MQSeries for ptx

(formerly DYNIX/ptx)

Quick Beginnings

Version 5.0

Second Edition – December, 2000

Note
Before using this information and the product it supports, be sure to read the general information under Appendix E, "Notices".

Second edition (December 2000)

This edition applies to MQSeries for ptx Version 5.0 and to any subsequent releases and modifications until otherwise indicated in new editions.

If you want to make comments regarding this publication please send them to one of the folowing:

Mail	**email**	**Fax**
Willow Technology, Inc. Publications Dept. 900 Lafayette Street, Suite 604 Santa Clara, CA 95050-4967 U.S.A	support@willowtech.com	+1.408.296.7700

When you send information to Willow Technology, you grant Willow Technology a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

About this manual

This manual refers to the MQSeries for ptx Version 5.0 product produced by Willow Technology.

This product is part of the MQSeries family of products. These products provide application programming services that enable application programs to communicate with each other using *message queues*. This form of communication is referred to as *commercial messaging*. The applications involved can exist on different nodes on a wide variety of machine and operating system types. They use a common application programming interface, called the Message Queuing Interface or MQI, so that programs developed on one platform can readily be transferred to another.

This manual describes the system administration aspects of Willow Technology MQSeries for ptx and the services it provides to support commercial messaging in a ptx environment. This includes managing the queues that applications use to receive their messages, and ensuring that applications have access to the queues that they require.

Who this manual is for?

Primarily, this manual is for system administrators, and system programmers who manage the configuration and administration tasks for MQSeries. It is also useful to application programmers who must have some understanding of MQSeries administration tasks.

What you need to know to understand this manual

To use this manual, you should have a good understanding of the ptx operating system, and utilities associated with it. You do not need to have worked with message queuing products before, but you should have an understanding of the basic concepts of message queuing.

Part 1. Guidance Section

Chapter 1. About MQSeries

This chapter introduces MQSeries and describes the basic concepts of MQSeries and messaging. It contains basic explanations of the following topics:

- "Message Queuing"
- "Messages and Queues"
- "MQSeries Objects"
- "Clients and Servers"
- "Instrumentation Events"
- "Transactional Support"

Message Queuing

MQSeries enables applications to use message queuing to participate in message-driven processing. Applications can communicate across different platforms by using the appropriate message queuing software products. The applications are shielded from the mechanics of the underlying communications.

MQI – a Common Application Programming Interface

All MQSeries products implement a common application programming interface (message queue interface or MQI), regardless of the platform on which the applications are run. The calls made by the applications and the messages they exchange are common. This makes it much easier to write and maintain applications than it is when using traditional methods. It also makes it easier to port applications from one platform to another.

The MQI is described in detail in the *MQSeries Application Programming Reference manual*.

Time-Independent Applications

With message queuing, the exchange of messages between the sending and receiving programs is time independent. This means that the sending and receiving applications are decoupled so that the sender can continue processing without having to wait for the receiver to acknowledge the receipt of the message. In fact, the target application does not even have to be running when the message is sent. It can retrieve the message after it is started.

Message-Driven Processing

On arriving on a queue, messages can automatically start an application using a mechanism known

as *triggering*. If necessary, the applications can be stopped when the message or messages have been processed.

Messages and Queues

Messages and queues are the basic components of a message queuing system.

What is a Message?

A *message* is a string of bytes that has meaning to the applications that use it. Messages are used for transferring information 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.

What is a Queue?

A *queue* is a data structure that stores messages. The messages may be put on 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 and receive messages using MQI calls. For example, one application can put a message on a queue, and another application can retrieve the message from the same queue.

MQSeries Objects

An MQSeries object is a recoverable resource managed by MQSeries. Many of the tasks described in this chapter involve manipulating the following types of MQSeries object:

- Queue managers
- Queues
- Process definitions
- Channels

For system administrators, commands are available to manipulate objects. Default objects are

created for you when you create a queue manager.

Each object has a *name* associated with it and can be referenced in MQSeries commands and MQI calls by that name. Names must be unique within each of the object types. For example, you can have a queue and a process with the same name, but you cannot have two queues with the same name.

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 commands received.
- Special events such as trigger events or instrumentation events are generated when the appropriate conditions are met.
- Messages are put on the correct queue, as requested by the application making the MQPUT call. 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 simply 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.

