MQSeries® for Compaq NonStop™ Kernel

# System Administration

*Version 5 Release 1*

SC34-5886-00

MQSeries® for Compaq NonStop™ Kernel

# System Administration

*Version 5 Release 1*

SC34-5886-00

**Note!**

Before using this information and the product it supports, be sure to read the general information under "Appendix Q. Notices" on page 383.

**First Edition (June 2001)**

This edition applies to MQSeries for Compaq NSK, Version 5.1 and to all subsequent releases and modifications until otherwise indicated in new editions.

**© Copyright International Business Machines Corporation 1993, 2001. All rights reserved.**
US Government Users Restricted Rights – Use, duplication or disclosure restricted by GSA ADP Schedule Contract with IBM Corp.

# Contents

© Copyright IBM Corp. 1993, 2001

## Chapter 5. Administering local MQSeries objects . . . . . . . . . . 83

## Chapter 6. Automating administration tasks . . . . . . . . . . . . . . . 107

## Chapter 7. Administering remote MQSeries objects. . . . . . . . . . 111

# Figures

© Copyright IBM Corp. 1993, 2001

# Tables

© Copyright IBM Corp. 1993, 2001

# About this book

MQSeries for Compaq NonStop Kernel, Version 5 Release 1—referred to in this book as MQSeries for Compaq NSK or MQSeries, as the context permits—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 book describes the system administration aspects of MQSeries for Compaq NSK, Version 5 Release 1, and the services it provides to support commercial messaging in a Compaq NSK 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 book is for

Primarily, this book 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 book

To use this book, you should have a good understanding of the Compaq NSK operating system and associated utilities. 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.

## How to use this book

The body of this book:

- Introduces MQSeries
- Describes day-to-day management of an MQSeries for Compaq NonStop Kernel system, addressing topics such as administration of local and remote MQSeries objects, security, transactional support, and problem determination

## Information about MQSeries on the Internet

**MQSeries URL**

The URL of the MQSeries product family home page is:

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

© Copyright IBM Corp. 1993, 2001

## MQSeries on the Internet

# What's new in MQSeries for Compaq NSK V5.1

The following new function is described in this edition of the *MQSeries for Compaq NSK V5.1 System Administration Guide*.

## Performance enhancements

**New queue server process**
A new queue server process has been introduced (into the queue manager) which provides message storage for one or more local queues and manages all GET and PUT operations on those queues. It provides an efficient implementation for non-persistent messaging and supports the new messaging functions for Version 5.1. See "Chapter 2. MQSeries for Compaq NSK V5.1 architecture" on page 19 for more information.

**Changes in status server operation**
The status server replaces the file-based approach to channel status. It supports the status information of those objects that are not local queues and provides efficient access to channel status information. See "Chapter 2. MQSeries for Compaq NSK V5.1 architecture" on page 19 for more information.

**Non-persistent messages**
You can now take advantage of the performance improvements offered by non-persistent messages.

**FASTPATH binding support for trusted applications**
If your application is suitable, you can connect to a queue manager using FASTPATH bindings to enjoy significant performance improvements. FASTPATH applications are restricted in certain ways and must be well behaved since this form of binding provides less protection for the critical internal data of the queue manager. See "Appendix I. Building and running applications" on page 323 for more information.

**Improvements to disk storage for persistent messages**
As part of the new queue server architecture, the storage of persistent messages on disk has been modified to provide enhanced performance for all sizes of message. No alternate key files are required for queue files and a new type of disk storage for very large messages has been introduced that maximizes the efficiency of storage for messages up to 100 MB in size.

## Upgraded MQSeries functionality

**MQSeries queue manager clusters**
MQSeries queue managers can be connected to form a cluster of queue managers. Within a cluster, queue managers can make the queues they host available to every other queue manager. Any queue manager can send a message to any other queue manager in the same cluster without the need for explicit channel definitions, remote queue definitions, or transmission queues for each destination. The main benefits of MQSeries clusters are:

- Fewer system administration tasks
- Increased availability
- Workload balancing

© Copyright IBM Corp. 1993, 2001

See the *MQSeries Queue Manager Clusters* book for a complete description of this function.

