MQSeries® for Sun Solaris

Quick Beginnings

Version 5.2

GC33-1870-03

MQSeries® for Sun Solaris

Quick Beginnings

Version 5.2

GC33-1870-03

Note!
Before using this information and the product it supports, be sure to read the general information under "Appendix C. Notices" on page 81.

Fourth edition (December 2000)

This edition applies to MQSeries for Sun Solaris, V5.2, and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

Figures

Tables

Welcome to MQSeries for Sun Solaris

This book describes MQSeries for Sun Solaris (SPARC Platform Edition), V5.2 and explains how to plan for, install and use the product

Road map

Use Table 1 to find the information you need to get started with *MQSeries for Sun Solaris*.

Table 1. Getting started road map

If you want to...	Refer to...
Learn about system requirements for MQSeries for Sun Solaris	"Chapter 1. Planning to install the MQSeries for Sun Solaris server" on page 3
Install MQSeries for Sun Solaris	"Chapter 2. Installing the MQSeries for Sun Solaris server" on page 9
Install an MQSeries for Sun Solaris client	"Chapter 3. Installing the MQSeries for Sun Solaris client" on page 23
Applying maintenance to MQSeries for Sun Solaris	"Chapter 4. Applying maintenance to MQSeries for Sun Solaris" on page 33
Uninstalling an MQSeries server or client	"Chapter 5. Uninstalling MQSeries for Sun Solaris" on page 35
Read about MQSeries for Sun Solaris	"Chapter 6. About MQSeries" on page 39
Start using command sets	"Chapter 7. Using the MQSeries command sets" on page 47
Start using the Web Interface	"Chapter 8. Using the MQSeries Internet Gateway" on page 59
View or print online documentation	"Chapter 9. Obtaining Additional Information" on page 61
Contact IBM®	*Sending your comments to IBM*

Conventions

Knowing the conventions used in this book will help you use it more efficiently.

- **Boldface type** indicates the name of an item you need to select or the name of a command.

Conventions

- *Italics type* indicates new terms, book titles, or variable information that must be replaced by an actual value.
- `Monospace type` indicates an example (such as a fictitious path or file name) or text that is displayed on the screen.

What's new in MQSeries for Sun Solaris, Version 5 Release 2

MQSeries for Sun Solaris, Version 5 Release 2 provides the following new and changed functions:

- Enhancements have been made to the performance of MQI function, channels, message logging, and application initialization and termination.
- You can now request immediate update of Object Authority Manager (OAM) data, rather than having to stop and restart the queue manager before authorization changes take effect.
- Changes have been made to the way in which OAM data is held, to improve performance.
- Support for Java™ on MQSeries is separately installable from the CD-ROM included in the MQSeries V5.2 product package. Alternatively, you can download the latest version of support for Java on MQSeries from the MQSeries Web site at:

  `http://www.ibm.com/software/mqseries/`
- Support is included for *pipelining*, which is the ability of the Message Channel Agent (MCA) to transfer messages using multiple threads.
- Channel send-exit programs can reserve space in the transmission buffer for their own use. Typically, this would be used by an exit that wanted to encrypt data and add a security key.
- Dynamic Host Configuration Protocol (DHCP) can now be used in queue manager clusters.
- Management of log files for recovery and restart has been improved.
- The area of main storage used to store information relating to a queue manager cluster can be increased dynamically. A new cluster workload-exit call (MQXCLWLN) is provided to support navigation of MQWDR, MQWQR, and MQWCR records held in dynamically increased storage.
- Minor changes to the MQSeries application programming functions have been made, including: support for MQRFH2 (the version-2 rules and formatting header); improvements to the processing of the *CodedCharSetId* field in MQSeries headers; the addition of a command-level value MQCMD_LEVEL_520; and C++ support for MQCNO Version 2 and Version 3.
- IBM WebSphere™ is supported as an XA coordinator.
- The way in which UNIX® signals are handled by MQSeries has been altered to minimize the impact on user applications.

What's new

- Data Connection SNAP-IX, V6.2 or later, can be used for SNA configuration.
- V5.0 of the Sun Workshop C++ compiler and V6.0 of the Forte C++ compiler are supported.

For a complete description of new and changed function in this product, see the *MQSeries V5.2 Release Guide*.

Part 1. Installing MQSeries for Sun Solaris

Chapter 1. Planning to install the MQSeries for Sun Solaris server

This chapter is a summary of the requirements for running MQSeries for Sun Solaris including:

- Network protocols
- Compilers
- Delivery media
- Various components of the product

The following information applies to the server environment only. For information about installing the IBM MQSeries for Sun Solaris client, see "Chapter 3. Installing the MQSeries for Sun Solaris client" on page 23.

Year 2000 compatibility

MQSeries, when used in accordance with its associated documentation, is capable of correctly processing, providing, and/or receiving date data within and between the twentieth and twenty-first centuries, provided that all products (for example, hardware, software, and firmware) used with this IBM Program properly exchange accurate date data with it.

Customers should contact third-party owners or vendors regarding the readiness status of their products.

IBM reserves the right to update the information shown here. For the latest information regarding levels of supported software, refer to:

`http://www.software.ibm.com/ts/mqseries/platforms/supported.html`

For the latest IBM statement regarding Year 2000 readiness, refer to:

`http://www.ibm.com/IBM/year2000/`

Hardware requirements

- MQSeries Servers:

  All Sun SPARC desktop or server systems and Sun UltraSPARC desktop or server systems, supported by the appropriate release of the Solaris operating environment, as shown in "Software requirements" on page 4.

Hardware requirements

Disk storage

The installation requirements depend on which components you install and how much working space you need. This, in turn, depends on the number of queues that you use, the number and size of the messages on the queues, and whether the messages are persistent or not. You also require archiving capacity on disk, tape, or other media.

Note: Use the `df -k` command to determine the amount of free space on your disk

These are the approximate storage requirements:

- Server:

  A minimum of 25 MB of disk space must be available for the product code and data in the filesystem containing the `/opt` directory.

  Working data for MQSeries for Sun Solaris is stored by default in `/var/mqm`.

- Online books

  In addition, if you install the online books in HTML format you require 35 MB of storage for the books in the `/opt` directory.

  After installation the books are placed in the `/opt/mqm/html` directory.

Note: For added confidence in the integrity of your data, you are strongly advised to put your logs onto a *different* physical drive from the one that you use for the queues.

Software requirements

Minimum supported levels are shown. Later compatible levels, if any, will be supported unless otherwise stated.

- Sun Solaris Version 2.6 with patches
  - 105181–12
  - 105210–19
  - 107733–06
  - 105568–13
  - 105591–05 (needed if C++ is used)
  - 106125–05
- Sun Solaris 7 with patches
  - 107171-02
  - 107544-03
  - 106541-09
  - 106950-03
  - 106980-05
  - 106327-05 (needed if C++ is used)

– 107443-11 (needed if using DCE 3.1 on Sun Solaris version 7)
– 107709-04 (needed if using DCE 3.1 on Sun Solaris version 7)

Notes:

1. 107544-03 must be installed before 106541-09.
2. 106541-09 is the earliest recommended level of kernel update patch.

- Sun Solaris 8

  Note: Currently we are unaware of any patches required for Solaris 8, however we recommend that you check the README for the latest information.

Connectivity

Network protocols supported are SNA LU 6.2 and TCP.

For SNA connectivity you can use SunLink SNA Peer-to-Peer Version 9.1 or SNAP-IX (SPARC Platform Edition), V6.2 or later.

If token ring is used: you need Sun TRI Driver 4.0 or later.

TCP/IP is part of the base operating system. In MQSeries for Sun Solaris, V5.2, enhancements are introduced that make it more practical to combine the use of DHCP with MQSeries queue manager clusters.

For information about specifying the SNA product you want to use (SunLink SNA or Data Connection SNAP-IX) see the *MQSeries V5.2 Release Guide*.

Compilers supported for Solaris applications

- Sun Workshop compiler C 4.2
- Sun Workshop compiler C++ 4.2
- Sun Workshop compiler C 5.0
- Sun Workshop compiler C++ 5.0
- Forte C 6 (Sun Workshop 6 C)
- Forte C++ 6 (Sun Workshop 6 C++)
- Merant Server Express V1.1

Options

You may use the following options with MQSeries for Sun Solaris.

Options

Transaction monitors

The following transaction processing monitors (coordination can be through X/Open XA interface) may be used:

- WebSphere 3.0x and 3.5x
- BEA Tuxedo Version 6.4 or 6.5

MQSeries for Sun Solaris, V5.2 supports WebSphere as an XA coordinator. For more information about the MQSeries application adaptor, and about how to write Component Broker applications please see the *WebSphere Application Server Enterprise Edition Component Broker MQSeries Application Adaptor Development Guide*, SC09–4444.

Databases

- Oracle 8i and 8iR2 (8.1.5 or 8.1.6)
- DB2® Universal Database V5.0, V6.1 and 7
- Sybase V11:
  - Adaptive Server Enterprise (A.S.E.), V11.5
  - Open Client (ctlib and dblib), V11.1
  - Embedded SQL/C, V11.0
  - XA Server, V11.1
- Sybase Adaptive Server Enterprise, V12, with DTM option.

Note: For more information on how to set up XA Coordination please refer to the *MQSeries System Administration* book.

DCE

- Transarc DCE-DFS 2.0 for Sun Solaris 2.6
- IBM DCE 3.1 for Sun Solaris 7
- IBM DCE 3.1 for Sun Solaris 8

This must be a DCE product that supports DES data encryption if you want to run the MQSeries supplied DCE send, receive, or message exits.

DCE names and security modules are provided with MQSeries for Sun Solaris, V5.2.

Delivery

MQSeries for Sun Solaris, V5.2 is supplied on CD-ROM.

Support for Java on MQSeries is separately installable from the CD-ROM included in this product package. Alternatively, you can download support for Java on MQSeries from the MQSeries Web site at

`http://www.ibm.com/software/mqseries`

where the latest version of this support is always available.

Installation

MQSeries for Sun Solaris takes approximately 5 minutes to install, using the Solaris **pkgadd** program. The installation process is described in "Chapter 2. Installing the MQSeries for Sun Solaris server" on page 9.

It is possible to install MQSeries using a silent install, for more information see "Silent installation" on page 14

MQSeries for Sun Solaris components

MQSeries for Sun Solaris, V5.2 contains the following components (sometimes called filesets):

Table 2. Components

Title	Description
MQSeries Server	Support for client connections. Requires the runtime component to be installed.
Man Pages	Man pages for the following commands: • Control commands • Message Queue Interface (MQI) • MQSeries commands (MQSC)
Sample Programs	Sample application programs.
Sun Solaris 2 client libraries	The MQSeries for Sun Solaris Client can be installed on the server machine, enabling you to have the MQSeries server and client on the same machine.

MQSeries for Sun Solaris components

Table 2. Components (continued)

Title	Description
MQSeries online documentation:	HTML versions of the • MQSeries Documentation • MQSeries Internet Documentation are provided. PDF versions of the MQSeries books are also on the CD-ROM, but are not listed as installable components. HTML and PDF versions of some of the MQSeries books are available in the following national languages: U.S. English Brazilian Portuguese French German Italian Japanese Korean Spanish Simplified Chinese Traditional Chinese
DCE Samples	This should be installed ***only*** if you are going to use DCE.
Internet Gateway runtime	Provides access to MQSeries applications through HTML, CGI, ICAPI, and NSAPI
Internet Gateway samples	
Message catalogs	
DCE	Support for DCE names or security on the server. This should be installed ***only*** if you are going to use DCE.

Note: The "base" product is automatically installed.

README file

Before starting to install MQSeries for Sun Solaris, review the README file, which you will find in the root directory of the CD-ROM.

Chapter 2. Installing the MQSeries for Sun Solaris server

This chapter tells you how to install MQSeries for Sun Solaris and how to verify that your installation has been successful.

The MQSeries product is installed into the `/opt/mqm` directory. This ***cannot*** be changed. However, if you do not have enough space in the `/opt/mqm` file system, follow the procedure given in "Creating another file system for product code" on page 11.

Note: The MQSeries product is contained in the `/mq_solaris` directory of the CD-ROM.

Preparing for installation

This section guides you through some of the steps you must perform before you install MQSeries for Sun Solaris.

If you have a previous version of MQSeries for Sun Solaris already installed, make sure you refer to "Migrating from an earlier version of MQSeries for Sun Solaris" on page 13 before installing MQSeries for Sun Solaris, V5.2.

Creating the system default objects

When you use the **crtmqm** command to create a queue manager with this release of MQSeries, the system default objects are automatically created. The sample MQSC definition file, amqscoma.tst, is no longer provided.

