(QL.B)|
```

2. Back at the command prompt, type `runmqsc < e1_st03.txt > report.txt` and press **Enter**.
3. Open `report.txt` in a text editor. Scroll to the bottom to ensure that all commands executed successfully, as shown in Figure 9.

**Figure 9. Output from running MQSC command file**



```

report.txt - Notepad
File Edit Format View Help
  9 : dis ql(QL.B)
AMQ8409: Display Queue details.
QUEUE(QL.B)
ACCTQ(QMGR)
ALTTIME(10.23.28)
BOTHRESH(0)
CLUSTER( )
CLWLRANK(0)
CRDATE(2006-09-17)
CURDEPTH(0)
DEFPRTY(0)
DEFSOPT(SHARED)
DESCR( )
GET(ENABLED)
INITQ( )
MAXDEPTH(2000)
MONQ(QMGR)
NOTRIGGER
OPPROCS(0)
PUT(ENABLED)
QDEPTHLO(20)
QDPLOEV(DISABLED)
QSVCI EV(NONE)
RETINTVL(999999999)
SHARE
TRIGDATA( )
TRIGMPRI(0)
USAGE(NORMAL)
TYPE(QLOCAL)
ALTDAT(2006-09-17)
BOQNAME( )
CLUSNL( )
CLWLPRTY(0)
CLWLUSEQ(QMGR)
CRTIME(10.23.28)
DEFBIND(OPEN)
DEFPSIST(NO)
DEFTYPE(PREDEFINED)
DISTL(NO)
HARDENBO
IPPROCS(0)
MAXMSGL(4194304)
MSGDLV SQ(PRIORITY)
NPMCLASS(NORMAL)
PROCESS( )
QDEPTHHI(80)
QDPHIEV(DISABLED)
QDPMAEV(ENABLED)
QSVCI NT(999999999)
SCOPE(QMGR)
STATQ(QMGR)
TRIGDPTH(1)
TRIGTYPE(FIRST)
9 MQSC commands read.
No commands have a syntax error.
All valid MQSC commands were processed.

```

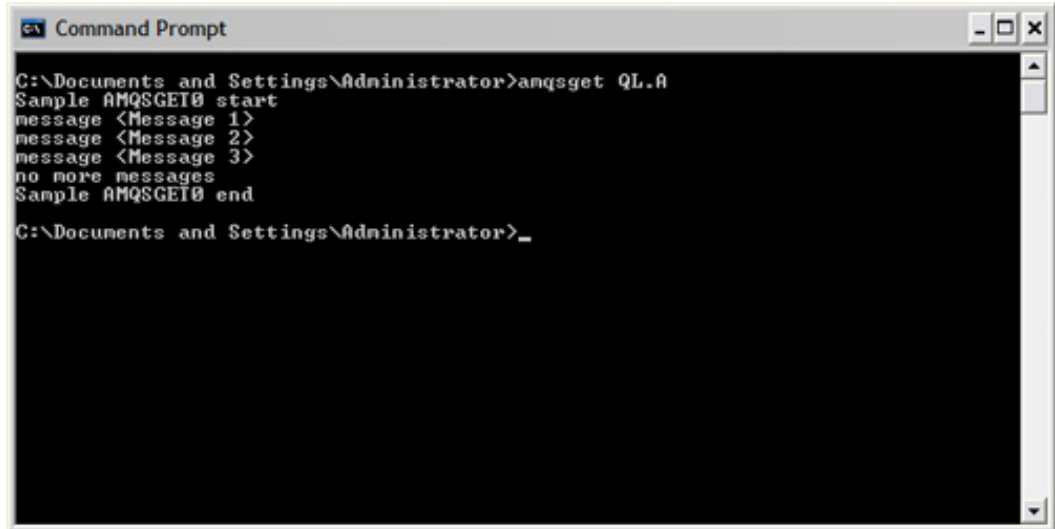
4. At the command prompt, type `cls` and press **Enter** to clear the screen.

### Use the sample programs

With your queues defined, you can now use the sample programs to put, browse, and get messages.

1. At the command prompt, type `amqspu QL.A` and press **Enter**.
2. Type `Message 1` and press **Enter**.
3. Type `Message 2` and press **Enter**.
4. Type `Message 3` and press **Enter**.
5. Press **Enter** to exit the program.
6. Type `cls` and press **Enter** to clear the screen.