**MQSeries Administration Interface (MQAI)**
MQSeries for Compaq NSK V5.1 now supports the MQSeries Administration Interface (MQAI), a programming interface that simplifies the use of PCF messages to configure MQSeries. For more information about the MQAI, including full command descriptions, see *MQSeries® Administration Interface Programming Guide and Reference*.

**Enhanced MQI support**
MQSeries for Compaq NSK V5.1 now supports advanced message functionality provided in Version 5.0 and Version 5.1 MQSeries releases on other platforms. This includes distribution list processing, reference messages, segmented messages and many other new options. See the *MQSeries Application Programming Guide* and the *MQSeries Application Programming Reference* for more information.

**Increased size of messages and message queues**
A message can be up to 100 MB in size. A message queue can be up to 4 GB.

**Automatic default object creation**
When you use the **crtmqm** command to create a queue manager, the system default objects are automatically created.

**Controlled, synchronous shutdown of a queue manager**
A new option has been added to the **endmqm** command to allow controlled, synchronous shutdown of a queue manager.

**Java™ support**
MQSeries for Compaq NSK V5.1 now works with Java compilers, allowing applications to be coded in Java. See "Appendix F. MQSeries and Compaq NonStop Server for Java" on page 309 for more information.

**OSS application support**
MQSeries for Compaq NSK V5.1 now works with NSK OSS applications using C, C++, Cobol and Java.

**Web administration**
With MQSeries for Compaq NSK V5.1, you can perform the following tasks using a Microsoft® Windows NT® system in association with an HTML browser, for example, Netscape Navigator or Microsoft Internet Explorer:

- Log on as an MQSeries Administrator
- Select a queue manager and issue MQSC commands against it
- Create, edit and delete MQSC scripts.

**Improved user exit mechanism**
The mechanism for binding and configuring user exit code for use with MQSeries has been considerably improved to provide an interface closer to the standard, and a common mechanism for all exits. See "Appendix L. User exits" on page 341 for details.

## Intercommunications

**TCP/IP**
MQSeries for Compaq NSK V5.1 now permits multiple Guardian TCP/IP server processes to be used by one queue manager. This means better configurations for load balancing across network hardware, and

redundancy in network connections for a queue manager and applications. See “Appendix M. Setting up communications” on page 351 for more information.

**SNA**

MQSeries for Compaq NSK V5.1 has an improved mechanism for managing and controlling remote initiation of channels for the SNA transport protocol. This new mechanism uses a listener process that runs under PATHWAY and is supported for both SNAX and InSession ICE products. The non-standard channel attribute AUTOSTART is no longer supported. See “Appendix M. Setting up communications” on page 351 for more information.

**Channels**

Channels now support *heartbeats* and the ability to transmit non-persistent messages outside of a unit of work to provide better performance.

MQSeries for Compaq NSK V5.1 now supports the optional automatic definition of channels for remotely initiated channels from other queue managers or clients.

## Compaq NSK-specific ease-of-use

**Compaq NSK Fix Command included with runmqsc**

**runmqsc** now includes the Compaq NSK Fix Command facility to allow you to recall and edit MQSC commands. For more information, see “Using Compaq NSK Fix Command” on page 28.

**Enhanced altmqfls utility**

The utility **altmqfls** has been changed substantially to provide detailed administrative management of the storage options for messages. See “Chapter 15. Scalability and performance” on page 201 and “dspmqfls (Display MQSeries file attributes)” on page 253 for more details.

**MQMC panels**

The MQMC administration panels provided as part of the queue manager PATHWAY environment have been upgraded to support the enhanced MQSeries functionality in this release.

# Part 1. Guidance

© Copyright IBM Corp. 1993, 2001

## Chapter 8. Protecting MQSeries objects . . . 125

## Chapter 9. MQSeries dead-letter queue handler 145

## Chapter 10. Instrumentation and EMS events 157

## Chapter 11. Understanding transactional support and messaging . . . . . . . . . 165

## Chapter 12. Recovery and restart . . . . . . 169

## Chapter 13. Configuration files . . . . . . . 173

# Chapter 1. Introduction

This chapter introduces MQSeries for Compaq NonStop Kernel Version 5.1 (MQSeries for Compaq NSK V5.1) from an administrator's perspective. It describes the basic concepts of MQSeries and messaging. It contains these sections:

## MQSeries and message queuing

MQSeries allows application programs to use message queuing to participate in message-driven processing. Application programs can communicate across different platforms by using the appropriate message queuing software products. For example, Compaq NSK and MVS/ESA™ applications can communicate through MQSeries for Compaq NSK and MQSeries for OS/390® respectively. The applications are shielded from the mechanics of the underlying communications.