If you used amqscoma.tst to customize your settings for V5.0, and you want to use the same settings with V5.2 of the product:

1. Save your copy of amqscoma.tst
2. Install MQSeries V5.2
3. Load your copy of amqscoma.tst and use the file to recreate your default objects

Before installation

Before you can install MQSeries for Sun Solaris you:

- Must install any patches listed in the README file
- Must create a group with the name `mqm`
- Must add **root** to the `mqm` group.
- Must create a user ID with the name `mqm`

- Are recommended to create and mount a /var/mqm file system, or /var/mqm, /var/mqm/log, and /var/mqm/errors file systems for your data

  You should allow a minimum of 30 MB of storage for /var/mqm, 2 MB of storage for /var/mqm/errors, and 20 MB of storage for /var/mqm/log if you are creating separate file systems.

  For a single file system, use the sum of these figures as a guide.

  Notes:

  1. To determine the size of the /var/mqm file system you should consider:
     - The maximum number of messages in the system at one time
     - Contingency for message build-ups, if there is a system problem
     - The average size of the message data plus 500 bytes for the message header
     - The number of queues
     - The size of log files and error messages, if these are not going to be in a separate file system.
  2. It is better to use all the space available, monitor the usage for a few weeks and, if appropriate, reduce the amount of space allocated. If the initial file system is too small, this can cause problems later.
  3. If you create separate partitions, the following directories ***must*** be on a local file system:
     - /var/mqm
     - /var/mqm/log

     You can NFS mount the /var/mqm/errors and /var/mqm/trace directories to conserve space on your local system.
  4. The size of the log file depends upon the log settings that you use. The size recommended is for circular logging (you will need to check whether this is appropriate for your environment) using the default settings. For further information on log sizes see the *MQSeries System Administration* book.

After installation, this user ID (mqm) owns the directories and files that contain the resources associated with the product. This group and user must be defined for any machine on which the MQSeries software is to be installed, whether the machine is a client or a server machine.

If you want to run any administration commands, for example, **crtmqm** (create queue manager) or **strmqm** (start queue manager), your user ID must be a member of group mqm.

For stand-alone machines, you can create the new user and group IDs locally. For machines administered in a network information services (NIS) domain, you can create the user and group IDs on the NIS master server machine.

Creating another file system for product code

If you do not want to have the product code installed in the/opt/mqm file system, for example, if that file system is too small to contain the product, you can do one of two things:

1. Create a new file system and mount it as /opt/mqm.
2. Create a new directory anywhere on your machine that is large enough to contain the product, and create a symbolic link from /opt/mqm to this new directory. For example:

```
mkdir /bigdisk/mqm
ln -s /bigdisk/mqm /opt/mqm
```

Notes:

1. Whichever of these options you pick, you ***must*** do it before installing the product code.
2. The file system into which the code is installed can be a remote network device, for example NFS, provided that the mount options are defined on that device to allow ***setuid*** programs – including root access – to be run.

Kernel configuration

MQSeries makes use of semaphores, shared memory, and file descriptors, and it is probable that the default kernel configuration is not adequate.

In particular, the default number of semaphores is 60, which is *not* sufficient to support MQSeries.

If you attempt to use MQSeries without increasing semmni, the number of semaphores, the queue manager fails and produces a First Failure Support Technology™™ (FFST) file. This file indicates that the system call semop received an argument that was not valid. An example of a possible set of kernel values for all supported versions of Solaris is given in Figure 1 on page 12. However, for Sun Solaris the number of semaphores and semaphore sets has to be such that their control structures occupy less than 25% of the kernel storage.

After installation, you should review the machine's configuration. To do this type the following command:

```
sysdef -i
```

To change the values, add a set parameter = value line to the /etc/system file. For further information on setting up the system, see the Sun Solaris System Administration documentation.

```
set shmsys:shminfo_shmmax = 4294967295
set shmsys:shminfo_shmseg = 1024
set shmsys:shminfo_shmmni = 1024
set semsys:seminfo_semaem = 16384
set shmsys:shminfo_semmni = 1024
set semsys:seminfo_semmap = 1026
set semsys:seminfo_semmns = 16384
set semsys:seminfo_semmsl = 100
set semsys:seminfo_semopm = 100
set semsys:seminfo_semmnu = 2048
set semsys:seminfo_semume = 256
set msgsys:msginfo_msgmap = 1026
set msgsys:msginfo_msgmax = 4096
```

Figure 1. Kernel parameter values - example setting on a Solaris system

Notes:

1. Shared memory usage does not vary with message rate or persistence.
2. Semaphore and swap usage does not vary with message size, message rate or persistence.
3. MQSeries queue managers are independent of each other. Therefore system kernel parameters, for example `shmmni`, `semmni`, `semmns`, and `semmnu` need to allow for the number of queue managers in the system.

For more details see the relevant SupportPac™ , which is available on the MQSeries Web site at `http://www.ibm.com/software/mqseries/`.

Sun Solaris has a low default system soft limit for the number of file descriptors. When running a multi threaded process, you might reach the soft limit for file descriptors. This will give you the MQSeries reason code MQRC_UNEXPECTED_ERROR (`2195`) and, **if there are enough** file descriptors, an MQSeries FFST™ file.

To avoid this problem you can increase the system soft limit for the number of file descriptors.

To increase the number of file descriptors edit the **/etc/system** file and change the value of the system soft limit to match the system hard limit (1024) by adding `set rlim_fd_cur=1024`.

Additionally, if you are running MQSeries under the Lotus® Domino™ server, you can reduce the number of active server threads in the Domino HTTP server process by opening the server **Name and address** book, and reducing the **Number active threads** value on the server document to between 50 and 60.

Migrating from an earlier version of MQSeries for Sun Solaris

You are strongly recommended to backup all

- Queue manager data
- Queue manager logs
- Queue manager object definitions

before starting to migrate to a new version of MQSeries.

To migrate from an earlier version of MQSeries for Sun Solaris to MQSeries for Sun Solaris, V5.2, you need to end all MQSeries activity on the target machine using `endmqm`. See "Stopping a queue manager" on page 51 for information on how to use the `endmqm` command.

Changes to signal handling

In MQSeries for Sun Solaris, V5.2, the way in which signals are handled has changed. These changes, and their effects on your existing applications, are described in the *MQSeries V5.2 Release Guide*.

Changes to qm.ini for LU62 channels using Sunlink 9.1

In MQSeries for Sun Solaris, V5.2, it is possible to choose which LU6.2 software is used. For Sunlink 9.1 a new entry is required under the LU6.2 stanza. For details see the *MQSeries V5.2 Release Guide*.

Installing the MQSeries for Sun Solaris server

This section describes the installation of the MQSeries for Sun Solaris server.

Notes:

1. If you have previously installed MQSeries on your system, you need to remove the product using the **pkgrm** program. See "Migrating from an earlier version of MQSeries for Sun Solaris" for more information.
2. If the product is present, but not installed correctly, you might need to manually delete the files and directories contained in:
   `/opt/mqm`

Carry out the following procedure:

1. Mount the CD-ROM.
2. Use the Solaris **pkgadd** program, to install the software by carrying out the following procedure:
   a. Type `pkgadd -d /cdrom/mq_solaris`.
   b. You are prompted for a list of components to install. Select those you require. If you want to install the entire MQSeries product, select **all**.
   c. Press the Enter key.

Server installation

For further information on using **pkgadd** to install software packages, see the Solaris documentation.

Silent installation

There is a silent installation which can be run using the script `silent.sh` in the silent directory, off the root of the CD. This script allows you to perform a silent non-interactive install of MQSeries, which produces no output and requires no input. More information can be found in the script file `silent.sh`.

Note: To use silent installation, it is necessary to modify the response file as described in the **pkgadd** documentation, and then run

```
./silent.sh
```

Installing the server and client on the same machine

To install an MQSeries for Sun Solaris client on the server machine, use the MQSeries Server CD-ROM. Choose the client install option on the server CD-ROM to install the client code on the server machine. Do not use the MQSeries Clients CD-ROM.

You might install components from the MQSeries Clients CD-ROM onto a machine and then later want to install the MQSeries server component on the same machine. If so, you must first remove from the machine any of the components that were installed from the MQSeries Clients CD-ROM. You can then use the MQSeries Server CD-ROM to install the server, client, and any other components that you need. You cannot install the server on a machine that already has other components installed from the MQSeries Clients CD-ROM.

For more information about installing the client on a different machine from the server see "Chapter 3. Installing the MQSeries for Sun Solaris client" on page 23.

Support for Java on MQSeries

If you have applications that require support for Java on MQSeries, you will need to install it separately after installing MQSeries.

Support for Java on MQSeries is separately installable from the CD-ROM included in this product package. Alternatively, you can download support for Java on MQSeries from the MQSeries Web site at

```
http://www.ibm.com/software/mqseries
```

, where the latest version of this support is always available

Translated messages

Messages in U.S. English are always available. If you require another of the languages that is supported by MQSeries for Sun Solaris, V5.2, you *must* ensure that your NLSPATH environment variable includes the appropriate directory.

For example:

```
export LANG=de
export NLSPATH=/usr/lib/locale/%L/LC_MESSAGES/%N
```

Translated books

If you choose to install the On-line Documentation component, you will get books in the language that was specified when your operating system was installed. However, some books may not be available in languages other than U.S. English and some hypertext links between books may not work. To overcome this you must install a complete set of books in U.S. English as well as those in your national language. See "Online Information" on page 62 for more information about hypertext linking between translated books.

Verifying the installation of MQSeries for Sun Solaris

This section describes how to verify that MQSeries for Sun Solaris has been correctly installed and configured. You do this by following the steps outlined in "Verification procedure".

If you want to verify a communications link between multiple MQSeries installations (for example between two servers or between a client and a server), you must ensure that the required communications protocols have been installed (and configured) on ***both*** machines.

The supported protocols are TCP and SNA.

Note: The following examples assume that you will be using a TCP connection; for information about using other protocols, see the *MQSeries Intercommunication* book.

However, you can also verify a *local* installation (which has no communications links with other MQSeries installations) without any communications protocols installed.

Verification procedure

You can verify an MQSeries installation at three levels:

- A local (stand-alone) installation, involving no communication links to other MQSeries machines

- A server-to-server installation, involving communication links with other MQSeries servers
- A client/server installation, involving communication links between a server machine and an MQSeries client

Verification of local and server-to-server installations is described in "Verifying a local installation", and in "Verifying a server-to-server installation" on page 18. For information on verifying a client/server installation, see "Verifying the installation" on page 27.

Verifying a local installation

Before verifying the local installation using the sample applications, check that `/opt/mqm/samp/bin` has been included in your PATH environment variable.

Follow these steps to install and test a simple configuration of one queue manager and one queue, using sample applications to put a message onto the queue and to read the message from the queue:

1. Install MQSeries for Sun Solaris on the workstation (include the Server and samples components as a minimum).
2. Create a default queue manager (in this example called `venus.queue.manager`):
   - At the command prompt in the window type:
     ```
     crtmqm -q venus.queue.manager
     ```
   - Press Enter.

     Messages are displayed telling you that the queue manager has been created, and that the default MQSeries objects have been created.

     Note: In prior releases of MQSeries it was necessary to run a script file called **amqscoma.tst** to define the MQSeries default objects. This step is not required in this release of the product.
3. Start the default queue manager:
   - Type `strmqm` and press Enter.

     A message tells you when the queue manager has started.
4. To enable MQSC commands:
   - Type `runmqsc` and press Enter.

   Note: MQSC has started when the following message is displayed:
   ```
   Starting MQSeries Commands.
   ```

   MQSC has no command prompt.
5. Define a local queue (in this example, called `ORANGE.QUEUE`):
   - Type the following and press Enter:

```
define qlocal (orange.queue)
```

Note: Any text entered in MQSC in lowercase is converted automatically to uppercase unless you enclose it in single quotation marks. This means that if you create a queue with the name `orange.queue`, you must remember to refer to it in any commands outside MQSC as `ORANGE.QUEUE`.

The message `MQSeries queue created` is displayed when the queue has been created.

You have now defined:
- A default queue manager called `venus.queue.manager`
- A queue called `ORANGE.QUEUE`

6. Stop MQSC by typing **end**, and pressing Enter.

   The following message is displayed:

   ```
   One MQSC commands read.
   No commands have a syntax error.
   All valid MQSC commands were processed.
   ```

7. The command prompt is now displayed again.

To test the queue and queue manager, use the samples **amqsput** (to put a message on the queue) and **amqsget** (to get the message from the queue):

1. Change into the following directory, if it is not in your PATH:

   ```
   /opt/mqm/samp/bin
   ```

2. To put a message on the queue, type the following command and press Enter:

   ```
   ./amqsput ORANGE.QUEUE
   ```

   The following message is displayed:

   ```
   Sample AMQSPUT0 start
   target queue is ORANGE.QUEUE
   ```

3. Type some message text and then press Enter **twice**.

   The following message is displayed:

   ```
   Sample AMQSPUT0 end
   ```

   Your message is now on the queue and the command prompt is displayed again.