```
Command Prompt
C:\Documents and Settings\Administrator>amqsget QL.A
Sample AMQSGEIB start
message <Message 1>
message <Message 2>
message <Message 3>
no more messages
Sample AMQSGEIB end
C:\Documents and Settings\Administrator>_
```

11. Type `cls` and press **Enter** to clear the screen.

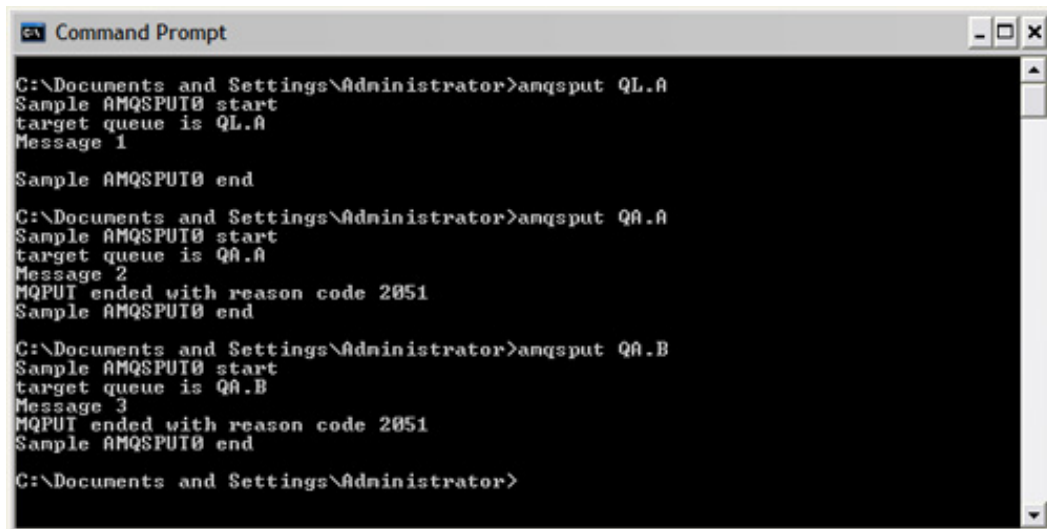
### Work with alias queues

Here you work with alias queues and see how the attributes of alias queues, and the queues they resolve to, are related.

1. At the command prompt, type `runmqsc` and press **Enter** to start the MQSC command processor.
2. Type `def qa(QA.A) targq(QL.A)` and press **Enter** to create an alias queue named QA.A, which resolves to local queue QL.A.
3. Type `alter qa(QA.A) put(disabled)` and press **Enter** to inhibit put requests on the alias queue.
4. Type `alter ql(QL.B) put(disabled)` and press **Enter** to inhibit put requests on local queue QL.B.
5. Type `def qa(QA.B) targq(QL.B)` and press **Enter** to create an alias queue named QA.B, which resolves to local queue QL.B.
6. Type `end` and press **Enter** to exit `runmqsc`.
7. Type `cls` and press **Enter** to clear the screen.
8. Type `amqsput QL.A` and press **Enter**.
9. Type `Message 1` and press **Enter**.

10. Press **Enter** to exit amqspu. Note that there were no problems putting messages on QL.A.
11. Type `amqspu QA.A` and press **Enter**.
12. Type `Message 2` and press **Enter**. The sample program ends with reason code 2051, indicating that put has been inhibited on this queue. So, even though the queue to which QA.A resolves (QL.A) can receive messages, the attributes of QA.A prevent messages being sent to QL.A through the alias queue. You can find a list of reason codes in the *WebSphere MQ Messages* manual in the WebSphere MQ library (see [Resources](#)).
13. Type `amqspu QA.B` and press **Enter**.
14. Type `Message 3` and press **Enter**. Again the sample program ends with reason code 2051. Even though QA.B is not put-inhibited, the local queue to which it resolves is, so messages cannot be put on QA.B. Your screen should look like Figure 12.

**Figure 12. Putting messages on alias and local queues**