Queues

A queue is an MQSeries object that can store messages. Each queue has *queue attributes* that determine what happens when applications reference the queue in MQI calls. 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)

Process Definitions

A *process definition* object defines an application that is to be started in response to a trigger event on an MQSeries queue manager.

A trigger event is a logical combination of conditions that is detected by a queue manager. For example, a trigger event may be generated when the number of messages on a queue reaches a predefined level. This event causes the queue manager to put a trigger message on a specified

initiation queue. This trigger message is retrieved by a *trigger monitor*, a special application that monitors an initiation queue. The trigger monitor then starts up the application program that was specified in the trigger message.

If a queue manager is to use triggering, at least one initiation queue must be defined for that queue manager.

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

Channels

A channel provides a communication path. 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 the transmission of 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. If two-way communication is required between two queue managers, two message channels are required.

An *MQI channel* connects an MQSeries client to a queue manager on a server machine. It is for the transfer of MQI calls (for example, MQPUT) and responses only and is bidirectional. A channel definition exists for each end of the link. On some platforms, some types of MQI channel can be defined automatically.

For more information on channels and how to use them, see the *MQSeries Intercommunication* manual.

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 MQI calls from applications and pass them to an *MQI 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 *MQI 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 MQI server machine. A server can support local MQSeries applications as well.

The difference between an MQI server and an ordinary queue manager is that the MQIserver can support MQI clients, and each MQI client has a dedicated communications link with the MQI server. For more information about creating channels for clients and servers, see the *MQSeries Intercommunication* manual. For information about client support in general, see the *MQSeries Clients* manual.

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, the following conditions give rise to a *Queue Full* event:

- Queue Full events are enabled for a specified queue, and
- An application issues an MQPUT call to put a message on that queue, but 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
- An application attempting to open a queue and specifying a user ID that is not authorized

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.

Types of Event

MQSeries events are categorized as follows:

Queue manager events

These events 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 events 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 events 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.

Transactional Support

An application program can group a set of updates into a *unit of work*. These updates are usually logically related and must all be successful for data integrity to be preserved. If one update succeeded while another failed then data integrity would belost.

A unit of work *commits* when it completes successfully. At this point all updates made within that unit of work are made permanent or irreversible. If the unit of work fails then all updates are instead *backed out*. *Syncpoint coordination* is the process by which units of work are 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, 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 MQI 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 over queue manager restart.

Part 2. Planning for and Installing MQSeries for ptx

This part of the manual gices you advice on planning and preparing for the installation of MQSeries on ptx. It also gives you step-by-step guidance on the installation process for MQSeries on these platforms. It contains the following chapters:

- Chapter 2, "Planning to Install the MQSeries for ptx Server"
- Chapter 3, "Installing the MQSeries for ptx Server"

Chapter 2. Planning to Install the MQSeries for ptx Server

This chapter is a summary of the requirements to run MQSeries for ptx, the network protocols and the compilers supported, the delivery media, and the various components of the product.

Hardware Requirements

System

Required:

- Any system running ptx running 4.5.1 or later
- 128MB RAM

Recommended:

- A Pentium Pro class processor
- 256MB+ RAM

Disk Space Required

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.

These are the approximate storage requirements:

- A minimum of 25 MB of disk space must be available for the product code and data in the filesystem containing the /opt directory.
- Online manuals in HTML format: 35 MB

Working data for MQSeries for ptx is stored by default in /var/mqm.

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.

- ptx V4.5.1 or later.

Required Supplements and Patches

- ptx 4.5.1:

  ptx/TCP/IP V4.6.1rv1

- ptx 4.5.2:

- ptx 4.6.1:

Connectivity

TCP/IP is the only network protocol supported.

- ptx/BaseComms V1.2.0 or later
- ptx/TCP/IP V4.6.1rv1
- ptx/LAN V4.7.1

Compilers Supported for MQSeries for ptx Applications

- ptx 4.5.1 or later C compiler
- C++ V5.2.3 compiler
- Micro Focus COBOL compiler V4.x for UNIX

Delivery

MQSeries for ptx is supplied on CD-ROM.

Installation

MQSeries for ptx takes approximately 5 minutes to install using the **pkgadd** program.