4. To get the message from the queue, type the following command and press Enter:

   ```
   ./amqsget ORANGE.QUEUE
   ```

   The sample program starts, your message is displayed, the sample ends, and the command prompt is displayed again.

The verification is complete.

Verifying a server-to-server installation

The steps involved in verifying a server-to-server installation are more complex, because the communications link between the two machines must be checked.

Follow these steps to set up two workstations, one as a sender and one as a receiver.

Sender workstation:

1. Create a default queue manager called `saturn.queue.manager`:
   - At a command prompt in a window, type:
     ```
     crtmqm -q saturn.queue.manager
     ```
   - Press Enter.

     Messages are displayed telling you that the queue manager has been created, and that the default MQSeries objects have been created.

     Note: In some prior releases of MQSeries it was necessary to run a script file called **amqscoma.tst** to define the MQSeries default objects. This step is not required in this release of the product.
2. Start the queue manager:
   - Type the following and then press Enter:
     ```
     strmqm
     ```

     A message tells you when the queue manager has started.
3. Enable MQSeries Commands (MQSC) by typing the following command and then pressing Enter:
   ```
   runmqsc
   ```

   Note: MQSC has started when the following message is displayed:
   ```
   Starting MQSeries Commands
   ```

   MQSC has no command prompt
4. Define a local queue to be used as a transmission queue, called `TRANSMIT1.QUEUE`:
   - Type the following and press Enter:
     ```
     define qlocal (transmit1.queue) usage (xmitq)
     ```

     The message `MQSeries queue created` is displayed when the queue has been created.
5. Create a local definition of the remote queue:

```
define qremote (local.def.of.remote.queue) rname (orange.queue) +
 rqmname ('venus.queue.manager') xmitq (transmit1.queue)
```

Note: The RNAME parameter specifies the name of the queue on the remote machine to which the message is being sent. Therefore, the name specified by the RNAME parameter (ORANGE.QUEUE) must be the same as the name of the queue to which the message is being sent (ORANGE.QUEUE on the receiver workstation).

6. Define a sender channel:

```
define channel (first.channel) chltype (sdr) conname (9.20.11.182) +
 xmitq (transmit1.queue) trptype (tcp)
```

Notes:

a. The value *9.20.11.182* is the TCP address of the receiver workstation (note that this example is TCP specific).

b. If required the port number can be used explicitly, for example`(9.20.11.182(8192)).`

You have now defined the following objects:

- A default queue manager called `saturn.queue.manager`
- A transmission queue called `TRANSMIT1.QUEUE`
- A remote queue called `LOCAL.DEF.OF.REMOTE.QUEUE`
- A sender channel called `FIRST.CHANNEL`

7. Stop MQSC by typing **end**, and pressing Enter.

Now set up the receiver workstation.

Receiver workstation:

Note: You must be logged in as a superuser, or as root, to perform steps 1 to 4.

1. Edit the file /etc/services. If you do not have the following line in that file, add it as shown:

```
MQSeries        1414/tcp        # MQSeries channel listener
```

2. Edit the file /etc/inetd.conf. If you do not have the following line in that file, add it as shown:

```
MQSeries stream tcp nowait mqm /opt/mqm/bin/amqcrsta amqcrsta
```

Note: If you are not creating `venus.queue.manager` (in step 5) as the **default** queue manager on this workstation, add `-m venus.queue.manager` to the end of this line to specify the name of the queue manager to use.

3. Find the process ID of the inetd with the command:

```
ps -ef | grep inetd
```

4. Run the command:

```
kill -1 inetd processid
```

5. Create a default queue manager (in this example called `venus.queue.manager`):
   - At the command prompt, type:
     ```
     crtmqm -q venus.queue.manager
     ```
   - Press Enter.

     Messages are displayed telling you that the queue manager has been created, and that the default MQSeries objects have been created.

     Note: In some prior releases of MQSeries it was necessary to run a script file called **amqscoma.tst** to define the MQSeries default objects. This step is not required in this release of the product.
6. Start the queue manager:
   - Type the following and then press Enter:
     ```
     strmqm
     ```

     A message tells you when the queue manager has started.
7. Enable MQSC by typing the following command and then pressing Enter:
   ```
   runmqsc
   ```

   Note: MQSC has started when the following message is displayed:
   ```
   Starting MQSeries Commands

   .
   ```

   MQSC has no command prompt.
8. Define a local queue (in this example, called `ORANGE.QUEUE`):
   - Type the following and press Enter:
     ```
     define qlocal (orange.queue)
     ```

     The message
     ```
     MQSeries queue created
     ```

     is displayed when the queue has been created.
9. Create a receiver channel:
   ```
   define channel (first.channel) chltype (rcvr) trptype (tcp)
   ```

   You have now defined the following objects:
   - A default queue manager called `venus.queue.manager`
   - A queue called `ORANGE.QUEUE`
   - A receiver channel called `FIRST.CHANNEL`

10. Stop MQSC by typing **end** and pressing Enter.

Establishing communication between the workstations:

1. If the queue managers on the two workstations have been stopped for any reason, restart them now (using the **strmqm** command).
2. On the ***Sender*** workstation start the sender channel by entering the following command:

   ```
   runmqchl -c FIRST.CHANNEL -m saturn.queue.manager
   ```

   The receiver channel on the receiver workstation is started automatically when the sender channel starts.
3. On the ***Sender*** workstation, use the `amqsput` sample program to send a message to the queue on the receiver workstation:

   ```
   amqsput LOCAL.DEF.OF.REMOTE.QUEUE
   ```

   Note: You put the message to the local definition of the remote queue, which in turn specifies the name of the remote queue.
4. Type the text of the message and press Enter ***twice***.
5. On the ***Receiver*** workstation, use the `amqsget` sample program to get the message from the queue:

   ```
   amqsget ORANGE.QUEUE
   ```

   The message is displayed.

   The verification is complete.

User exits

Check that your user exits are linked with threaded libraries before using them on this version of the product.

See the *MQSeries Application Programming Guide* for further details on threaded libraries.

Setting the queue manager CCSID on MQSeries for Sun Solaris

The coded character set identifier (CCSID) is fixed when the queue manager is created. The CCSID used is the one for the code set of the locale that you are using to run the **crtmqm** command.

Examples of setting the CCSID on Solaris Versions 2.6 and 7:

```
export LANG=en_US
export LANG=pl
```

Examples of setting the CCSID on Solaris 8:

```
export LANG=en_US.ISO8859-1
  uses the code set ISO8859-1
  and will set a CCSID of 819
export LANG=pl_PL.ISO8859-2
  uses the code set ISO8859-2
  and will set a CCSID of 912
```

To modify an existing queue manager CCSID, follow this procedure:

1. Record the existing queue manager CCSID, using the MQSeries (MQSC) command:

   ```
   DISplay QMGR CCSID
   ```

2. Change the CCSID to the new CCSID, with the MQSC command:

   ```
   ALTer QMGR CCSID
   ```

3. Stop the queue manager.
4. Restart the queue manager and any channels it uses.

Chapter 3. Installing the MQSeries for Sun Solaris client

This chapter describes how to:

- Plan for the installation of the MQSeries for Sun Solaris client.
- Install the MQSeries for Sun Solaris client on a different machine from the MQSeries server, using the MQSeries Client CD-ROM.
- Verify communication between the MQSeries for Sun Solaris client and an MQSeries server.

Note: The MQSeries client is installed into the `/opt/mqm` directory. This ***cannot*** be changed.

Planning to install the MQSeries for Sun Solaris client

This section identifies:

- The hardware and software required by the MQSeries for Sun Solaris client
- Tasks you must perform before installing the MQSeries for Sun Solaris client

Hardware requirements

The MQSeries for Sun Solaris client

- Can be installed on all Sun SPARC desktop or server systems, and Sun UltraSPARC desktop or server systems, supported by the appropriate release of the Solaris operating environment. See "Software requirements" for information about the required operating environment.
- Requires a minimum of 25 MB of disk space.

Note: Solaris systems from other manufacturers are not supported.

Software requirements

Minimum supported levels are shown. Later compatible levels, if any, will be supported unless otherwise stated.

- Sun Solaris Version 2.6 with patches
  - 105181–12
  - 105210–19
  - 107733–06
  - 105568–13
  - 105591–05 (needed if C++ is used)
  - 106125–05
- Sun Solaris 7 with patches
  - 107171-02
  - 107544-03

- 106541-09
- 106950-03
- 106980-05
- 106327-05 (needed if C++ is used)

Notes:

1. 107544-03 must be installed before 106541-09.
2. 106541-09 is the earliest recommended level of kernel update patch.

- Sun Solaris 8

  Currently we are unaware of any patches required for Solaris 8, however we recommend that you check the README file for the latest information.

Connectivity

Network protocols supported are SNA LU 6.2 and TCP.

For SNA connectivity you can use SunLink SNA Peer-to-Peer Version 9.1 or SNAP-IX (SPARC Platform Edition), V6.2 or later.

If token ring is used: you need Sun TRI Driver 4.0 or later.

TCP/IP is part of the base operating system. In MQSeries for Sun Solaris, V5.2, enhancements are introduced that make it more practical to combine the use of DHCP with MQSeries queue manager clusters.

For information about specifying the SNA product you want to use (SunLink SNA or Data Connection SNAP-IX) see the *MQSeries V5.2 Release Guide*.

Compilers supported for Solaris applications

- Sun Workshop compiler C 4.2
- Sun Workshop compiler C++ 4.2
- Sun Workshop compiler C 5.0
- Sun Workshop compiler C++ 5.0
- Forte C 6 (Sun Workshop 6 C)
- Forte C++ 6 (Sun Workshop 6 C++)
- Merant Server Express V1.1

The installation directory

To install MQSeries on a Sun Solaris system you use the MQSeries Client CD-ROM supplied as part of the MQSeries product.

Note: If you plan to install an MQSeries client and server on the same machine see "Installing the server and client on the same machine" on page 14.

The MQSeries for Sun Solaris client is installed into the `/opt/mqm` directory. This cannot be changed.

Before installation

Before you can install the MQSeries for Sun Solaris client, you must:

- Check the README file for latest information.
- Create a group with the name mqm.
- Create a user ID with the name mqm.
- Create and mount either a /var/mqm file system, or /var/mqm, /var/mqm/log, and /var/mqm/errors file systems.

Notes:

1. If you create separate partitions, the directory /var/mqm must be on a local file system.

   You can NFS mount the /var/mqm/errors and /var/mqm/trace directories to conserve space on your local system.
2. After installation, the user ID mqm owns the directories and files that contain the resources associated with the product. This group and user must be defined for any machine on which the MQSeries software is to be installed, whether the machine is a client or a server machine.
3. For stand-alone machines, you can create the new user and group IDs locally (or allow them to be created automatically during the installation). For machines administered in a network information services (NIS) domain, you must create the user and group IDs on the NIS master server machine before beginning the installation.

Kernel configuration

See the MQSeries family web site at `http://www.ibm.com/software/mqseries/` for a SupportPac that gives additional performance information.

Migrating from an earlier version of MQSeries for Sun Solaris

You are strongly recommended to backup all

- Queue manager data
- Queue manager logs
- Queue manager object definitions

before starting to migrate to a new version of MQSeries.

To migrate from an earlier version of MQSeries for Sun Solaris to MQSeries for Sun Solaris, V5.2, you first need to end all MQSeries activity on the target machine using `endmqm`. See “Stopping a queue manager” on page 51 for information on how to use the `endmqm` command.

Check for shared memory segments and semaphores using `ipcs -a`.

Migrating from an earlier version

Next, you must use the `ipcrm` command to remove any shared memory segments and semaphores.

If you fail to remove any shared memory segments and semaphores before uninstalling, later versions of MQSeries might be corrupted.

You are now ready to install MQSeries for Sun Solaris, V5.2, as described in "Installing the MQSeries for Sun Solaris client".

Changes to signal handling

In MQSeries for Sun Solaris, V5.2, the way in which signals are handled has changed. These changes, and their effects on your existing applications, are described in the *MQSeries V5.2 Release Guide*.

Installing the MQSeries for Sun Solaris client

To install the MQSeries for Sun Solaris client, carry out the following procedure:

1. Check whether Volume Manager is running on your system by typing the following command:

   ```
   /usr/bin/ps -ef | /bin/grep vold
   ```

   If the Volume Manager is running, the CD–ROM is mounted on /cdrom/mq_solaris automatically. If it is not running, mount the CD-ROM by typing the following commands:

   ```
   mkdir -p /cdrom/mqclient
   mount -F hsfs -r /dev/dsk/cntndnsn /cdrom/mqclient
   ```

   substituting `cntndnsn` with the name of your CD-ROM device.