```
Command Prompt
C:\Documents and Settings\Administrator>amqspu QL.A
Sample AMQSPUT0 start
target queue is QL.A
Message 1
Sample AMQSPUT0 end
C:\Documents and Settings\Administrator>amqspu QA.A
Sample AMQSPUT0 start
target queue is QA.A
Message 2
MQPUT ended with reason code 2051
Sample AMQSPUT0 end
C:\Documents and Settings\Administrator>amqspu QA.B
Sample AMQSPUT0 start
target queue is QA.B
Message 3
MQPUT ended with reason code 2051
Sample AMQSPUT0 end
C:\Documents and Settings\Administrator>
```

15. Type `runmqsc` and press **Enter** to start the MQSC command processor.
16. Type `clear ql(QL.A)` and press **Enter** to clear the messages from local queue QL.A.
17. Type `end` and press **Enter** to exit runmqsc.
18. Close the command prompt window.



---

## Section 5. Publish/Subscribe

This section discusses the publish/subscribe capabilities included with WebSphere MQ, and describes how to set up and manage this function.

### Overview

WebSphere MQ Publish/Subscribe releases applications from having to know anything about target applications. Essentially, a sending application sends information (*publish*) to a standard destination that is managed by WebSphere MQ Publish/Subscribe. The distribution is handled by Publish/Subscribe.

The receiving application also need only to *subscribe* to a standard location, registering interest, and then await the delivery of the information.

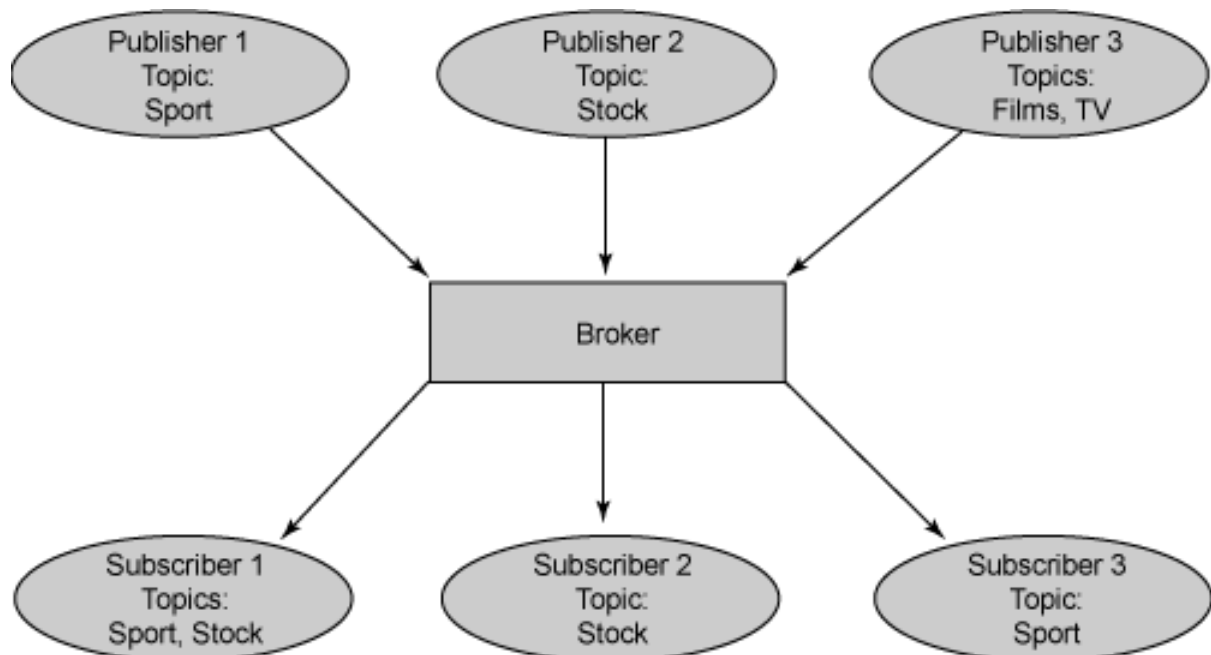
WebSphere MQ messages are the vehicle used to transport the information between publishers and subscribers. The subject of the information is called a *topic*. A publishing application specifies a topic when it sends a message. The subscriber application identifies the topic it is interested in. Only information that matches the subscriber's topics is sent.

Obviously, there is a requirement for a process to handle the proper routing of information. This is handled by a *broker*.

Topics that are related can be grouped into *streams*, allowing for fewer total items that the broker needs to manage. It also makes access control simpler. If a topic does not belong to a particular stream, the broker has a default stream.

Figure 13 shows a simple configuration depicting a news service.

### Figure 13. Simple publish/subscribe configuration



In this example, a single stream can be made up of several kinds of information:

- Publisher 1 is publishing data about sports results (Topic:Sport)
- Publisher 2 is publishing data about stock prices (Topic:Stock)
- Publisher 3 is publishing data about film reviews (Topic:Films) and television (Topic:TV)

In the bottom part of Figure 13, three subscribers have indicated their interest in different topics:

- Subscriber 1 will get information about sports results and stock prices
- Subscriber 2 will get film review information
- Subscriber 3 will receive the sports results

Since nobody has subscribed to TV, no information is distributed.

Broker configuration can become very complex with many brokers in a network. However, only one broker is allowed per queue manager.

A broker network must be arranged as a hierarchy with the top broker being the *root broker*. It will have one or more *child brokers* and is known as a *parent broker*. The child brokers may also have child brokers, forming a hierarchy. This eases the number of channels required in the network.

Brokers can exchange subscription registrations and deregistrations, publications

and requests to delete publications, and information about themselves. The broker is known by the same name as the local queue manager.

Publishers and subscribers can reside anywhere in a WebSphere MQ network, as long as there is a route from their queue manager to the broker.

Detailed information on WebSphere MQ Publish/Subscribe is in the *WebSphere MQ Publish/Subscribe User's Guide* in the WebSphere MQ library (see [Resources](#)).

## Setting up the broker

Queues required for publish/subscribe functions are automatically defined when the broker starts, if they do not already exist.

Streams are implemented as sets of queues, one at each broker that supports the particular stream. The queue at each broker for a specific stream will have the same name.

All brokers will have a default stream, using a queue called `SYSTEM.BROKER.DEFAULT.STREAM`. This queue receives all publication messages for the default stream. Administrators can create streams by setting up a series of local queues (one on each broker) with the same name. However, streams beginning with `SYSTEM.BROKER` are reserved for WebSphere MQ use.

Each broker also has a control queue (`SYSTEM.BROKER.CONTROL.QUEUE`) where all command messages are sent, except publications and requests to delete publications. It is a predefined queue that takes its properties from the `SYSTEM.DEFAULT.LOCAL.QUEUE`.

The `SYSTEM.BROKER.ADMIN.STREAM` queue is used by the broker to publish its own configuration information. It is possible to write an administration application that can use the information published on this stream.

If an administrator chooses, the stream queues can be created dynamically when they are needed. You need to ensure that a model queue called `SYSTEM.BROKER.MODEL.QUEUE` is defined. Its definition is available in the script `amqsfmda.tst`, which comes with WebSphere MQ.

Normal WebSphere MQ access control techniques apply to applications and brokers opening queues for Publish/Subscribe messages. There is no topic-based security; the access check is for the stream.

## Controlling the broker

Control of the broker is accomplished by a series of control commands specific to broker operations. Since brokers run within the context of a queue manager, you need to specify the queue manager that the broker is associated with when issuing these commands, unless it is the default queue manager.