MQSeries products implement a common application programming interface known as the message queue interface (MQI) whatever platform the applications are run on. This makes it easier for you to port applications from one platform to another.

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

### Time-independent applications

With message queuing, the exchange of messages between the sending and receiving programs is independent of time. 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 has been started.

### Message-driven processing

Upon arrival 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.

© Copyright IBM Corp. 1993, 2001

## What is a message?

A *message* is a string of bytes that is meaningful 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:

- *application data*

  The content and structure of the application data is defined by the application programs that use them.
- *message descriptor*

  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.

  The format of the message descriptor is defined by MQSeries. For a complete description of the message descriptor, see the *MQSeries Application Programming Reference* guide.

The format of the message descriptor is defined by MQSeries. For a complete description of the message descriptor, see the *MQSeries Application Programming Reference*.

There are two types of messages: persistent messages and non-persistent messages. They differ in the following ways:

- Persistent messages survive the restarting of a queue manager. Non-persistent messages do not survive the restarting of a queue manager.
- Non-persistent messages are not normally written to disk and therefore are faster and use less resources to be added and removed from a queue than persistent messages.
- Under some failure conditions, non-persistent messages are not as reliable as persistent messages.
- Persistent messages cannot be put on a temporary dynamic queue.

### Message lengths

In MQSeries, the maximum message length is 100 MB (where, 1 MB equals 1 048 576 bytes). The message length can be limited by:

- The maximum message length defined for the receiving queue.
- The maximum message length defined for the queue manager.
- The maximum message length defined by either the sending or receiving application.
- The amount of storage available for the message.

It might take several messages to send all the information that an application requires.

Increasing the maximum message length could have some negative implications. Also, it could result in the message being too large for the queue or queue manager. In these cases, a message can be split into segments and then regrouped into a logical message. Logical grouping of messages allows applications to group messages that are similar and to ensure the sequence of the messages. For more on message segmentation and grouping, see the *MQSeries Application Programming Guide*.

## What is a queue?

A *queue* is a data structure used to store messages. The messages may be put on the queue by application programs or by a queue manager as part of its normal operation.

Each queue is owned by a *queue manager*. The queue manager is responsible for maintaining the queues it owns and for storing all the messages it receives onto the appropriate queues.

The maximum size of a queue is 4 GB. For information about planning the amount of storage you require for queues, see the *MQSeries Planning Guide* or visit the following web site for platform-specific performance reports:

http://www.ibm.com/software/mqseries/txppacs/txpm1.html

### How do applications send and receive messages?

Applications send and receive messages using *MQI calls*. For example, to put a message onto a queue, an application:

1. Opens the required queue by issuing an MQI MQOPEN call.
2. Issues an MQI MQPUT call to put the message onto the queue
3. Another application can retrieve the message from the same queue by issuing an MQI MQGET call.

For more information about MQI calls, see the *MQSeries Application Programming Reference*.

### Predefined and dynamic queues

Queues can be characterized by the way they are created:

- *Predefined queues* are created by an administrator using the appropriate command set. For example, the MQSC command DEFINE QLOCAL creates a predefined local queue. Predefined queues are permanent; they exist independently of the applications that use them and survive MQSeries restarts.
- *Dynamic queues* are created when an application issues an OPEN request specifying the name of a *model queue*. The queue created is based on a template queue definition, which is the model queue. You can create a model queue using the MQSC command DEFINE QMODEL. The attributes of a model queue, for example the maximum number of messages that can be stored on it, are inherited by any dynamic queue that is created from it.

  Model queues have an attribute that specifies whether the dynamic queue is to be permanent or temporary. Permanent queues survive application and queue manager restarts; temporary queues are lost on a restart.