Note
MQSeries for ptx does not use ptx/ADMIN for installation.

MQSeries for ptx Components

When you install MQSeries for ptx you can choose which components to install. The components are as follows:

Runtime component

Support for external applications. This does **not** enable you to write your own applications.

Base

Support to enable you to create and support your own applications. Requires the runtime component to be installed.

Server

Support for client connections. Requires the runtime component to be installed.

Man

Man pages for the following commands:

- Control commands
- Message Queue Interface (MQI)
- MQSeries commands (MQSC)

Samples

Sample application programs.

MQSeries code

The MQSeries code in the following National Languages:

- US English
- French
- German
- Spanish

Note
The "base" product is automatically installed

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 MQSeries Version 2 (Sequent V1.0.0 or V1.1.0 versions), and you want to use the same settings with Version 5.0 of the product:

1. Save your copy of amqscoma.tst
2. Install MQSeries Version 5.0
3. Load your copy of amqscoma.tst and use the file to recreate your default objects

README file

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

Chapter 3. Installing MQSeries for ptx

This chapter tells you how to install MQSeries for ptx 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" below.

Preparing for installation

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

Before Installation

Before you can install MQSeries for ptx you:

- Must create a group with the name `mqm`.
- Must create a user ID with the name `mqm`.
- Must add `root` to the mqm group.
- Are recommended to create and mount a `/var/mqm` file system, or `/var/mqm/`, `/var/mqm/log`, and `/var/mqm/errors` file systems.

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.

If you are using a single file system, use the sum of these figures as a guide.

Notes

1. The `/var/mqm` file system should be large enough to contain all the messages, on all the queue managers, on this system.

2. If you create separate partitions, the following directories **must** be on a local file system:
   `/var/mqm`
   `/var/mqm/log`

3. You can choose to 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 using the default settings. For further information on log sizes see "Calculating the size of the log" in the MQSeries *System Administration* manual.

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.

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 th**e** `/opt/mqm` file system, for example, if that file system is too small to contain the product, you can:

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.

Installing MQSeries for ptx

This section describes the installation of the MQSeries for ptx server.

Notes

1. If you have previously installed MQSeries on your system, you need to remove the product using the `pkgrm` program.
2. If the product is present, but not installed correctly, you mat need to manually delete the files and directories contained in:
`/var/mqm`
`/opt/mqm`

Carry out the following procedure:

1. `mount` the CD-ROM as the `/cdrom` directory.
2. use the **`pkgadd`** program, and carry out the following procedure:
    1. Perform a `pkgadd -d /cdrom/ptx_500.img`.
    2. When you are prompted for which packages are to be installed, select the ones you require. If you want to install the entire MQSeries product, select `all`.
    3. Press the `Enter` key.

Note
To remove MQSeries use the **pkgrm** command and delete the /var/mqm directory tree.

Kernel configuration

MQSeries makes use of semaphores and shared memory. The default kernel configuration on ptx is not adequate. In particular, the **default** number of semaphores and shared memory segments defined in the kernel are **not** sufficient to support MQSeries.

The minimum recommended values of the kernel parameters are as follows:

```
SEMMAP 8192
SEMMSL 1024
SEMMNI 8192
SEMMNS 8192
```

```
SEMMNU 8192
SEMOPM 80
SEMUME 64
SHMMAX 268435456
SHMMNI 2048
SHMSEG 3500
```

Note
These values may need to be increased if you obtain any First Failure Support Technology™ (FFST) records.

After installation, you should review the machine's configuration and increase the values if necessary.

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 message persistence. 3. MQSeries queue managers are independent of each other. Therefore system kernel parameters, for example `shmmni` need to allow for the number of queue managers in the system.

For further information, see the chapter entitled "Tunable Parameters" in the *DYNIX/ptx System Configuration and Performance Guid*e.

Once you have made any changes to the kernel parameters, you must restart the system.

Translated messages

Messages in US English are always available. If you require another of the languages that is supported by MQSeries for ptx Version 5, you *must* ensure that your `NLSPATH` environment variable includes the appropriate directory.

For example:

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

Verifying your installation

This section describes how to verify that MQSeries for ptx 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 only supported protocol is TCP.

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 (standalone) 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". For information on verifying a client/server installation, see the *MQSeries Clients* manual.