The broker control commands are:

<code>strmqbrk</code>	Starts the broker. Can be triggered (see <a href="#">Triggering</a> ). All required queues are created if they do not exist when this command is issued.
<code>endmqbrk</code>	Stops the broker. Allows for controlled or immediate shutdown. There is no preemptive shutdown, as exists for queue managers. All control information is retained, and registrations for publishes and subscribers remain in force. Messages are simply queued for the broker until it is restarted.
<code>dspmqbrk</code>	Displays the broker status. One of the following values is returned: <ul style="list-style-type: none"> <li>• Starting</li> <li>• Running</li> <li>• Stopping (immediate shutdown)</li> <li>• Quiescing (controlled shutdown)</li> <li>• Not active</li> <li>• Ended abnormally</li> </ul>
<code>dltmqbrk</code>	Deletes the broker. To delete a broker, it must be stopped and the queue manager must be active. In a complex structure of brokers, it is dangerous to delete a broker as it might be higher up in the hierarchy than others. The delete will fail with an error message if this is the case.
<code>clrmqbrk</code>	Clears the brokers memory of a neighboring target broker. It will cancel all subscriptions from the target broker. This can only be done when the broker is stopped. This operation is required on both sides in order to stop messages from flowing.
<code>migmqbrk</code>	Migrates the broker to a WebSphere MQ Integrator broker.

## Message broker exits

The message broker exit permits customization of a publication at a broker; for

instance, causing traffic for different streams to be sent across different channels.

The exit is invoked after the broker determines that it will send a publication to a specific broker or subscriber. The exit can change the message descriptor (MQMD) values as well as the publication information itself.

The exit must be set up as part of the queue manager definition.

A parameter block is used for input/output. It contains information related to the invocation of the exits and results of the invocation. It is possible to issue MQI calls in the exit with some restrictions (for example, never use MQDISC).

One of the sample programs provided with WebSphere MQ Publish/Subscribe is a routing exit program. It may be useful to understand its operations before attempting to develop other exits.

---

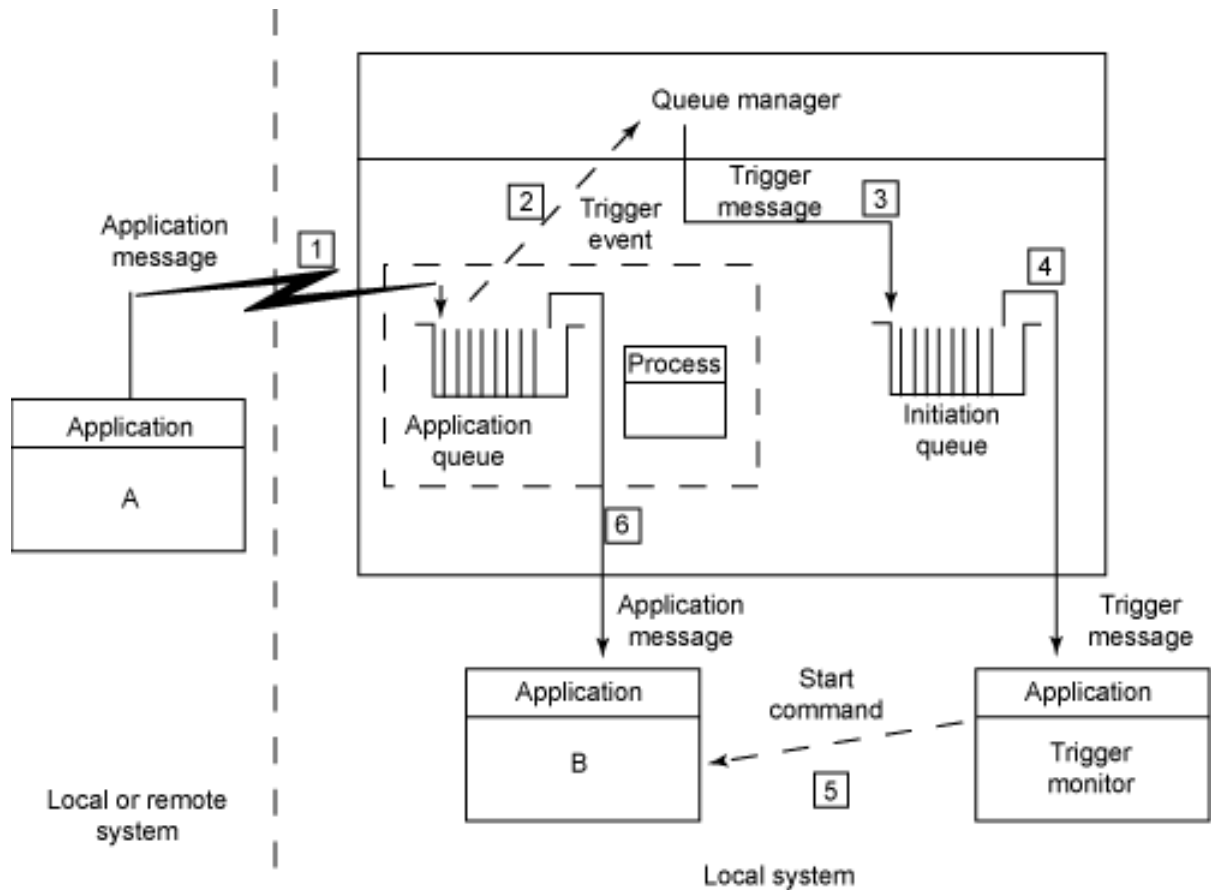
## Section 6. Triggering

An application does not have to be running when a message is sent to it -- it can start later. WebSphere MQ provides a facility that enables an application to be started automatically when there are messages available for retrieval. This facility is known as *triggering*.

### Overview

Let's start the discussion of triggering by looking at the example shown in Figure 14 and following its flow.

#### Figure 14. Triggering flow



The sequence of actions is:

1. Application A puts a message on an application queue that is enabled for triggering.
2. If the conditions for triggering are met, a *trigger event* occurs, and the queue manager examines the process object referenced by the application queue. The process object identifies the application to be started, which in this case is Application B.
3. The queue manager creates a *trigger message* whose fields contain information copied from certain attributes of the process object and the application queue. The queue manager puts the trigger message on an *initiation queue*.
4. A long running program called a *trigger monitor* gets the trigger message, examines its contents, and ...
5. ... starts Application B, passing the entire trigger message as a parameter.

6. Application B opens the application queue and gets messages from it.

A trigger monitor can start the application synchronously within its own unit of execution, or asynchronously as a separate unit of execution. Trigger monitors are supplied with WebSphere MQ, but users may write their own.

## Queue attributes controlling triggering

To define an application queue for triggering, the `DEFINE QLOCAL` command must contain the following parameters:

### **TRIGGER**

Triggering is enabled.

### **PROCESS(*string*)**

The name of the process object which identifies the application that can service the application queue.

### **INITQ(*string*)**

The name of the initiation queue.

Additional parameters that control triggering may be specified. These are:

### **TRIGMPRI(*integer*)**

The threshold message priority for triggers. The queue manager ignores messages with a priority lower than this when deciding whether a trigger event should occur.

### **TRIGTYPE(*triggertype*)**

- A trigger type of `FIRST` causes a trigger event to occur when the queue changes from being empty to having one message on it.
- A trigger type of `DEPTH` causes a trigger event to occur when the number of messages on the queue reaches the value indicated by the `TRIGDPTH` parameter.  
When triggering by depth, the queue manager disables triggering by setting the application queue to `NOTRIGGER` after it has created a trigger message. It is the responsibility of the application to re-enable triggering by using the `MQSET` call.
- A trigger type of `NONE` indicates that no trigger event occurs.
- A trigger type of `EVERY` causes a trigger event to occur for every message put on the queue.

**TRIGDPTH(*integer*)**

The number of messages that must be on the queue before a trigger event occurs for `TRIGTYPE(DEPTH)`.

**TRIGDATA(*string*)**

Data that is copied into the trigger message.

## Process attributes

A process object is defined with the MQSC command `DEFINE PROCESS`. Its synonym is `DEF PRO`. It has a keyword `APPLICID` that names the application to be started.

The command:

```
DEFINE PROCESS(MY_PROCESS) APPLICID('C:\MyApp\myprogram.exe')
```

defines a process named `MY_PROCESS` that points to the program specified in `APPLICID`.

## Conditions for a trigger event

All of the conditions listed here must be satisfied for a trigger event to occur:

- A message is put on a queue.
- The priority of the message is not below the value specified in the `TRIGMPRI` attribute of the queue.
- The number of messages previously on the queue is correct for the trigger type.
- The queue is not already open for input (for `TRIGTYPE(FIRST)` and `TRIGTYPE(DEPTH)` only).
- The queue is enabled for get requests.
- A process object exists.
- The initiation queue exists and is enabled for put and get requests.
- The trigger monitor has the initiation queue open for input.
- The queue is defined as `TRIGGER`.



- The queue is not defined as `TRIGTYPE(NONE)`.

## Trigger monitor

Several trigger monitors are supplied with WebSphere MQ. The trigger monitor `runmqtrm` is common to most WebSphere MQ queue managers. Executed without parameters, the `runmqtrm` control command starts a trigger monitor on the default queue manager, using the default initiation queue (`SYSTEM.DEFAULT.INITIATION.QUEUE`). It also provides for parameters to specify the queue manager and initiation queue.

## Trigger monitor errors

Messages relating to the operation of a trigger monitor are produced for two reasons:

- To report on normal activities, such as when a trigger monitor starts and when it ends. These messages do not normally require any user action.
- To report on abnormal conditions, such as when a trigger monitor fails to open the initiation queue or when it fails to start the specified application. These messages normally indicate that user action is required to correct the condition.

Trigger monitor messages may be written to the standard output device and to an error log with more information.

## Implementing triggering (a hands-on exercise)

In this section you'll set up and test a simple triggering application. You will also use a request-reply scenario and see how a reply-to queue can be used. The two sample applications you'll use to implement triggering are `amqsreq` and `amqsech`.

The `amqsreq` sample program is invoked from a command prompt in exactly the same way as `amqsput`, but it accepts a third input parameter. The program reads lines of text from the standard input device, converts them to request messages, and puts the messages on the named queue. Each request message requires a reply-to queue name, which is specified as the third input parameter. If it is omitted, the name defaults to `SYSTEM.SAMPLE.REPLY`. If the name resolves to a model queue, a dynamic queue is created. When the input of text is terminated (null line or EOF), the program waits for the reply messages and writes the text within each reply message to the standard output device.

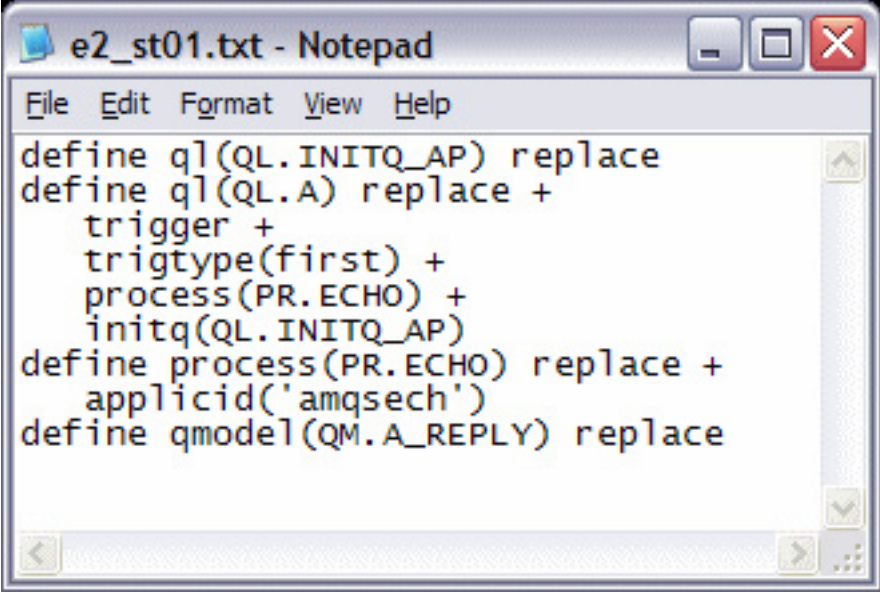
The amqsech sample program is designed to be started by a trigger monitor and not from a command prompt. The program connects to the queue manager passed to it by the trigger monitor and opens the queue also passed by the trigger monitor. This queue is considered the *request queue*. The program gets a message from the request queue, creates a new message containing the same application data as the original message, and puts the new message on the reply-to queue named in the message descriptor of the original message. The program then gets each of the remaining messages on the request queue in turn, and generates a reply in the same way. When the request queue is empty, the program closes the queue and disconnects from the queue manager.

## Configure the queue manager for triggering

First you need to create the WebSphere MQ objects needed for triggering.

1. Use a text editor to create an MQSC command file named e2\_st01.txt, as shown in Figure 15.

**Figure 15. Command file to configure triggering**