### Retrieving messages from queues

In MQSeries, suitably authorized applications can retrieve messages from a queue according to these retrieval algorithms:

- First-in-first-out (FIFO).
- Message priority, as defined in the message descriptor. Messages that have the same priority are retrieved on a FIFO basis.
- A program request for a specific message.

The MQGET request from the application determines the method used.

# Objects

Many of the tasks described in this book involve manipulating MQSeries *objects*. In MQSeries Version 5.1, the object types include queue managers, queues, process definitions, channels, clusters, and namelists.

The manipulation or *administration* of objects includes:

- Starting and stopping queue managers
- Creating objects, particularly queues, for applications.
- Working with channels to create communication paths to queue managers on other (remote) systems. This is described in detail in the *MQSeries Intercommunication* book.
- Creating *clusters* of queue managers to simplify the overall administration process, or to achieve workload balancing.

This book contains detailed information about administration in the following chapters:

- "Chapter 3. Using the MQSeries command sets" on page 27
- "Chapter 4. Managing queue managers" on page 41
- "Chapter 5. Administering local MQSeries objects" on page 83
- "Chapter 7. Administering remote MQSeries objects" on page 111

## Object names

Each instance of a queue manager has an object name. This object name must be unique within the network of queue managers for proper identification of the target queue manager to which a message is sent.

The object name must be unique within a queue manager and object type. For example, you can have a queue and a process with the same name; however, you cannot have two queues with the same name.

An object name can have a maximum of 48 characters, with the exception of *channels*. Channel objects can have a maximum of 20 characters. For more information about names see "Using names" on page 227.

## Managing objects

MQSeries provides facilities for creating, altering, displaying, and deleting objects. These include:

- MQSC commands (MQSC), which can be entered from the keyboard or read from a file
- MQM (screen-based interface)
- Programmable Command Format (PCF) commands, which a program can use.
- Control commands, which you can enter interactively from the operating-system command line.

For more information, see "Chapter 3. Using the MQSeries command sets" on page 27.

### Object attributes

The properties of an object are defined by its object attributes. You can specify or change some object attributes, but only view others. For example, the maximum message length that a queue can accommodate is defined by its `MaxMsgLength` attribute. You can specify this object attribute when you create a queue. The *DefinitionType* attribute specifies how the queue was created. You can only display the *DefinitionType* attribute.

In MQSeries, there are two ways of referring to an object attribute:

- Using its PCF name, for example, *MaxMsgLength*. The PCF name is the formal name of an attribute.
- Using its MQSC name, for example, MAXMSGL.

The formal name of an attribute is its PCF name. Because using the MQSC facility is an important part of this book, you are more likely to see the MQSC name in examples than the PCF name of a given attribute.

## MQSeries queue managers

A queue manager provides message queuing services to applications. 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 you are provided with the appropriate reason code.

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.

### MQI calls

A queue manager object can be used for various MQI calls. For example, you can inquire about object attributes using the MQINQ MQI call.

**Note:** Messages are always put on queue objects, not on queue manager objects. You cannot put a message on a queue manager object.

## MQSeries queues

Queues are defined to MQSeries for Compaq NSK using:

- MQSC DEFINE commands
- Message Queue Management (MQM) facility of MQSeries for Compaq NSK
- PCF command Create Queue
- MQAI commands

These commands specify the type of queue and its object attributes. For example, a local queue has object attributes that specify when the applications reference that queue in MQI calls. Examples of object attributes are:

- Whether applications can retrieve messages from the queue (GET enabled)
- Whether applications can put messages on the queue (PUT enabled)
- Whether access to the queue is exclusive to one application or shared among applications
- The maximum number of messages that can be stored on the queue at the same time (maximum queue depth)
- The maximum length of messages that can be put on the queue

For further information:
- About MQSC, see the *MQSeries MQSC Command Reference*
- About MQM, see “Using the Message Queue Management (MQM) facility” on page 63
- About PCF commands, see the *MQSeries Programmable System Management* book

## Using queue objects

In MQSeries, there are various types of queue object. Each type of object can be manipulated by the product commands and is associated with real queues in different ways:

- **Local queue object**

  A local queue object identifies a local queue belonging to the queue manager to which the application is connected. All queues are local queues in the sense that each queue belongs to a queue manager and, for that queue manager, the queue is a local queue.
- **Remote queue object**

  A remote queue object identifies a queue belonging to another queue manager. This queue must be defined as a local queue to that queue manager. The information you specify when you define a remote queue object allows the local queue manager to find the remote queue manager, so that any messages destined for the remote queue go to the correct queue manager.