2. Use the Sun Solaris **pkgadd** program, to install the software by carrying out the following procedure:
   a. Type `pkgadd -d /cdrom/mqclient`.
   b. You are prompted for a list of components to install. Select those you require. If you want to install the entire MQSeries client, select **all**.
   c. Press the Enter key.

For further information on using **pkgadd** to install software packages, see the Solaris documentation.

Translated messages

Messages in U.S. English are always available. If you require another of the languages supported by MQSeries for Sun Solaris, you **must** ensure that your NLSPATH environment variable includes the appropriate directory.

For example, to select messages in German, use the following:

```
export LANG=de
export NLSPATH=/usr/lib/locale/%L/LC_MESSAGES/%N
```

Verifying the installation

You can verify your MQSeries client and server installation using the supplied sample PUT and GET programs. These verify that your installation has been successfully completed and that the communication link is working.

Before starting make sure you have the same user ID on both client and server systems. For more information about security and the reasons for having the same user ID on both machines please see the *MQSeries Intercommunication* manual.

How does it work?

Instructions are given on how to use the supplied sample PUT and GET programs to verify that an MQSeries client has been installed correctly, by guiding you through the following tasks:

1. Setting up the server
2. Setting up the MQSeries client
3. Putting a message on the queue
4. Getting the message from the queue
5. Ending verification

The installation used for the example

These instructions assume that:

- The full MQSeries product has been installed on a UNIX server machine. If you are connecting the MQSeries for Sun Solaris client to a non UNIX MQSeries server, please see the *MQSeries Clients* book for verification instructions.
- The MQSeries for Sun Solaris client software has been installed on the client machine.
- TCP/IP is configured and initialized on both the server and the client machines.

What the example shows

The following example shows how to create a queue manager called *saturn.queue.manager*, a local queue called *QUEUE1*, and a server-connection channel called *CHANNEL1* on the server. It shows how to create the client-connection channel on the MQSeries client workstation; and how to use the sample programs to put a message onto a queue, and then get the message from the queue.

Note: MQSeries object definitions are case-sensitive. You must type the examples ***exactly*** as shown.

Security: The example does ***not*** address any client security issues. See the *MQSeries Clients* book for details if you are concerned with MQSeries client security issues.

Setting up the server

On the server, create a directory to hold working files (called mqverify, for example), and make this the current directory. Follow these steps to set up the server workstation:

1. Create a default queue manager called saturn.queue.manager by entering the following command at the command prompt:

   ```
   crtmqm -q saturn.queue.manager
   ```

2. Start the queue manager by entering the following command:

   ```
   strmqm
   ```

3. Start MQSeries commands (MQSC) by entering the following command:

   ```
   runmqsc
   ```

   MQSC does not provide a prompt, but should respond with the message:

   ```
   Starting MQSeries Commands
   ```

4. Create a local queue called QUEUE1 by entering the following command:

   ```
   DEFINE QLOCAL(QUEUE1)
   ```

5. Create a server-connection channel by entering the following command:

   ```
   DEFINE CHANNEL(CHANNEL1) CHLTYPE(SVRCONN) TRPTYPE(TCP) MCAUSER(' ')
   ```

6. Stop MQSC by typing `end` and pressing Enter.

7. Configure a listener to start the MQI channels.

   You can use either the standard UNIX TCP listener or the MQSeries listener to start the MQI channels. For performance reasons, the MQSeries listener is likely to be the better choice.

   Alternatively, to configure the MQSeries listener to start MQI channels, use the **runmqlsr** control command:

   ```
   runmqlsr -m queue_manager -t TCP -p port number
   ```

   To run the job in the background, add an ampersand character (&) to the end of the command.

   To configure the inetd daemon to start the MQI channels, you must be logged in as a superuser, or as root.

   a. Edit the file /etc/services. If you do not have the following line in that file, add it as shown:

      ```
      MQSeries       1414/tcp      # MQSeries channel listener
      ```

1414 is the default port number. If you are using a different port, specify its number instead.

b. Edit the file /etc/inetd.conf. If you do not have the following line in that file, add it as shown:

```
MQSeries stream tcp nowait mqm /opt/mqm/bin/amqcrsta amqcrsta
```

Note: If saturn.queue.manager is not the **default** queue manager on this workstation, you need to add -m saturn.queue.manager to the end of this line.

c. Run the command:

```
kill -1 inetd processid
```

Setting up the client

When an MQSeries application is run on the MQSeries client, the information it requires is:

- The name of the MQI channel
- The communications protocol
- The address of the server

You provide this information by defining a client-connection channel. The name used for this channel must be the name used for the server-connection channel defined on the server. In this example the MQSERVER environment variable is used to define the client-connection channel. This is the simplest way, although not the only one.

Before starting, type `ping server-address` (where server-address is the TCP/IP host name of the server) to confirm that your MQSeries client and server TCP/IP sessions have been initialized. You can use the network address, in the format n.n.n.n, in the **ping** command instead of the host name.

If the **ping** command fails, check that your TCP/IP software is correctly configured and has been started.

Defining a client-connection channel, using MQSERVER

Create a client-connection channel by setting the MQSERVER environment variable, as follows:

```
MQSERVER=CHANNEL1/TCP/'server-address(port)'
```

and then

```
export  MQSERVER
```

where:

CHANNEL1
Is the name of the server-connection channel already defined on the server.

TCP Is the communications protocol.

server-address
Is the TCP/IP host name of the server.

(port) Is optional and is the TCP/IP port number the server is listening on. If you do not give a port number MQSeries uses the one specified in the QM.INI file. If no value is specified in the QM.INI file, MQSeries uses the port number identified in the TCP/IP services file for the service name MQSeries. If this entry in the services file does not exist, a default value of 1414 is used. It is important that the port number used by the client and the port number used by the server listener program are the same.

Putting a message on the queue

First, ensure `/opt/mqm/samp/bin` is included in your PATH environment variable.

On the MQSeries client workstation, put a message on the queue using the AMQSPUTC sample program:

1. Enter the following command:

   ```
   amqsputc QUEUE1 saturn.queue.manager
   ```

   where saturn.queue.manager is the name of the queue manager on the server.

   The following message is displayed:

   ```
   Sample AMQSPUT0 start
   target qname is QUEUE1
   ```

2. Type some message text and press Enter twice.
   The following message is displayed:

   ```
   Sample AMQSPUT0 end
   ```

The message is now on the queue.

Getting the message from the queue

On the MQSeries client workstation, get a message from the queue using the amqsgetc sample program:

- From the directory containing the sample programs, enter the following command:

  ```
  amqsgetc QUEUE1 saturn.queue.manager
  ```

  Where saturn.queue.manager is the name of the queue manager on the server.

The message is removed from the queue and displayed.

Ending verification

The verification process is now complete.

You can stop the queue manager on the server by typing:

```
endmqm -w saturn.queue.manager
```

If you want to delete the queue manager on the server, enter

```
dltmqm saturn.queue.manager
```

Verifying the installation

Chapter 4. Applying maintenance to MQSeries for Sun Solaris

This chapter tells you how to apply maintenance to MQSeries for Sun Solaris.

Maintenance updates in the form of a Program Temporary Fix (PTF), also known as CSDs, are supplied on CD-ROM. They can also be downloaded from :

```
http://www.ibm.com/software/mqseries/
```

Attention

Do not have any queue managers operating during installation of maintenance on MQSeries for AIX®. To end all running queue managers issue the commands:

```
endmqm -i Qmname
endmqlsr -m Qmname
```

Space requirements

A PTF requires hard disk space for installation. In addition, the installation process requires an identical amount of disk space to save the previous level. For example, a 16 MB PTF requires 32 MB of space.

This allows a PTF to be removed, and the previous level to be automatically restored. If disk space is limited, the backup can be suppressed by creating an empty flag file called MQPTF_NOSAVE in the directory `/var/sadm/pkg`.

Note that if this option is used, the previous level will not be restored if a PTF is removed. The only way to restore a previous level in this instance is to reinstall the product and then to reapply a previous PTF image.

Applying the maintenance information

1. Mount the CD-ROM.
2. Install the software as follows:
   a. Type `pkgadd -d /cdrom/mq_solaris/mqm/patchname`.
   b. Press the Enter key.

For further information on using **pkgadd** to install software packages, see the Solaris documentation.

Restoring the previous service level

To restore the previous service level:

1. Log in as root, or use the command `su`.
2. Use the **pkgrm** command to remove the latest PTF from the system.

   For example, to remove PTF `U469913` issue the following command:

   ```
   pkgrm mqm.upd02
   ```

   Note: Any error messages displayed of the form `<shared pathname not removed>` can be ignored.

   Details of the **pkgrm** command can be found in the Solaris documentation, or by using the **man pkgrm** command.
3. If you have installed an MQI client, and the client was updated after installing the PTF that is being removed, then you ***must*** specifically update your MQI client installation again, after the PTF has been removed.

Chapter 5. Uninstalling MQSeries for Sun Solaris

Use the `pkgrm` program to uninstall MQSeries from your Sun Solaris system.

If for any reason the product was not properly installed then you will have to delete the files and directories contained in `/opt/mqm`.

Part 2. Getting started with MQSeries

Chapter 6. About MQSeries

This chapter introduces IBM MQSeries. It describes its basic functions and its relationships with operating systems, applications, and other middleware products.

Introduction

MQSeries is a communications system that provides assured, asynchronous, once-only delivery of data across a broad range of hardware and software platforms.

These characteristics make MQSeries the ideal infrastructure for application-to-application communication, and make it an appropriate solution whether the applications run on the same machine or on different machines that are separated by one or more networks.

MQSeries supports all the important communication protocols and even provides routes between networks that use different protocols. MQSeries bridges and gateway products allow easy access (with little or no programming) to many existing systems and application environments—for example, Lotus Notes™, Web browsers, Java applets, and many others.

The assured delivery capability reflects the many functions built in to MQSeries to ensure that data is not lost because of failures in the underlying system or network infrastructure. Assured delivery enables MQSeries to form the backbone of critical communication systems and to be entrusted with delivering high-value data. There are also options that allow you to select a less robust quality of service, where this is appropriate. For example, there might be circumstances where you might prefer faster delivery with less emphasis on assured delivery.

The asynchronous processing support in MQSeries means that the exchange of data between the sending and receiving applications is time independent. This allows the sending and receiving applications to be decoupled so that the sender can continue processing, without having to wait for the receiver to acknowledge that it has received the data. In fact, the target application does not even have to be running when the data is sent. Likewise, the entire network path between the sender and receiver may not need to be available when the data is in transit.

Once-only delivery of data is a vital consideration, particularly in financial and business applications where duplicate requests to move large sums of money from one account to another are precisely what you do not want to happen!

Messages, queues, and queue managers

The three fundamental concepts in MQSeries that you need to understand are:
- Messages
- Queues
- Queue managers

Messages

A *message* is a string of bytes that has meaning to the applications that use it. Messages are used for transferring data from one application to another (or to different parts of the same application). The applications can be running on the same platform, or on different platforms.

MQSeries messages have two parts; the *application data* and a *message descriptor*. The content and structure of the application data is defined by the application programs that use the data. The message descriptor identifies the message and contains other control information, such as the type of message and the priority assigned to the message by the sending application.

Queues

A *queue* is a data structure in which messages are stored. The messages may be put on, or got from, the queue by applications or by a queue manager as part of its normal operation.

Queues exist independently of the applications that use them. A queue can exist in main storage (if it is temporary), on disk or similar auxiliary storage (if it must be kept in case of recovery), or in both places (if it is currently being used, and must also be kept for recovery). Each queue belongs to a *queue manager*, which is responsible for maintaining it. The queue manager puts the messages it receives onto the appropriate queue.

Queues can exist either in your local system, in which case they are called *local queues*, or at another queue manager, in which case they are called *remote queues*.

Applications send to, and receive messages from, queues. For example, one application can put a message on a queue, and another application can get the message from the same queue.

Each queue has *queue attributes* that determine what happens when applications reference the queue. The attributes indicate:

- Whether applications can retrieve messages from the queue (get enabled)
- Whether applications can put messages onto the queue (put enabled)
- Whether access to the queue is exclusive to one application or shared between applications
- The maximum number of messages that can be stored on the queue at the same time (maximum queue depth)
- The maximum size of messages that can be put on the queue (maximum message size)

Queue managers

A queue manager provides queuing services to applications, and manages the queues that belong to it. It ensures that:

- Object attributes are changed according to the details received.
- Special events (such as instrumentation events or triggering) are generated when the appropriate conditions are met.
- Messages are put on the correct queue, as requested by the application. The application is informed if this cannot be done, and an appropriate reason code is given.