Verifying a Local Installation

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 ptx on the server (include the Base Server component 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 the following and then press Enter:
  `strmqm`
- A message tells you when the queue manager has started.

4. Enable MQSC commands 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.

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 pressing Ctrl-D, or 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:

   `/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. If you are not already in the following directory, change to it now:

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

5. 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 servers, one as a sender and one as a receiver.

Sender Server:

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 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 server).

6. Define a sender channel:

   ```
   define channel (first.channel) chltype (sdr) conname (32.64.128.255) +
   xmitq (transmit1.queue) trptype (tcp)
   ```

   where *32.68.128.255* is the TCP address of the receiver server.

   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 pressing Ctrl-D, or typing `end`, and pressing Enter.

   Now set up the receiver server.

Receiver Server

Note
You must be logged in as a superuser, or as root, to perform step 1 to step 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* as the default queue manager on this server, you need to add `-m venus.queue.manager` to the end of this line.

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

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

4. Run the command:

   ```
   kill -1 inetd processed
   ```

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 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 pressing Ctrl-D or typing **end** and pressing Enter.

Establishing communication between the Servers:

1. If the queue managers on the two servers have been stopped for any reason, restart them now (using the `strmqm` command).

2. On the **Sender** server start the sender channel:

   ```
   runmqchl -c FIRST.CHANNEL -m saturn.queue.manager
   ```
   The receiver channel on the receiver server is started automatically when the sender channel starts.