```
e2_st01.txt - Notepad
File Edit Format View Help
define ql(QL.INITQ_AP) replace
define ql(QL.A) replace +
  trigger +
  trigtype(first) +
  process(PR.ECHO) +
  initq(QL.INITQ_AP)
define process(PR.ECHO) replace +
  applid('amqsech')
define qmodel(QM.A_REPLY) replace
```

There are four commands in this file. The first command defines a queue to be used as an initiation queue; the second defines a local queue enabled for triggering; the third defines a process object; and the fourth defines a model queue to be used as the reply-to queue.

2. Open a command prompt window, type `runmqsc < e2_st01.txt` and press **Enter**. Your results should look like Figure 16.

**Figure 16. Results of running command file**

```

C:\Documents and Settings\Administrator>runmqsc < e2_st01.txt
5724-H72 (C) Copyright IBM Corp. 1994, 2004. ALL RIGHTS RESERVED.
Starting MQSC for queue manager QMC1.

1 : define ql(QL.INITQ_AP) replace
AMQ8006: WebSphere MQ queue created.
2 : define ql(QL.A) replace +
   : trigger +
   : trigtype(first) +
   : process(PR.ECHO) +
   : initq(QL.INITQ_AP)
AMQ8006: WebSphere MQ queue created.
3 : define process(PR.ECHO) replace +
   : applicid('amqsech')
AMQ8010: WebSphere MQ process created.
4 : define qmodel(QM.A_REPLY) replace
AMQ8006: WebSphere MQ queue created.
4 MQSC commands read.
No commands have a syntax error.
All valid MQSC commands were processed.

C:\Documents and Settings\Administrator>

```

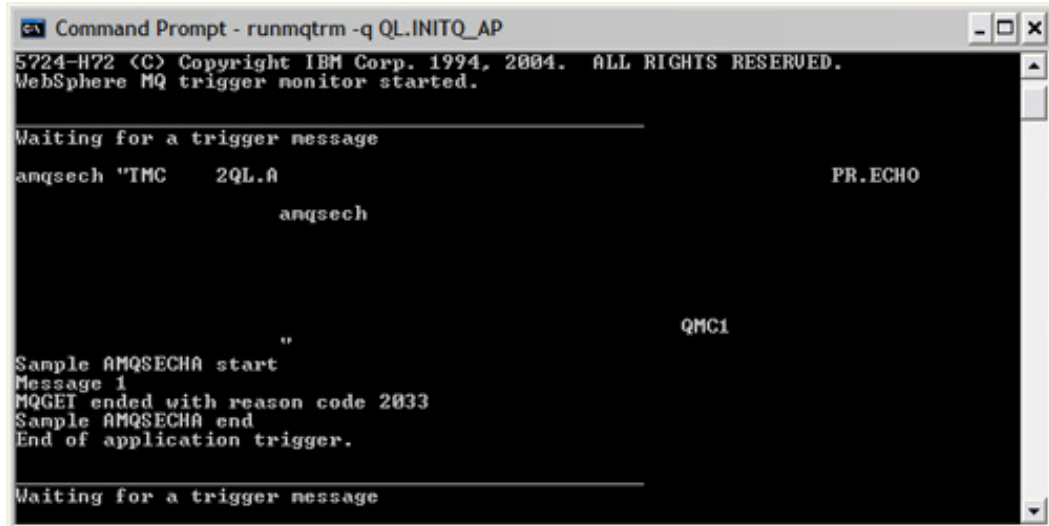
3. Type `cls` and press **Enter** to clear the screen.
4. Open a second command prompt window.
5. In this window, type `runmqtrm -q QL.INITQ_AP` and press **Enter** to start the trigger monitor.

## Test triggering

Now you are ready to test triggering.

1. In the first command prompt window (not the one where the trigger monitor is running), type `amqsreq QL.A QMC1 QM.A_REPLY` and press **Enter**.
2. Type `Message 1` and press **Enter**. You should see the trigger monitor in the other command prompt window start `amqsech` and echo the message you typed, as shown in Figure 17.

### Figure 17. Triggered application



```
Command Prompt - runmqtrm -q QL.INITQ_AP
5724-H72 (C) Copyright IBM Corp. 1994, 2004. ALL RIGHTS RESERVED.
WebSphere MQ trigger monitor started.

Waiting for a trigger message
anqsech "TMC 2QL.A PR.ECHO
      anqsech

Sample AMQSECHA start
Message 1
MQGET ended with reason code 2033
Sample AMQSECHA end
End of application trigger.