Each queue belongs to a single queue manager and is said to be a *local queue* to that queue manager. The queue manager to which an application is connected is said to be the local queue manager for that application. For the application, the queues that belong to its local queue manager are local queues. A *remote queue* is a queue that belongs to another queue manager. A *remote queue manager* is any queue manager other than the local queue manager. A remote queue manager may exist on a remote machine across the network or it may exist on the same machine as the local queue manager. MQSeries supports multiple queue managers on the same machine.

MQSeries configurations

In the simplest configurations, MQSeries is installed on a machine and a single queue manager is created. This queue manager then allows you to define queues. Local applications can then use these queues to exchange messages.

Communication by applications with queues managed by another queue manager requires *message channels* to be defined. It is not necessary to define a channel directly to the target queue manager and it is often appropriate to define one only to the next hop (that is, an intermediate queue manager). Message channels available from that queue manager will be used to deliver the message to the target queue manager (or even to a subsequent hop).

MQSeries configurations

More complex configurations can be created using a client-server structure. The MQSeries product can act as an MQSeries server to MQSeries clients. The clients and server do not need to be on the same platform. MQSeries supports a broad range of client platforms. The MQSeries products typically include clients for a variety of platforms. Additional MQSeries clients are available from the MQSeries Web site.

In a client-server configuration, the MQSeries server provides messaging and queuing services to the clients, as well as to any local applications. The clients are connected to the server through dedicated channels (known as *client channels*) for clients. This is a cost-effective deployment method because a server can support hundreds of clients with only a single copy of the MQSeries server product. However, the client channel must be continuously available whenever the MQSeries applications on the client are running. This contrasts with the message channels, which need not be continuously available to support MQSeries applications running on the server.

See "Channels" for more information.

MQSeries also supports the concept of *clusters* to simplify setup and operation. A cluster is a named collection of queue managers and any one queue manager can belong to none, one, or several such clusters. The queue managers in a cluster can exist on the same or different machines.

There are two major benefits from the use of clusters:

1. Communication between members of a cluster is greatly simplified, particularly because the channels required for exchanging messages are automatically defined and created as needed.
2. Some or all of the queues of participating queue managers can be defined as being cluster queues, which has the effect of making them automatically known and available to all other queue managers in the cluster.

See "Clusters" on page 43 for more information.

Channels

A channel provides a communication path to a queue manager. There are two types of channel: message channels and MQI channels.

A *message channel* provides a communication path between two queue managers on the same, or different, platforms. The message channel is used for transmitting messages from one queue manager to another, and shields the application programs from the complexities of the underlying networking protocols. A message channel can transmit messages in one direction only. Two message channels are required if two-way communication is required between two queue managers.

A *client channel* (also known as an *MQI channel*) connects an MQSeries client to a queue manager on a server machine and is bidirectional.

If you want to read more information about channels and how MQSeries uses them to communicate across the systems in your network, see the *MQSeries Intercommunication* book.

Clients and servers

MQSeries supports client-server configurations for MQSeries applications.

An *MQSeries client* is a part of the MQSeries product that is installed on a machine to accept MQSeries calls from applications and pass them to an *MQSeries server* machine. There they are processed by a queue manager. Typically, the client and server reside on different machines, but they can also exist on the same machine.

An *MQSeries server* is a queue manager that provides queuing services to one or more clients. All the MQSeries objects (for example, queues) exist only on the queue manager machine (that is, on the MQSeries server machine). A server can support local MQSeries applications as well.

The difference between an MQSeries server and an ordinary queue manager is that the MQSeries server can support MQSeries clients, and each MQSeries client application has a dedicated communication link with the MQSeries server.

For more information about client support, see the *MQSeries Clients* book.

Clusters

A cluster is a named collection of queue managers.

Clusters require that at least one of the queue managers in the cluster be defined as a *repository* (that is, a place where the shared cluster information can be held). More typically, two or more such repositories are usually designated to provide continued availability in the case of system failure. MQSeries makes sure that the information in the repositories is synchronized.

When a queue is defined as a cluster queue, it can be regarded as a public queue in that it is freely available to other queue managers in the cluster. This contrasts with noncluster queues, which are accessible only when a local definition of them is available. Thus, a non-cluster queue has the characteristics of a private queue, accessible only to those queue managers that have been configured to know about them.

Public queues with the same name in the same cluster are regarded as equivalent. If a message is sent to that queue name, MQSeries (by default)

sends it to any one of the instances, using a load-balancing algorithm. If you do not want this to happen, you can use the queue manager and queue name in the address, thus forcing the message to be delivered to a specific queue manager. Alternatively, you can replace the load-balancing routine with a different implementation. This is typical of MQSeries, in that there are many examples of where standard behavior can be changed by implementing user code in exits designed for this purpose.

You can read a full explanation in the *MQSeries Queue Manager Clusters* book.

MQSeries capabilities

MQSeries can be used to create many different types of solutions. Some exploit the platform support, or the bridge and gateway capabilities, to connect existing systems in an integrated way or to allow new applications to extract information from, or interchange information with, existing systems. Other solutions support business application servers, where a central pool of MQSeries applications can manage work sent across networks. Complex routing of information for workflow scenarios can be supported. Publish/subscribe or "send and forget" are other application scenarios that use different message flows. Load balancing and hot-standby systems can be built using the power and flexibility of MQSeries, which includes specific functions to support many of these diverse scenarios.

See the *MQSeries Application Programming Guide* for more information about writing MQSeries applications.

Transactional support

An application program may need to group a set of updates into a *unit of work*. Such updates are usually logically related and must all be successful for data integrity to be preserved. Data integrity would be lost if one update in the group succeeded while another failed. MQSeries supports transactional messaging.

A unit of work *commits* when it completes successfully. At this point all updates made within that unit of work are made permanent and irreversible. Alternatively, all updates are *backed out* if the unit of work fails. *Syncpoint coordination* is the process by which a unit of work is either committed or backed out with integrity.

A *local* unit of work is one in which the only resources updated are those of the MQSeries queue manager. Here, syncpoint coordination is provided by the queue manager itself, using a single-phase commit process.

A *global* unit of work is one in which resources belonging to other resource managers, such as XA-compliant databases, are also updated. Here, a

two-phase commit procedure must be used and the unit of work may be coordinated by the queue manager itself, or externally by another XA-compliant transaction manager such as IBM CICS®, IBM Transaction Server, IBM TXSeries™, Transarc Encina, or BEA Tuxedo.

When the queue manager coordinates global units of work itself it becomes possible to integrate database updates within MQSeries units of work. That is, a mixed MQSeries and SQL application can be written, and commands can be used to commit or roll back the changes to the queues and databases together.

The queue manager achieves this using a two-phase commit protocol. When a unit of work is to be committed, the queue manager first asks each participating database manager whether it is prepared to commit its updates. Only if all of the participants, including the queue manager itself, are prepared to commit, are all of the queue and database updates committed. If any participant cannot prepare its updates, the unit of work is backed out instead.

Full recovery support is provided if the queue manager loses contact with any of the database managers during the commit protocol. If a database manager becomes unavailable while it is in doubt (that is, it has been called to prepare but has yet to receive a commit or backout decision), the queue manager remembers the outcome of the unit of work until it has been successfully delivered. Similarly, if the queue manager terminates with incomplete commit operations outstanding, these are remembered when the queue manager restarts.

Instrumentation events

You can use MQSeries instrumentation events to monitor the operation of queue managers.

Instrumentation events cause special messages, called *event messages*, to be generated whenever the queue manager detects a predefined set of conditions. For example, a *Queue Full* event message is generated if: Queue Full events are enabled for a specified queue; an application issues an MQPUT call to put a message on that queue; and the call fails because the queue is full.

Other conditions that can give rise to instrumentation events include:
- A predefined limit for the number of messages on a queue being reached
- A queue not being serviced within a specified time
- A channel instance being started or stopped

If you define your event queues as remote queues, you can put all the event queues on a single queue manager (for those nodes that support instrumentation events). You can then use the events generated to monitor a network of queue managers from a single node.

Capabilities

MQSeries instrumentation events are categorized as follows:

Queue manager events
These are related to the definitions of resources within queue managers. For example, if an application attempts to open a queue but the associated user ID is not authorized to perform that operation, a queue manager event is generated.

Performance events
These are notifications that a threshold condition has been reached by a resource. For example, a queue depth limit has been reached or, following an MQGET request, a queue has not been serviced within a predefined period of time.

Channel events
These are reported by channels as a result of conditions detected during their operation. For example, a channel event is generated when a channel instance is stopped.

Message-driven processing

When they arrive on a queue, messages can automatically start an application, using a mechanism known as *triggering*. If necessary, the application can be stopped when the message or messages have been processed.

Programming MQSeries

MQSeries applications can be developed using a variety of programming languages and styles. Procedural and object-oriented programming is supported, depending on the MQSeries platform, using, for example, Visual Basic®, C, C++, Java, COBOL, and PL/I.

MQSeries function is logically divided into what is normally required by applications (such as putting messages on a queue) and what is necessary for administration (such as changing queue or queue manager definitions). Application function is known as the *MQI* (message queue interface). Administration function is known as the *MQAI* (message queuing administration interface). Applications can mix MQI and MQAI functionality, as required.

The administration functions can be implemented in two ways:

1. Most often, using MQAI language bindings
2. Sending messages to administration queues, to achieve the same results as with the MQAI, using programmable command formats (PCFs)

Chapter 7. Using the MQSeries command sets

This chapter introduces the command sets that can be used to perform system administration tasks on MQSeries objects.

Administration tasks include creating, starting, altering, viewing, stopping, and deleting MQSeries objects such as queue managers, queues, processes, channels, and namelists. To perform these tasks, you must select the appropriate command from one of the supplied command sets.

Introducing command sets

MQSeries provides three command sets for performing administration tasks:
- Control commands
- MQSC commands
- PCF commands

This section describes the command sets that are available. Some tasks can be performed using either a control command or an MQSC command, but other tasks can be performed using only one type of command. For a comparison of the facilities provided by the different types of command set, see the *MQSeries System Administration* book.

Control commands

Control commands fall into three categories:

- *Queue manager commands*, including commands for creating, starting, stopping, and deleting queue managers and command servers.
- *Channel commands*, including commands for starting and ending channels and channel initiators.
- *Utility commands*, including commands associated with authority management and conversion exits.

Using control commands

In MQSeries in UNIX environments, you enter control commands in a shell window. In these environments, control commands, including the command name itself, the flags, and any arguments, are case sensitive. For example, in the command:

```
crtmqm -u SYSTEM.DEAD.LETTER.QUEUE jupiter.queue.manager
```

- The command name must be **crtmqm**, not **CRTMQM**.
- The flag must be `-u`, not `-U`.

- The dead-letter queue is SYSTEM.DEAD.LETTER.QUEUE.
- The argument is specified as jupiter.queue.manager, which is different from JUPITER.queue.manager.

Therefore, take care to type the commands exactly as you see them in the examples.

The following list contains a brief description of each of the control commands. You can obtain help for the syntax of any of the commands by entering the command followed by a question mark. MQSeries responds by listing the syntax required for the selected command.

crtmqcvx (data conversion)
Creates a fragment of code that performs data conversion on data type structures.

crtmqm (create queue manager)
Creates a local queue manager and defines the default and system objects.

dltmqm (delete queue manager)
Deletes a specified queue manager.

dmpmqlog (dump log)
Dumps a formatted version of the MQSeries system log.

dspmqaut (display authority)
Displays the current authorizations to a specified object.

dspmqcsv (display command server)
Displays the status of the command server for the specified queue manager.

dspmqfls (display MQSeries files)
Displays the real file system name for all MQSeries objects that match a specified criterion.

dspmqtrc (display MQSeries formatted trace output)
Displays MQSeries formatted trace output.

dspmqtrn (display MQSeries transactions)
Displays details of in-doubt transactions.

endmqcsv (end command server)
Stops the command server on the specified queue manager.

endmqlsr
Ends a listener process.

endmqm (end queue manager)
Stops a specified local queue manager.

endmqtrc (end MQSeries trace)
Ends tracing for the specified entity or all entities.

rcdmqimg (record media image)
Writes an image of an MQSeries object, or group of objects, to the log for use in media recovery.

rcrmqobj (recreate object)
Recreates an object, or group of objects, from their images contained in the log.

rsvmqtrn (resolve MQSeries transactions)
Commits or backs out internally or externally coordinated in-doubt transactions.

runmqchi (run channel initiator)
Runs a channel initiator process.

runmqchl (run channel)
Runs either a Sender (SDR) or a Requester (RQSTR) channel.

runmqdlq (run dead-letter queue handler)
Starts the dead-letter queue (DLQ) handler, a utility that you can run to monitor and handle messages on a dead-letter queue.

runmqlsr (run listener)
Runs a listener process.

runmqsc (run MQSeries commands)
Issues MQSC commands to a queue manager.

runmqtrm (start trigger monitor)
Invokes a trigger monitor.

setmqaut (set/reset authority)
Changes the authorizations to an object or to a class of objects.

strmqcsv (start command server)
Starts the command server for the specified queue manager.

strmqm (start queue manager)
Starts a local queue manager.

strmqtrc (start MQSeries trace)
Enables tracing.