3. On the **Sender** server, use the `amqsput` sample program to send a message to the queue on the receiver server:

```
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** server, use the `amqsget` sample program to get the message from the queue:

   ```
   amqsget ORANGE.QUEUE
   ```

   The message is displayed.

The verification is complete.

Setting the Queue Manager CCSID on MQSeries for ptx

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.

Note
The queue manager CCSID cannot be changed once the queue manager has been created.

To change the CCSID, you must:

- Delete the queue manager
- Change the locale
- Recreate the queue manager.

Examples of setting the CCSID:

`export LANG=en_US.ISO8859-1`

uses the code set ISO8859-1

and will set a CCSID of 819

`export LANG=en_US.ISO8859-2`

uses the code set ISO8859-2

and will set a CCSID of 912

`export LANG=C` **(this is the default locale)**

uses the code set ISO8859-1

and will set a CCSID of 819

You can display the queue manager CCSID by using the `dis qmgr` command from within `runmqsc`.

See Appendix D, "Support for Different Code Sets on MQSeries for ptx" for further information about supported code sets.

Part 3. Using MQSeries for ptx

This part of the manual shows you how to use MQSeries for ptx. It contains the following chapters:

- Chapter 4, "Using the MQSeries Command Sets"
- Chapter 5, "Obtaining Additional Information"

Chapter 4 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 (see "Introducing 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, whilst 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 Administratio*n manual.

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 UNIXÑ 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* manual.

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 Command Reference*.

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, the *MQSeries System Administration* manual.

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* manual.

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* manual.

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.

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

Note
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. After a preemptive shutdown, you must manually clean up the ipc shared memory segements and semaphores before restarting the queue manager.

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.

Restarting a Queue Manager

To restart a queue manager called `saturn.queue.manager`, use the command:

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

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* manual. For a complete description of the MQSC commands, see the *MQSeries Command Reference*.

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".

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 Command Reference* manual 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".

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:

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", we defined the queue RANGE.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.

Part 4. Appendixes

Appendix A. Installing the MQSeries Product License

License Management

MQSeries for ptx is nodelocked to a specific machine using LicensePower/iFOR licensing technology. The product ships with a temporary license key that enables the product to be used on any machine for approximately 60 days after you receive it.

If you have purchased the product, you must contact Willow Technology for a permanent license (see below for the procedure). Updating the temporary license to a permanent license only requires the replacement of, or a modification to, the license file itself - you do not have to reinstall or replace anything else.

License Key Installation

MQSeries for ptx is nodelocked to a specific machine using LicensePower/IFOR licensing technology. The software ships with a temporary license key that enables the software to be used for approximately 60 days after you receive it.

If your license expires, you will no longer be able to run MQSeries for ptx. Contact Willow Technology for a new license. As soon as a valid license is installed, MQSeries will again be fully operational.

Before you can run MQSeries for ptx (issue the `crtmqm` or `strmqm` commands, you must create a file containing the license information. Willow ships a temporary license with the software. To install the temporary license, follow these steps:

1. As root/superuser, copy the license certificate, amqtcert.lic, to /opt/mqm/lib.

2. Launch /opt/mqm/bin/setmqtry.

3. Read the license agreement, pressing the enter key to page though the license. At the end of the license, you must enter 'yes' to accept the terms of the agreement and install the license.

4. Once the license is installed, you should receive the message "There are <nn> days left in the trial period for this copy of MQSeries" where <nn> is the number of days left (maximum 60).

5. You may now proceed to use MQSeries for ptx. Each time you start a Queue Manager, a message indicating the number of days before the temporary license expires will be output. If the license expires, you will receive the following message when you attempt to create or start a Queue Manager: "AMQ7120: The Trial Period license for this copy of MQSeries has expired".

Obtaining a Permanent License

You may request a permanent license once you have purchased the product. You must provide us with the following information for us to issue a permanent license:

1. Your Proof of Entitlement number received when you purchased the product;

2. The output from the /opt/mqm/bin/i4target command from the system where you are planning on running this copy of MQSeries for ptx. This contains the hostid for your machine.

This information should be sent via email to support@willowtech.com, or by fax to +1.408.296.7700.

We will then email or fax you the permanent license certificate. If you receive it electronically, the file will be named "amqpcert.lic". If you receive it by fax, you must copy the contents EXACTLY to a file name "amqpcert.lic".

Once you have received your permanent/production license certificate, follow these steps to enable it:

1. As root/superuser, copy the license certificate, amqpcert.lic, to /opt/mqm/lib.