Waiting for a trigger message
```

3. Close both command prompt windows.

---

## Section 7. Problem determination

In this section, we'll discuss WebSphere MQ aids for determining the cause of a problem. The IBM Redbook *WebSphere MQ V6 Fundamentals* (see [Resources](#)) has an excellent chapter on troubleshooting that provides more details than this tutorial.

### Configuration problems

If you receive an error message that the queue manager is unavailable, the cause could be something simple such as the queue manager has not been started, or an application not specified, or incorrect queue manager name. If the cause is not something so obvious, check the following:

- The appropriate configuration files exist. See the *WebSphere MQ System Administration Guide* (in [Resources](#)) for the location of these files on your platform.
- Ensure that the configuration files have the appropriate permissions assigned them.
- Ensure that the WebSphere MQ configuration file references the queue manager and has the correct information for locating the files associated with it.

## Error messages

Error messages identify normal errors, typically caused by users, such as the use of an invalid parameter on a control command. The messages are national language enabled through the use of message catalogs. These messages are written to the associated terminal, if any, and are written to an error log file with additional information.

Error messages are only written to the error log file named AMQERR01.LOG. There is a separate error log file with this name in each of three directories. Which of these directories is used for a specific error message depends on the information available to WebSphere MQ at the time of the error. The directories are categorized as:

- Errors where the queue manager name is known and the queue manager is available.
- Errors where the queue manager name is not known or the queue manager is not available.
- All other errors.

When the error log file AMQERR01.LOG fills up (the default capacity is 256KB), its contents are copied to AMQERR02.LOG and AMQERR01.LOG is then reused. Before the copy, AMQERR02.LOG is copied to AMQERR03.LOG. The previous contents of AMQERR03.LOG, if any, are discarded. In this way, AMQERR02.LOG and AMQERR03.LOG are used to maintain a history of error messages.

## First-failure support technology

If an unexpected event has been detected by a WebSphere MQ queue manager, such as an internal queue manager failure, information is provided in a first-failure support technology (FFST) report. The information in these reports relates to the internal operations of WebSphere MQ, and, as such, can be very useful for an IBM representative diagnosing an issue with WebSphere MQ.

Though not all FFST reports represent a failure in WebSphere MQ, because they represent unexpected events you should check for them from time to time, and keep all FFST reports generated. The *WebSphere MQ System Administration Guide* describes the names and locations of FFST reports for your platform.

## Trace

WebSphere MQ provides trace functions that can be started and stopped. When trace is engaged, a large amount of information on the operation of WebSphere MQ is captured in a variety of files. To keep the size of these files reasonable, you can limit the trace functions by time or to a specific WebSphere MQ component. You can also limit tracing to the MQI for diagnosing application problems. The *WebSphere MQ System Administration Guide* describes how to perform a trace for your platform.

---

## Section 8. Recovery

This section describes logging in WebSphere MQ, and how messages and WebSphere MQ objects can be recovered in the event of a failure.

### Message persistence

Messages in WebSphere MQ are considered either *persistent* or *nonpersistent*.

Persistent messages are never lost as a result of a system failure, or as a result of a communication failure when they are being transmitted from one queue manager to another. To achieve this, the persistent messages are written out to a log. When a queue manager is restarted following a system failure, it recovers persistent messages as necessary from the logged data.

Since logging is not used for nonpersistent messages, these can be used for better performance when it is not critical that messages survive a queue manager restart.

Both persistent and nonpersistent messages may be stored on the same queue. The only exception to this is a temporary dynamic queue, which can only store nonpersistent messages.

### Types of logs

Two types of logging are provided by WebSphere MQ: *circular* and *linear*. The type of logging is selected when a queue manager is created; the size and location of the log files may also be specified at this time.

#### **Circular logging**

Used when media recovery is not required. The log files in circular logging are viewed as a closed ring; when a log file contains no active log records, it becomes available for reuse. An active log record is one that is still required to restart the queue manager. One of the advantages of circular logging is that

the disk space required for the log does not increase with time.

### Linear logging

Needed to support media recovery. The log files are viewed as a sequence. A log file used in linear logging is never deleted, but it does become inactive when it contains no active log records. New log files are added to the sequence as required, so no space is reused. Inactive log files could be archived to release disk space, but there is no automatic mechanism to support this.

Periodically, the queue manager performs a *log checkpoint*, which provides a point of consistency for the queue manager data. A checkpoint is recorded in the log as a series of checkpoint records. Checkpoints reduce restart time by minimizing the log replay required.

## Recovering persistent messages

When a queue manager restarts after a system failure, it automatically recovers any persistent messages. It also recovers any damaged WebSphere MQ object that prevents it from starting, but this does not normally include a local queue that is damaged. Such a queue might be detected later when an attempt is made to access it.

In order to restart, a queue manager only requires:

- The log records written since the last checkpoint.
- The log records written by transactions that were still active at the time the queue manager stopped. Uncommitted persistent messages, put or got inside these transactions, are rolled back during restart.

## Damaged objects and media recovery

As noted above, a queue manager does not normally detect that a local queue is damaged during a restart. It only detects a damaged local queue if it is storing uncommitted persistent messages that were put or got inside a transaction that is still active when the queue manager stopped. In this case, the queue manager automatically recreates the local queue, as it needs to be able to roll back the transaction that did not complete.

More typically, a damaged local queue is only detected when an attempt is made to access it. When this occurs, the local queue can be re-created from a linear log by using the `rcrmqobj` control command. You should record a media image of a local queue (using the `rcdmqimg` control command) regularly, so it won't take too long if it becomes necessary to re-create it.

Although you can use the control commands just discussed on other types of WebSphere MQ objects besides local queues, the simplest way to re-create other types of objects is to rerun the WebSphere MQ command that created them in the first place. This applies to alias queues, model queues, local definitions of remote queues, process objects, and channels.

## Dumping the log

The `dmpmqlog` control command can be used to dump a formatted version of the log. It can only be used when the queue manager is not running.

By default, the dump commences from the *head* of the log. The head of the log is the checkpoint that commences the active portion of the log. Normally, this is the most recent checkpoint. However, the head of the log may be positioned at an earlier checkpoint if there are transactions that are still active when the queue manager stopped and there are uncommitted persistent messages that were put or got inside these transactions prior to the most recent checkpoint.

Because a queue manager takes a checkpoint during a normal shutdown, the active portion of the log only contains a small number of log records under these circumstances. However, there are options on the `dmpmqlog` control command that let you specify a different starting point for the dump. The dump may commence:

- At the *base* of the log. The base of the log is the first log record in the log file containing the head of the log.
- At an individual log record identified by a specified *log sequence number* (LSN). Each log record is identified by a unique LSN.
- At the log file identified by a specified *extent number*. All log files have a file name of the form `Snnnnnnn.LOG`, where *nnnnnnn* is the extent number. This option is only available for linear logging.

The information that is formatted in the log includes the put and get of persistent messages, transaction events, and the creation, alteration, and deletion of WebSphere MQ objects.

## Recovery (a hands-on exercise)

Now see how persistent messages survive a queue manager restart.

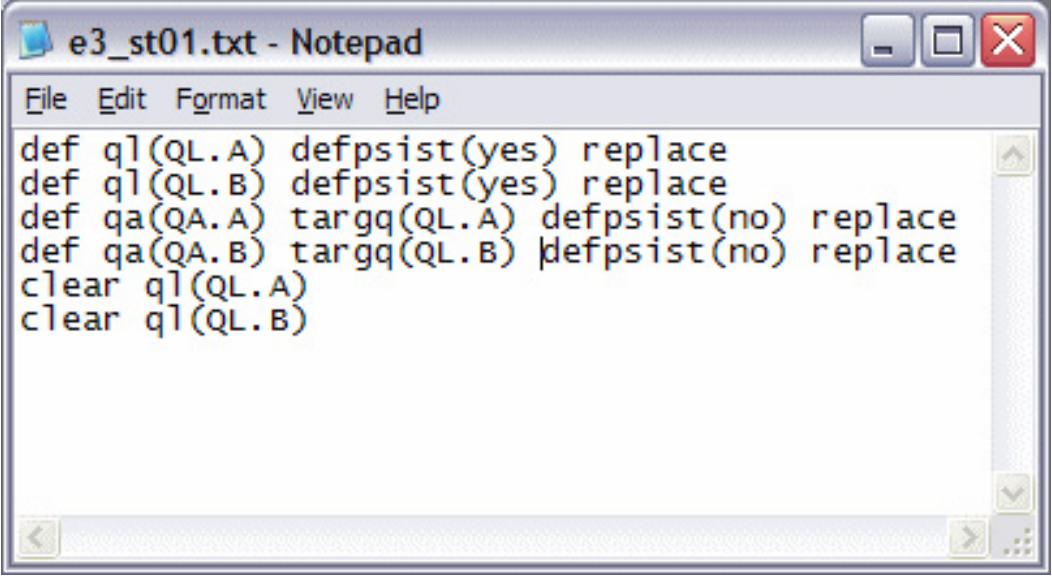
1. Create a MQSC command file that defines two local queues, QL.A and QL.B, that have the default persistence of YES. Also create two alias queues, QA.A and QA.B, that resolve to these local



queues, but have default persistence set to NO.

Finally, clear the messages from the local queues. Call the file e3\_st01.txt. It should look like Figure 18.

**Figure 18. MQSC command file for persistence**