For more information about the syntax and purpose of control commands, see the *MQSeries System Administration* book.

MQSeries (MQSC) commands

You use the MQSeries (MQSC) commands to manage queue manager objects, including the queue manager itself, channels, queues, and process definitions. For example, there are commands to define, alter, display, and delete a specified queue.

When you display a queue, using the DISPLAY QUEUE command, you display the queue *attributes*. For example, the MAXMSGL attribute specifies the maximum length of a message that can be put on the queue. The command does not show you the messages on the queue.

For detailed information about each MQSC command, see the *MQSeries MQSC Command Reference* book.

Running MQSC commands

You run MQSC commands by invoking the control command **runmqsc**. You can run MQSC commands:

- Interactively by typing them at the keyboard
- As a sequence of commands from a text file

For more information about using MQSC commands, see the *MQSeries System Administration* book.

PCF commands

MQSeries programmable command format (PCF) commands allow administration tasks to be programmed into an administration program. In this way you can create queues and process definitions, and change queue managers, from a program. PCF commands cover the same range of functions that are provided by the MQSC facility. You can therefore write a program to issue PCF commands to any queue manager in the network from a single node. In this way, you can both centralize and automate administration tasks.

Note: Unlike MQSC commands, PCF commands and their replies are not in a text format that you can read.

For a complete description of the PCF data structures and how to implement them, see the *MQSeries Programmable System Management* book.

Working with queue managers

This section describes how you can perform operations on queue managers, such as creating, starting, stopping, and deleting them. MQSeries provides control commands for performing these tasks.

Before you can do anything with messages and queues, you must create at least one queue manager.

Creating a default queue manager

The following command:

- Creates a default queue manager called saturn.queue.manager
- Creates the default and system objects automatically
- Specifies the names of both a default transmission queue and a dead-letter queue

```
crtmqm -q -d MY.DEFAULT.XMIT.QUEUE -u
SYSTEM.DEAD.LETTER.QUEUE  saturn.queue.manager
```

where:

-q Indicates that this queue manager is the default queue manager.

-d MY.DEFAULT.XMIT.QUEUE
Is the name of the default transmission queue.

-u SYSTEM.DEAD.LETTER.QUEUE
Is the name of the dead-letter queue.

saturn.queue.manager
Is the name of this queue manager. This must be the last parameter specified on the **crtmqm** command.

For more information about these attributes, see the *MQSeries System Administration* book.

Starting a queue manager

Although you have created a queue manager, it cannot process commands or MQI calls until it has been started. Start the queue manager by typing in this command:

```
strmqm saturn.queue.manager
```

The **strmqm** command does not return control until the queue manager has started and is ready to accept connect requests.

Stopping a queue manager

To stop a queue manager, use the **endmqm** command. For example, to stop a queue manager called `saturn.queue.manager` use this command:

```
endmqm saturn.queue.manager
```

Quiesced shutdown

By default, the above command performs a *quiesced shutdown* of the specified queue manager. This may take a while to complete—a quiesced shutdown waits until all connected applications have disconnected.

Use this type of shutdown to notify applications to stop; you are not told when they have stopped.

You can specify the `-w` flag if you require confirmation that the queue manager has stopped. For example:

```
endmqm -w saturn.queue.manager
```

The command prompt does not return until the queue manager has stopped.

Immediate shutdown

An *immediate shutdown* allows any current MQI calls to complete, but any new calls fail. This type of shutdown does not wait for applications to disconnect from the queue manager. Use this as the normal way to stop the queue manager, optionally after a quiesce period.

For an immediate shutdown, the command is:

```
endmqm -i saturn.queue.manager
```

Preemptive shutdown

Do not use this method unless all other attempts to stop the queue manager using the **endmqm** command have failed. This method can have unpredictable consequences for connected applications.

If an immediate shutdown does not work, you must resort to a *preemptive shutdown*, specifying the `-p` flag. For example:

```
endmqm -p saturn.queue.manager
```

This stops all queue manager code immediately.

Deleting a queue manager

To delete a queue manager called `saturn.queue.manager`, first stop it, then use the following command:

```
dltmqm saturn.queue.manager
```

Note: Deleting a queue manager is a serious step, because you also delete all resources associated with that queue manager, including all queues and their messages, and all object definitions.

Working with MQSeries objects

This section describes briefly how to use MQSC commands to create, display, change, copy, and delete MQSeries objects.

You can use the MQSC facility interactively (by entering commands at the keyboard) or you can redirect the standard input device (stdin) to run a sequence of commands from a text file. The format of the commands is the same in both cases. The examples included here assume that you will be using the interactive method.

For more information about using MQSC commands, see the *MQSeries System Administration* book. For a complete description of the MQSC commands, see the *MQSeries MQSC Command Reference* book.

Before you can run MQSC commands, you must have created and started the queue manager that is going to run the commands. For more information see "Creating a default queue manager" on page 50.

Using the MQSC facility interactively

To start using the MQSC facility interactively, use the **runmqsc** command. Open a shell and enter:

```
runmqsc
```

A queue manager name has not been specified; therefore the MQSC commands will be processed by the default queue manager. Now type in any MQSC commands, as required. For example:

```
DEFINE QLOCAL (ORANGE.LOCAL.QUEUE)
```

Continuation characters must be used to indicate that a command is continued on the following line:

- A minus sign (–) indicates that the command is to be continued from the start of the following line.
- A plus sign (+) indicates that the command is to be continued from the first nonblank character on the following line.

Command input terminates with the final character of a nonblank line that is not a continuation character. You can also terminate command input explicitly by entering a semicolon (;). (This is especially useful if you accidentally enter a continuation character at the end of the final line of command input.)

Feedback from MQSC commands

When you issue commands from the MQSC facility, the queue manager returns operator messages that confirm your actions or tell you about the errors you have made. For example:

```
AMQ8006: MQSeries queue created
 .
 .
 .
AMQ8405: Syntax error detected at or near end of command segment below:-
Z
```

The first message confirms that a queue has been created; the second indicates that you have made a syntax error.

These messages are sent to the standard output device. If you have not entered the command correctly, refer to the *MQSeries MQSC Command Reference* book for the correct syntax.

Ending interactive input to MQSC

To end interactive input of MQSC commands, enter the MQSC END command:

```
END
```

Alternatively, you can use the EOF character `CTRL+D`.

If you are redirecting input from other sources, such as a text file, you do not have to do this.

Defining a local queue

For an application, the local queue manager is the queue manager to which the application is connected. Queues that are managed by the local queue manager are said to be local to that queue manager.

Use the MQSC command DEFINE QLOCAL to create a definition of a local queue and also to create the data structure that is called a queue. You can also modify the queue characteristics from those of the default local queue.

In this example, the queue we define, ORANGE.LOCAL.QUEUE, is specified to have these characteristics:

- It is enabled for gets, disabled for puts, and operates on a first-in-first-out (FIFO) basis.
- It is an 'ordinary' queue, that is, it is not an initiation queue or a transmission queue, and it does not generate trigger messages.
- The maximum queue depth is 1000 messages; the maximum message length is 2000 bytes.

The following MQSC command does this:

```
DEFINE QLOCAL (ORANGE.LOCAL.QUEUE) +
       DESCR('Queue for messages from other systems') +
       PUT (DISABLED) +
       GET (ENABLED) +
       NOTRIGGER +
       MSGDLVSQ (FIFO) +
       MAXDEPTH (1000) +
       MAXMSGL (2000) +
       USAGE (NORMAL);
```

Notes:

1. Most of these attributes are the defaults as supplied with the product. However, they are shown here for purposes of illustration. You can omit them if you are sure that the defaults are what you want or have not been changed. See also "Displaying default object attributes".
2. USAGE (NORMAL) indicates that this queue is not an initiation queue or a transmission queue.
3. If you already have a local queue on the same queue manager with the name ORANGE.LOCAL.QUEUE, this command fails. Use the REPLACE attribute if you want to overwrite the existing definition of a queue, but see also "Changing local queue attributes" on page 56.

Displaying default object attributes

When you define an MQSeries object, it takes any attributes that you do not specify from the default object. For example, when you define a local queue, the queue inherits any attributes that you omit in the definition from the default local queue, which is called SYSTEM.DEFAULT.LOCAL.QUEUE. The

default local queue is created automatically when you create the default queue manager. To see exactly what these attributes are, use the following command:

```
DISPLAY QUEUE (SYSTEM.DEFAULT.LOCAL.QUEUE)
```

Note: The syntax of this command is different from that of the corresponding **DEFINE** command.

You can selectively display attributes by specifying them individually. For example:

```
DISPLAY QUEUE (ORANGE.LOCAL.QUEUE) +
        MAXDEPTH +
        MAXMSGL +
        CURDEPTH;
```

This command displays the three specified attributes as follows:

```
AMQ8409: Display Queue details.
    QUEUE(ORANGE.LOCAL.QUEUE)
    MAXDEPTH(1000)
    MAXMSGL(2000)
    CURDEPTH(0)
```

CURDEPTH is the current queue depth; that is, the number of messages on the queue. This is a useful attribute to display, because by monitoring the queue depth you can ensure that the queue does not become full.

Copying a local queue definition

You can copy a queue definition using the LIKE attribute on the **DEFINE** command.

For example:

```
DEFINE QLOCAL (MAGENTA.QUEUE) +
       LIKE (ORANGE.LOCAL.QUEUE)
```

This command creates a queue with the same attributes as our original queue ORANGE.LOCAL.QUEUE, rather than those of the system default local queue.

You can also use this form of the **DEFINE** command to copy a queue definition, but substituting one or more changes to the attributes of the original. For example:

```
DEFINE QLOCAL (THIRD.QUEUE) +
       LIKE (ORANGE.LOCAL.QUEUE) +
       MAXMSGL(1024);
```

This command copies the attributes of the queue ORANGE.LOCAL.QUEUE to the queue THIRD.QUEUE, but specifies that the maximum message length on the new queue is to be 1024 bytes, rather than 2000.

Notes:

1. When you use the LIKE attribute on a **DEFINE** command, you are copying the queue attributes only. You are not copying the messages on the queue.
2. If you define a local queue, without specifying LIKE, it is the same as `DEFINE LIKE(SYSTEM.DEFAULT.LOCAL.QUEUE).`

Changing local queue attributes

You can change queue attributes in two ways, using either the **ALTER QLOCAL** command or the **DEFINE QLOCAL** command with the REPLACE attribute. In "Defining a local queue" on page 54, we defined the queue ORANGE.LOCAL.QUEUE. Suppose, for example, you wanted to increase the maximum message length on this queue to 10 000 bytes.

- Using the **ALTER** command:

  ```
  ALTER QLOCAL (ORANGE.LOCAL.QUEUE) MAXMSGL(10000)
  ```

  This command changes a single attribute, that of the maximum message length; all the other attributes remain the same.

- Using the **DEFINE** command with the REPLACE option, for example:

  ```
  DEFINE QLOCAL (ORANGE.LOCAL.QUEUE) MAXMSGL(10000) REPLACE
  ```

  This command changes not only the maximum message length, but all the other attributes, which are given their default values. The queue is now put enabled whereas previously it was put inhibited. Put enabled is the default, as specified by the queue SYSTEM.DEFAULT.LOCAL.QUEUE, unless you have changed it.

  If you decrease the maximum message length on an existing queue, existing messages are not affected. Any new messages, however, must meet the new criteria.

Clearing a local queue

To delete all the messages from a local queue called MAGENTA.QUEUE, use the following command:

```
CLEAR QLOCAL (MAGENTA.QUEUE)
```

You cannot clear a queue if:

- There are uncommitted messages that have been put on the queue under syncpoint.
- An application currently has the queue open.

Deleting a local queue

Use the MQSC command **DELETE QLOCAL** to delete a local queue. A queue cannot be deleted if it has uncommitted messages on it. However, if the queue has one or more committed messages, and no uncommitted messages, it can be deleted only if you specify the PURGE option. For example:

```
DELETE QLOCAL (PINK.QUEUE) PURGE
```

Specifying NOPURGE instead of PURGE ensures that the queue is not deleted if it contains any committed messages.

Browsing queues

MQSeries provides a sample queue browser that you can use to look at the contents of the messages on a queue. The browser is supplied in both source and executable formats.

The default file names and paths are:

Source

/opt/mqm/samp/amqsbcg0.c

Executable

/opt/mqm/samp/bin/amqsbcg

The sample requires two input parameters, the queue manager name and the queue name. For example:

```
amqsbcg ORANGE.LOCAL.QUEUE saturn.queue.manager
```

There are no defaults; both parameters are required.