2. Launch /opt/mqm/bin/setmqprd.

3. You should receive the message: "This copy of MQSeries is now running in Production mode."

4. You may now proceed to use MQSeries for ptx.

Updating a license only requires the replacement of or a modification to the license file itself - you do not have to reinstall or replace anything else.

Appendix B. Migrating from an Earlier Version of MQSeries

This appendix tells you how to install the current version of the product over an earlier version.

Stop MQSeries…

You must halt all MQSeries activity on the target machine before installing the new release.

Carry out the following procedure:

1. Install the new version of the product as described in "Installing MQSeries for ptx".
2. Issue a `crtmqm` or `strmqm` command.

Appendix C. Sample MQI programs and MQSC files

MQSeries for ptx 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 C-1 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 C-1. 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 C-2 lists the sample MQI source files. By default, the source files are in directory /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 C-2. 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	Echoes a message from a message queue to the reply-to queue. Can be run as a triggered application program.
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	amq0get0.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	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.
amqsvfcx.c	--	A sample C skeleton of a Data Conversion exit routine.
amqsptl0.c	--	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.		

Miscellaneous Tools

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

Table C-3. 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 D. Support for different codesets on MQSeries for ptx

MQSeries for ptx supports many of the codesets used by the locales - that is, the subsets of the user's environment which define the conventions for a specific culture - that are provided as standard on MQSeries for ptx.

Note that to install most of these locales on an MQSeries for ptx system the Language Supplement (LS) for the operating system must be installed. 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 codeset.

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

Table D-1 shows the locales and the CCSIDs that are registered for the codeset used by the locale.

Table D-1. Locales and CCSIDs

Locale	Language	Codeset	CCSID
C	English	ISO 8859-1	819
Ar	Arabic	ISO 8859-6	1089
Ar_AA	Arabic	ISO 8859-6	1089
bu	Bulgarian	ISO 8859-5	915
bu_BG	Bulgarian	ISO 8859-5	915
cs	Czech	ISO 8859-2	912
cs_CZ	Czech	ISO 8859-2	912
Da	Danish	ISO 8859-1	819
Da_DK	Danish	ISO 8859-1	819
De	German	ISO 8859-1	819
De_DE	German	ISO 8859-1	819

De_AT	German - Austria	ISO 8859-1	819
De_CH	German - Switzerland	ISO 8859-1	819
el	Greek	ISO 8859-7	813
el_GR	Greek	ISO 8859-7	813
en	English - United Kingdom	ISO 8859-1	819
en_GB	English - United Kingdom	ISO 8859-1	819
en_UK	English - United Kingdom	ISO 8859-1	819
en_AU	English - Australia	ISO 8859-1	819
en_CA	English - Canada	ISO 8859-1	819
en_US	English - USA	ISO 8859-1	819
es	Spanish	ISO 8859-1	819
es_ES	Spanish	ISO 8859-1	819
fi	Finnish	ISO 8859-1	819
fi_FI	Finnish	ISO 8859-1	819
fr	French - France	ISO 8859-1	819
fr_FR	French - France	ISO 8859-1	819
fr_BE	French - Belgium	ISO 8859-1	819
fr_CA	French - Canada	ISO 8859-1	819
fr_CH	French - Switzerland	ISO 8859-1	819
hr	Croatian	ISO 8859-2	912
hr_HR	Croatian	ISO 8859-2	912
hu	Hungarian	ISO 8859-2	912
hu_HR	Hungarian	ISO 8859-2	912
is	Icelandic	ISO 8859-1	819
it	Italian - Italy	ISO 8859-1	819

it_IT	Italian - Italy	ISO 8859-1	819
it_CH	Italian - Switzerland	ISO 8859-1	819
iw	Hebrew	ISO 8859-8	916
iw_IL	Hebrew	ISO 8859-8	916
ja	Japanese	eucJP	5050
ja_JP	Japanese	eucJP	5050
mk	Macedonian	ISO 8859-5	915
mk_MK	Macedonian	ISO 8859-5	915
nl	Dutch - Netherlands	ISO 8859-1	819
nl_NL	Dutch - Netherlands	ISO 8859-1	819
nl_BE	Dutch - Belgium	ISO 8859-1	819
no	Norwegian	ISO 8859-1	819
no_NO	Norwegian	ISO 8859-1	819
pl	Polish	ISO 8859-2	912
pl_PL	Polish	ISO 8859-2	912
POSIX	English	ISO 8859-1	819
pt	Portuguese	ISO 8859-1	819
pt_PT	Portuguese	ISO 8859-1	819
ro	Romanian	ISO 8859-2	912
ro_RO	Romanian	ISO 8859-2	912
ru	Russian	ISO 8859-5	915
ru_RU	Russian	ISO 8859-5	915
ru_SU	Russian	ISO 8859-5	915
sh	Serbocroatian	ISO 8859-2	912
sh_SP	Serbocroatian	ISO 8859-2	912

sh_YU	Serbocroatian	ISO 8859-2	912
sl	Slovene	ISO 8859-2	912
sl_SL	Slovene	ISO 8859-2	912
sk	Slovak	ISO 8859-2	912
sk_SK	Slovak	ISO 8859-2	912
sr	Serbian Cyrillic	ISO 8859-5	915
sr_SP	Serbian Cyrillic	ISO 8859-5	915
tr	Turkish	ISO 8859-9	920
tr_TR	Turkish ISO	8859-9	920
sv	Swedish	ISO 8859-1	819
sv_SE	Swedish	ISO 8859-1	819
tr	Turkish	ISO 8859-9	920
tr_TR	Turkish ISO	8859-9	920

For further information listing inter-platform support for these locales, see the *MQSeries Distributed Queuing Guide*.

Appendix E. Notices

References in this publication to Willow Technology products, programs, or services do not imply that Willow Technology intends to make these available in all countries in which Willow Technology operates.

Any reference to a Willow Technology product, program, or service is not intended to state or imply that only that Willow Technology product, program, or service may be used.

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

WILLOW TECHNOLOGY, INC PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

Any functionally equivalent product, program, or service that does not infringe any of the intellectual property rights of Willow Technology and it's Licensors may be used instead of the Willow Technology product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by Willow Technology and it's Licensors, are the responsibility of the user.

Willow Technology and it's Licensors may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents.

Trademarks

Willow Technology, the Willow logo and willowtech.com are trademarks of Willow Technology, Inc.

The following terms are trademarks of the IBM Corporation in the United States, or other countries, or both:

AS/400	IBM	AIX
CICS	DB2	FFST
IBM	MQ	MQSeries
MVS	MVS/ESA	OS/400
ptx	DYNIX/ptx	Symmetry

NUMA-Q

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

LicensePower/iFOR is a registered trademark of Isogon Corporation.

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