```

File Edit Format View Help
def ql(QL.A) defpsist(yes) replace
def ql(QL.B) defpsist(yes) replace
def qa(QA.A) targq(QL.A) defpsist(no) replace
def qa(QA.B) targq(QL.B) defpsist(no) replace
clear ql(QL.A)
clear ql(QL.B)

```

2. Open a command prompt, type `runmqsc < e3_st01.txt` and press **Enter**. Ensure that all commands executed successfully.
3. Type `amqsput QL.A` and press **Enter**.
4. Type `Persistent message 1` and press **Enter** to place a persistent message on local queue QL.A.
5. Press **Enter** to exit `amqsput`.
6. Type `amqsput QL.B` and press **Enter**.
7. Type `Persistent message 2` and press **Enter** to place a persistent message on local queue QL.B.
8. Press **Enter** to exit `amqsput`.
9. Type `amqsput QA.A` and press **Enter**.
10. Type `Nonpersistent message 1` and press **Enter** to place a persistent message on local queue QL.A (the message is nonpersistent because the default persistence for the explicitly opened queue (QA.A) is set to NO).

11. Press **Enter** to exit `amqsput`.
  12. Type `amqsput QA.B` and press **Enter**.
  13. Type `Nonpersistent message 2` and press **Enter** to place a persistent message on local queue `QA.B`.
  14. Press **Enter** to exit `amqsput`.
  15. Type `amqsbcg QL.A` and press **Enter** to browse the messages on local queue `QL.A`. You should see both the persistent and the nonpersistent message.
  16. Type `endmqm -i QMC1` and press **Enter** to stop the queue manager.
  17. Type `strmqm` to start the queue manager.
  18. Type `amqsbcg QL.A` and press **Enter** to browse the messages on local queue `QL.A`. You should see only the persistent message.
- 

## Section 9. Summary

This tutorial covered installation and configuration of WebSphere MQ, and explored troubleshooting techniques for problems that might occur. Completing the five tutorials in this series can help you gain the knowledge you need to prepare for Test 996, IBM WebSphere MQ V6.0, Solution Design, but nothing can replace the experience and knowledge that is obtained from using the product and studying the documentation.

I hope you have found this tutorial helpful, and wish you luck as you prepare for your certification test.

# Resources

## Learn

- [WebSphere MQ Solution Designer certification prep series](#): Take this series of five tutorials to help prepare for the IBM certification Test 996, WebSphere MQ V6.0, Solution Design.
- Get certified as an IBM Certified Solution Designer - WebSphere MQ V6.0. Check out the objectives, sample assessment tests, and training resources for test 996, [IBM Certified Solution Designer - WebSphere MQ V6.0](#).
- Read the IBM Redbook™ [WebSphere MQ V6 Fundamentals](#) to gain a broad technical introduction to WebSphere MQ.
- Use the [WebSphere MQ library](#) for detailed documentation on WebSphere MQ.
- Stay current with [developerWorks technical events and webcasts](#).

## Get products and technologies

- Download a free trial version of [WebSphere MQ V6.0](#).
- Download a free trial version of [Rational Application Developer version 6.0](#).
- Build your next development project with [IBM trial software](#), available for download directly from developerWorks.

## Discuss

- [Participate in the discussion forum for this content](#).
- Participate in [developerWorks blogs](#) and get involved in the developerWorks community.

## About the author

Willy Farrell

Willy Farrell is a Senior Software Engineer in the IBM Developer Skills Program. As part of the developerWorks team, he provides relevant technical information and insight to developers on the latest e-business and industry trends and technologies through Web content, articles, speaking engagements, and consulting to faculty at [IBM Academic Initiative](#) member universities. He has been programming computers for a living since 1981, began using Java in 1996, and joined IBM in 1998. You can reach Willy at [willyf@us.ibm.com](mailto:willyf@us.ibm.com).