Chapter 8. Using the MQSeries Internet Gateway

This chapter introduces the MQSeries Internet Gateway. It also explains how to get more information about using the MQSeries Internet Gateway.

The MQSeries Internet Gateway is one of the installable components on the MQSeries Server CD-ROM, and is also available from the MQSeries Web site.

The following Gateways are available:
- MQSeries Internet Gateway for AIX
- MQSeries Internet Gateway for HP-UX
- MQSeries Internet Gateway for Linux
- MQSeries Internet Gateway for OS/2®
- MQSeries Internet Gateway for OS/390® OpenEdition®
- MQSeries Internet Gateway for Sun Solaris
- MQSeries Internet Gateway for Windows NT®

Overview of MQSeries Internet Gateway

MQSeries Internet Gateway provides a bridge between the synchronous World Wide Web and asynchronous MQSeries applications. With the MQSeries Internet Gateway, Web server software and MQSeries together provide an Internet-connected Web browser with access to MQSeries applications. This means that enterprises can take advantage of the low-cost access to global markets provided by the Internet, while benefitting from the robust infrastructure and assured message delivery of MQSeries.

User interaction with the MQSeries Internet Gateway is through HTML fill-out form POST requests; MQSeries applications respond by returning HTML pages to the MQSeries Internet Gateway, via an MQSeries queue.

The MQSeries Internet Gateway supports the following Web server interfaces:
- Common Gateway Interface (CGI)
- Internet Connection Application Programming Interface (ICAPI)
- Internet Services Application Programming Interface (ISAPI)
- Netscape Connection Application Programming Interface (NSAPI)

Note that:
- HP-UX does not support NSAPI.
- Sun Solaris does not support ISAPI.
- Linux supports CGI only.

MQSeries Internet Gateway documentation

The MQSeries product family Web site is at:

```
http://www.ibm.com/software/mqseries/
```

The following documentation is accessible from this Web site:

- *Getting Started with MQSeries Internet Gateway*. This is the starting point for the download and installation of MQSeries Internet Gateway.
- *MQSeries Internet Gateway User's Guide*. This is the main documentation for users of the MQSeries Internet Gateway.

Chapter 9. Obtaining Additional Information

This chapter describes the documentation for MQSeries for Sun Solaris. It starts with a list of the publications, and then discusses:

- "Hardcopy Books"
- "Online Information" on page 62

MQSeries for Sun Solaris is described in the following books:

Table 3. MQSeries for Sun Solaris books

Order Number	Title
	Sun Solaris Specific Books
GC33-1870	*MQSeries for Sun Solaris Quick Beginnings*
	MQSeries Family Books
GC34-5761	*MQSeries V5.2 Release Guide*
SC33-1872	*MQSeries Intercommunication*
SC34-5349	*MQSeries Queue Manager Clusters*
GC33-1632	*MQSeries Clients*
SC33-1873	*MQSeries System Administration*
SC33-1369	*MQSeries MQSC Command Reference*
SC33-1482	*MQSeries Programmable System Management*
SC34-5390	*MQSeries Administration Interface Programming Guide and Reference*
GC33-1876	*MQSeries Messages*
SC33-0807	*MQSeries Application Programming Guide*
SC33-1673	*MQSeries Application Programming Reference*
SX33-6095	*MQSeries Programming Interfaces Reference Summary*
SC33-1877	*MQSeries Using C++*

Hardcopy Books

The book that you are reading now is *MQSeries for Sun Solaris, V5.2 Quick Beginnings*. This book and the *MQSeries V5.2 Release Guide* are the only books that are supplied in hardcopy with the product. However, all books listed in Table 3 are available for you to order or print.

You can order publications from the IBMLink™ Web site at:

```
http://www.ibm.com/ibmlink
```

In the United States, you can also order publications by dialing **1-800-879-2755**.

In Canada, you can order publications by dialing **1-800-IBM-4YOU (1-800-426-4968).**

For further information about ordering publications contact your IBM authorized dealer or marketing representative.

For information about printing books, see "PDF" on page 63.

Online Information

This section describes:

Publications supplied with the product

On the product CD-ROM there is a directory called `books`. The books directory contains MQSeries books in HTML and PDF formats. To access them point your Web browser to `books/start.htm`.

HTML

You can view the MQSeries online documentation in HTML format directly from the CD-ROM. All books except for the *MQSeries Programming Interfaces Reference Summary* are available in U.S. English and also in some or all of the following national languages:

- French
- German
- Italian
- Japanese
- Korean
- Spanish
- Simplified Chinese

When you read the books in HTML, you can follow hypertext links from one book to another. If you are reading translated books and link to a book that is not available in your national language, the U.S. English version of the book will be opened instead.

PDF

A PDF (Portable Document Format), corresponding to each hardcopy book, is available on the CD-ROM. You can read PDFs using Adobe Acrobat Reader. Also, you can download them to your own file system, or you can print them on a PostScript printer. If you have a Web browser, you can access the PDFs on the product CD-ROM by pointing your browser to `books/start.htm`.

The PDFs are available in U.S. English and also in some or all of the following national languages:

- French
- German
- Italian
- Japanese
- Korean
- Spanish
- Simplified Chinese

To find out which ones are available in your language, look for the appropriate directory on the CD-ROM. The PDFs are in a subdirectory called ll_LL, where ll_LL is one of the following:

- en_US (English)
- fr_FR (French)
- de_DE (German)
- it_IT (Italian)
- ja_JP (Japanese)
- ko_KR (Korean)
- es_ES (Spanish)
- zh_CN (Simplified Chinese)

Within these directories, you can find the complete set of PDFs that are available. Table 4 shows the file names used for the PDF files.

Table 4. MQSeries publications – file names

Book	File Name
MQSeries for Sun Solaris Quick Beginnings	AMQDAC03
MQSeries V5.2 Release Guide	AMQZAY00
MQSeries Intercommunication	CSQZAE04
MQSeries Queue Manager Clusters	CSQZAH02
MQSeries Clients	CSQZAF04
MQSeries System Administration	AMQZAG01
MQSeries MQSC Command Reference	CSQZAJ04

Table 4. MQSeries publications – file names (continued)

Book	File Name
MQSeries Programmable System Management	CSQZAI03
MQSeries Administration Interface Programming Guide and Reference	CSQZAT01
MQSeries Messages	AMQZAO01
MQSeries Application Programming Guide	CSQZAL04
MQSeries Application Programming Reference	CSQZAK04
MQSeries Programming Interfaces Reference Summary	CSQZAM04
MQSeries Using C++	AMQZAN03

HTML and PDF Books on the World Wide Web

The MQSeries books are available on the World Wide Web as well as on the product CD-ROM. They are available in PDF and HTML format. The MQSeries product family Web site is at:

```
http://www.ibm.com/software/mqseries/
```

By following links from this Web site you can:
- Obtain latest information about the MQSeries product family.
- Access the MQSeries books in HTML and PDF formats.
- Download MQSeries SupportPacs.

BookManager CD-ROMs

The MQSeries library is supplied in IBM BookManager® format on a variety of online library collection kits, including the *Transaction Processing and Data* collection kit, SK2T-0730. You can view the softcopy books in IBM BookManager format using the following IBM licensed programs:

BookManager READ/2
BookManager READ/6000
BookManager READ/DOS
BookManager READ/MVS
BookManager READ/VM
BookManager READ for Windows®

Online Help

Man pages are provided for all API calls, MQSC commands, and relevant control commands including **crtmqm**, **strmqm**, and **endmqm**.

Part 3. Appendixes

Appendix A. Sample MQI programs and MQSC files

MQSeries for Sun Solaris provides a set of short sample MQI programs and MQSC command files. You can use these directly or modify them for experimental purposes.

MQSC command file samples

Table 5 lists the MQSC command file samples. These are simply ASCII text files containing MQSC commands. You can invoke the **runmqsc** command against each file in turn to create the objects specified in the file.

By default, these files are located in directory **/opt/mqm**/samp.

Table 5. MQSC command files

File name	Purpose
amqscic0.tst	Defines objects for use in the sample CICS transaction.
amqscos0.tst	Creates a set of MQI objects for use with the C and COBOL program samples.
amqmdefs.tst	Defines objects for the administration application sample.

C and COBOL program samples

Table 6 lists the sample MQI source files. By default, the source files are in **/opt/mqm**/samp and the compiled versions in directory **/opt/mqm**/samp/bin. To find out more about what the programs do and how to use them, see the *MQSeries Application Programming Guide*.

Table 6. Sample programs - source files

C	COBOL	Purpose
amqsbcg0.c	–	Reads and then outputs both the message descriptor and message context fields of all the messages on a specified queue.
amqsecha.c	amqmechx.cbl amqiechx.cbl	Echoes a message from a message queue to the reply-to queue. Can be run as a triggered application program.

Table 6. Sample programs - source files (continued)

C	COBOL	Purpose
amqsgbr0.c	amq0gbr0.cbl	Writes messages from a queue to `stdout`, leaving the messages on the queue. Uses MQGET with the browse option.
amqsget0.c	amqminqx.cbl amqiinqx.cbl	Removes the messages from the named queue (using MQGET) and writes them to `stdout`.
amqsinqa.c	amqminqx.cbl	Reads the triggered queue; each request read as a queue name; responds with information about that queue.
amqsput0.c	amq0put0.cbl	Copies `stdin` to a message and then puts this message on a specified queue.
amqsreq0.c	amq0req0.cbl	Puts request messages on a specified queue and then displays the reply messages.
amqsseta.c	amqmsetx.cbl amqisetx.cbl	Inhibits puts on a named queue and responds with a statement of the result. Runs as a triggered application.
amqstrg0.c	–	A trigger monitor that reads a named initiation queue and then starts the program associated with each trigger message. Provides a subset of the full triggering function of the supplied **runmqtrm** command.
amqsvfc0.c	–	A sample C skeleton of a Data Conversion exit routine.
amqsptl0.c	amq0ptl0.cbl	Putting messages to a distribution list.
amqsprma.c	–	Putting reference messages to a queue.
amqsgrma.c	–	Getting reference messages from a queue.
amqsxrma.c	–	Reference message channel exit.
Note: You can create the objects required by these samples using the MQSC command file amqscos0.tst.		

Supporting CICS and Encina for transaction processing

The samples include a CICS transaction and some associated headers and initialization programs.

Table 7. Samples for transaction processing with CICS and Encina

File name	Purpose
amqzscix.c	CICS initialization program
amqscic0.ccs	Sample CICS program
amqzscgx.c	GLUE program for CICS for Solaris
amqscih0.h	Header file for CICS transaction sample amqscic0
Note: You can create objects to support transaction processing using the MQSC command file amqscic0.tst.	

Supporting Tuxedo for transaction processing

The samples include client transactions and some associated definitions and configuration files.

Table 8. Samples for transaction processing with Tuxedo

File name	Purpose
amqstxsx.c	Sample server
amqstxgx.c	Sample GET client application
amqstxpx.c	Sample PUT client application
amqstxvx.flds	Field definition
ubbstxcx.cfg	Configuration file

Supporting databases

The database samples are located in the `xatm` subdirectory within the samples directory.

Table 9. Sample programs - databases

C	COBOL	Purpose
amqsxas0.c		Updates a single database within an MQSeries unit of work. Use only with amqsxas0.sgc
amqsxag0.c	amq0xag0.cbl	amqsxag0.c, with amqsxab0.sqc and amqsxaf0.sqc, or amq0xag0.cbl, with amq0xab0.sqb and amq0xaf0.sqb, updates two databases within an MQSeries unit of work.

Miscellaneous tools

These tool files are provided to support the formatter and code conversion.

Table 10. Miscellaneous files

File name	Location	Purpose
amqtrc.fmt	/opt/mqm/lib	Defines MQSeries trace formats.
ccsid.tbl	/var/mqm/conv/table	Edit this file to add any newly supported CSSID values to your MQSeries system.

Appendix B. Code sets supported on MQSeries for Sun Solaris

MQSeries for Sun Solaris supports most of the code sets used by the locales – that is, the subsets of the user's environment that define the conventions for a specific culture – that are provided as standard on Sun Solaris.

If the locale is not set, the value of the LANG environment variable is used. If neither the locale nor LANG environment variable is set the CCSID used is 819 - the ISO 8859-1 code set.

Note: Not all the locales listed below are supported by all versions of Solaris.

See "Migration to Euro support" on page 79 for information on support for the euro character.

The CCSID (Coded Character Set Identifier) used in MQSeries to identify the code set used for the message and message header data is obtained by analyzing the LC_CTYPE environment variable.

Table 11 shows the locales and the CCSIDs that are registered for the code set used by the locale.

Table 11. Locales and CCSIDs

Locale	Language	code set	CCSID
C	English	ISO8859-1	819
ar	Arabic	ISO8859-6	1089
ar_AA	Arabic	ISO8859-6	1089
ar_EY	Arabic - Eygpt	ISO8859-6	1089
bg	Bulgarian	ISO8859-5	915
bg_BG	Bulgarian	ISO8859-5	915
cs	Czech	ISO8859-2	912
cs_CZ	Czech	ISO8859-2	912
cz	Czech	ISO8859-2	912

Table 11. Locales and CCSIDs (continued)

Locale	Language	code set	CCSID
da	Danish	ISO8859-1	819
da.ISO8859-15	Danish	ISO8859-15	923
da_DK	Danish	ISO8859-1	819
da_DK.ISO8859-15	Danish	ISO8859-15	923
de	German	ISO8859-1	819
de.ISO8859-15	German	ISO8859-15	923
de.UTF-8	German	UTF-8	1208
de_DE	German	ISO8859-1	819
de_DE.ISO8859-15	German	ISO8859-15	923
de_DE.UTF-8	German	UTF-8	1208
de_AT	German - Austria	ISO8859-1	819
de_AT.ISO8859-15	German - Austria	ISO8859-15	923
de_CH	German - Switzerland	ISO8859-1	819
el	Greek	ISO8859-7	813
el_GR	Greek	ISO8859-7	813
el_GR.sun_eu_greek	Greek	sun_eu_greek	4090
en	English - United Kingdom	ISO8859-1	819
en_GB	English - United Kingdom	ISO8859-1	819
en_GB.ISO8859-15	English - United Kingdom	ISO8859-15	923
en_UK	English - United Kingdom	ISO8859-1	819
en_EU.ISO8859-15	English - Europe	ISO8859-15	923
en_EU.UTF-8	English - Europe	UTF-8	1208
en_AU	English - Australia	ISO8859-1	819
en_CA	English - Canada	ISO8859-1	819
en_IE	English - Ireland	ISO8859-1	819
en_IE.ISO8859-15	English - Ireland	ISO8859-15	923
en_NZ	English - New Zealand	ISO8859-1	819

Table 11. Locales and CCSIDs (continued)

Locale	Language	code set	CCSID
en_US	English - USA	ISO8859-1	819
en_US.ISO8859-15	English - USA	ISO8859-15	923
en_US.UTF-8	English - USA	UTF-8	1208
es	Spanish - Spain	ISO8859-1	819
es.ISO8859-15	Spanish - Spain	ISO8859-15	923
es.UTF-8	Spanish - Spain	UTF-8	1208
es_ES	Spanish - Spain	ISO8859-1	819
es_ES.ISO8859-15	Spanish - Spain	ISO8859-15	923
es_ES.UTF-8	Spanish - Spain	UTF-8	1208
es_AR	Spanish - Argentina	ISO8859-1	819
es_BO	Spanish - Bolivia	ISO8859-1	819
es_CL	Spanish - Chile	ISO8859-1	819
es_CO	Spanish - Columbia	ISO8859-1	819
es_CR	Spanish - Costa Rica	ISO8859-1	819
es_EC	Spanish - Equador	ISO8859-1	819
es_GT	Spanish - Guatemala	ISO8859-1	819
es_MX	Spanish - Mexico	ISO8859-1	819
es_NI	Spanish - Nicaragua	ISO8859-1	819
es_PA	Spanish - Panama	ISO8859-1	819
es_PE	Spanish - Peru	ISO8859-1	819
es_PY	Spanish - Paraguay	ISO8859-1	819
es_SV	Spanish - El Salvador	ISO8859-1	819
es_UY	Spanish - Uruguay	ISO8859-1	819
es_VE	Spanish - Venezuela	ISO8859-1	819
et	Estonian	ISO8859-1	819
et_EE	Estonian	ISO8859-15	923

Table 11. Locales and CCSIDs (continued)

Locale	Language	code set	CCSID
fi	Finnish	ISO8859-1	819
fi.ISO8859-15	Finnish	ISO8859-15	923
fi_FI	Finnish	ISO8859-1	819
fi_FI.ISO8859-15	Finnish	ISO8859-15	923
fr	French - France	ISO8859-1	819
fr.ISO8859-15	French - France	ISO8859-15	923
fr.UTF-8	French - France	UTF-8	1208
fr_FR	French - France	ISO8859-1	819
fr_FR.ISO8859-15	French - France	ISO8859-15	923
fr_FR.UTF-8	French - France	UTF-8	1208
fr_BE	French - Belgium	ISO8859-1	819
fr_BE.ISO8859-15	French - Belgium	ISO8859-15	923
fr_CA	French - Canada	ISO8859-1	819
fr_CH	French - Switzerland	ISO8859-1	819
he	Hebrew	ISO8859-8	916
he_IL	Hebrew	ISO8859-8	916
hr	Croatian	ISO8859-2	912
hr_HR	Croatian	ISO8859-2	912
hu	Hungarian	ISO8859-2	912
hu_HU	Hungarian	ISO8859-2	912
is	Icelandic	ISO8859-1	819
is_IS	Icelandic	ISO8859-1	819
it	Italian - Italy	ISO8859-1	819
it.ISO8859-15	Italian - Italy	ISO8859-15	923
it.UTF-8	Italian - Italy	UTF-8	1208
it_IT	Italian - Italy	ISO8859-1	819
it_IT.ISO8859-15	Italian - Italy	ISO8859-15	923
it_IT.UTF-8	Italian - Italy	UTF-8	1208

Table 11. Locales and CCSIDs (continued)

Locale	Language	code set	CCSID
it_CH	Italian - Switzerland	ISO8859-1	819
ja	Japanese	eucJP	5050
ja_JP	Japanese	eucJP	5050
ja_JP.PCK	Japanese	PCK	943
ja_JP.UTF-8	Japanese	UTF-8	1208
ko	Korean	eucKR	970
ko.UTF-8	Korean	UTF-8	1208
ko_KR	Korean	eucKR	970
lt	Lithuanian	ISO8859-13	921
lt_LT	Lithuanian	ISO8859-13	921
lv	Latvian	ISO8859-13	921
lv_LV	Lathvian	ISO8859-13	921
mk	Macedonian	ISO8859-5	915
mk_MK	Macedonian	ISO8859-5	915
nl	Dutch - Netherlands	ISO8859-1	819
nl.ISO8859-15	Dutch - Netherlands	ISO8859-15	923
nl_NL	Dutch - Netherlands	ISO8859-1	819
nl_NL.ISO8859-15	Dutch - Netherlands	ISO8859-15	923
nl_BE	Dutch - Belgium	ISO8859-1	819
nl_BE.ISO8859-15	Dutch - Belgium	ISO8859-15	923
no	Norwegian	ISO8859-1	819
no_NO	Norwegian	ISO8859-1	819
nr	Bosnian	ISO8859-2	912
pl	Polish	ISO8859-2	912
pl_PL	Polish	ISO8859-2	912
POSIX	English	ISO8859-1	819

Table 11. Locales and CCSIDs (continued)

Locale	Language	code set	CCSID
pt	Portuguese	ISO8859-1	819
pt.ISO8859-15	Portuguese	ISO8859-15	923
pt_PT	Portuguese	ISO8859-1	819
pt_PT.ISO8859-15	Portuguese	ISO8859-15	923
pt_BR	Portuguese - Brazil	ISO8859-1	819
ro	Romanian	ISO8859-2	912
ro_RO	Romanian	ISO8859-2	912
ru	Russian	ISO8859-5	915
ru.KOI8-R	Russian	KOI8-R	878
ru_RU	Russian	ISO8859-5	915
ru_RU.KOI8-R	Russian	KOI8-R	878
ru_RU.ANSI1251	Russian	ANSI1251	1251
ru_SU	Russian	ISO8859-5	915
sh_BA	Bosnian	ISO8859-2	912
sl	Slovene	ISO8859-2	912
sl_SI	Slovene	ISO8859-2	912
sk	Slovak	ISO8859-2	912
sk_SK	Slovak	ISO8859-2	912
sq_AL	Albanian	ISO8859-2	912
sr	Serbian	ISO8859-5	915
sr_SP	Serbian	ISO8859-5	915
sr_YU	Serbian	ISO8859-5	915
sv	Swedish	ISO8859-1	819
sv.ISO8859-15	Swedish	ISO8859-15	923
sv.UTF-8	Swedish	UTF-8	1208
sv_SE	Swedish	ISO8859-1	819
sv_SE.ISO8859-15	Swedish	ISO8859-15	923
sv_SE.UTF-8	Swedish	UTF-8	1208

Table 11. Locales and CCSIDs (continued)

Locale	Language	code set	CCSID
th	Thailand	TIS620.2533	874
th_TH	Thailand	TIS620.2533	874
tr	Turkish	ISO8859-9	920
tr_TR	Turkish	ISO8859-9	920
zh	Simplified Chinese	eucCN	1383
zh.GBK	Simplified Chinese	GBK	1386
zh.UTF-8	Simplified Chinese	UTF-8	1208
zh_TW	Traditional Chinese	eucTW	964
zh_TW.UTF-8	Traditional Chinese	UTF-8	1208
zh_TW.BIG5	Traditional Chinese	BIG5	950

For further information listing inter-platform support for these locales, see the *MQSeries Application Programming Reference* book.

Migration to Euro support

To use the *euro* character with MQSeries, first install any operating system updates necessary to display the euro character.

Now modify your MQSeries system:

- Edit the existing CCSID.TBL file, in `/var/mqm/conv/table/`, to enable the new euro version of the coded character set identifier (CCSID). To do this, remove the first # symbol from the required line of the **CCSID Mapping** section of the CCSID.TBL file. When you have done this, all new queue managers you create will adopt the new euro CCSID.

  Note: If you want to create a new queue manager with a CCSID that supports the euro character, select a euro-supporting locale. For more information refer to the MQSeries Web site at:

  `http://www.ibm.com/software/mqseries/`

- To modify any existing queue managers that do not support the euro character, follow this procedure:

  1. Enable MQSeries (MQSC) commands by typing `runmqsc`.
  2. Record the existing queue manager CCSID, with the MQSC command:

     ```
     DISplay QMGR CCSID
     ```

3. Change the CCSID to the euro support CCSID, with the MQSC command: `ALTer QMGR CCSID` *(no. of ccsid)*.
4. Stop the MQSC commands by typing `END`.
5. Stop the queue manager.
6. Restart the queue manager and any channels it uses by typing `strmqm`.

Now any new message issued using the queue manager CCSID uses the new euro CCSID. All messages now received using MQGET with conversion and requesting the queue manager CCSID to be used, are converted into the euro CCSID. CCSIDs and object text (for example descriptions, definitions, and exit names) from existing messages are not changed.

Now modify your applications to support the euro character. If these use hard coded CCSIDs, ensure they now use the new euro CCSID.

Appendix C. Notices

This information was developed for products and services offered in the United States. IBM may not offer the products, services, or features discussed in this information in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this information. The furnishing of this information does not give you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make

improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:

IBM United Kingdom Laboratories,
Mail Point 151,
Hursley Park,
Winchester,
Hampshire,
England
SO21 2JN.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

Trademarks

The following are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

AIX	AS/400	BookManager
CICS	DB2	First Failure Support Technology (FFST)
IBM	IBMLink	MQSeries
OpenEdition	OS/2	OS/390
SupportPac	TXSeries	VSE/ESA
WebSphere		

Lotus Notes and Domino are trademarks of Lotus Development Corporation in the United States, or other countries, or both.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States and/or other countries.

ActiveX, Microsoft, Visual Basic, Windows and Windows NT are registered trademarks of Microsoft Corporation in the United States and/or other countries.

UNIX is a registered trademark in the United States and/or other countries licensed exclusively through X/Open Company Limited.

Other company, product, and service names may be trademarks or service marks of others.

Index

R

S

T

U

V

W

Y

Sending your comments to IBM

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book.

Please limit your comments to the information in this book and the way in which the information is presented.

To make comments about the functions of IBM products or systems, talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate, without incurring any obligation to you.

You can send your comments to IBM in any of the following ways:

- By mail, to this address:

  User Technologies Department (MP095)
  IBM United Kingdom Laboratories
  Hursley Park
  WINCHESTER,
  Hampshire
  SO21 2JN
  United Kingdom

- By fax:
  - From outside the U.K., after your international access code use 44–1962–870229
  - From within the U.K., use 01962–870229
- Electronically, use the appropriate network ID:
  - IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
  - IBMLink: HURSLEY(IDRCF)
  - Internet: idrcf@hursley.ibm.com

Whichever method you use, ensure that you include:

- The publication title and order number
- The topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.

Printed in the United States of America
on recycled paper containing 10%
recovered post-consumer fiber.

GC33-1870-03

Spine information:

MQSeries® for Sun Solaris

MQSeries for Sun Solaris V5.2 Quick Beginnings

Version 5.2