  Before applications can send messages to a queue on another queue manager, you must have defined a transmission queue and channels between the queue managers, unless you have grouped one or more queue managers together into a cluster. For more information about clusters, see “Remote administration using clusters” on page 112.
- **Alias queue object**

  An alias queue object allows applications to access a queue by referring to it indirectly in MQI calls. When an alias queue name is used in an MQI call, the name is resolved to the name of either a local or a remote queue at run time. This allows you to change the queues that applications use without changing the application in any way—you merely change the alias queue definition to reflect the name of the new queue to which the alias resolves.

  An alias queue is not a queue, but an object that you can use to access another queue.
- **Model queue object**

  A model queue object defines a set of queue attributes that are used as a template for creating a dynamic queue. Dynamic queues are created by the queue manager when an application issues an MQOPEN request specifying a queue name that is the name of a model queue. The dynamic queue that is created in this way is a local queue whose attributes are taken from the model queue definition. The dynamic queue name can be specified by the application or the queue manager can generate the name and return it to the application.

Dynamic queues defined in this way may be temporary queues, which do not survive product restarts, or permanent queues, which do.

## Local queues used by MQSeries

MQSeries uses various local queues for specific purposes related to its operation. You *must* define them before MQSeries can use them.

**Application queues:** A queue that is used by an application (through the MQI) is referred to as an application queue. This queue can be a local queue on the queue manager to which an application is connected, or it can be a remote queue that is owned by another queue manager.

Applications can put messages on local or remote queues. However, they can get messages from a local queue only.

**Initiation queues:** Initiation queues are queues that are used in triggering. A queue manager puts a trigger message on an initiation queue when a trigger event occurs. A trigger event is a logical combination of conditions that is detected by a queue manager. For example, a trigger event can be generated when the number of messages on a queue reaches a predefined depth. 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 “Managing objects for triggering” on page 103. For more information about triggering, see the *MQSeries Application Programming Guide*.

**Transmission queues:** A transmission queue temporarily stores messages that are destined for a remote queue manager. You must define at least one transmission queue for each remote queue manager to which the local queue manager is to send messages directly. These queues are also used in remote administration. See “Administering a remote queue manager” on page 113. For information about the use of transmission queues in distributed queuing, see the *MQSeries Intercommunication* book.

**Cluster transmission queues:** Each queue manager within a cluster has a cluster transmission queue called SYSTEM.CLUSTER.TRANSMIT.QUEUE. A definition of this queue is created by default on every queue manager.

A queue manager that is a part of the cluster can send messages on the cluster transmission queue to any other queue manager that is in the same cluster.

Cluster queue managers can communicate with queue managers that are not a part of the cluster. To do this, the queue manager must define channels and a transmission queue to the other queue manager in the same way as in a traditional distributed-queuing environment.

During name resolution, the cluster transmission queue takes precedence over the default transmission queue. When a queue manager that is not part of the cluster puts a message onto a remote queue, the default action, if there is no transmission queue with the same name as the destination queue manager, is to use the default transmission queue.

When a queue manager is part of a cluster, the default action is to use the SYSTEM.CLUSTER.TRANSMIT.QUEUE, except when the destination queue manager is not part of the cluster.

**Dead-letter queues:** A dead-letter queue stores messages that cannot be routed to their correct destinations. For example, this event occurs when the destination queue is full. The supplied dead-letter queue is called SYSTEM.DEAD.LETTER.QUEUE. These queues are also referred to as undelivered-message queues on other platforms.

For distributed queuing, you should define a dead-letter queue on each active queue manager.

**Command queues:** The command queue, named SYSTEM.ADMIN.COMMAND.QUEUE, is a local queue to which suitably authorized applications can send MQSeries for Compaq NSK commands for processing. These commands are then retrieved by an MQSeries component called the command server. The command server validates the commands, passes valid commands to the queue manager for processing, and returns any responses to the appropriate reply-to queue.

**Reply-to queues:** When an application sends a request message, the application that receives the message can send a reply message to the sending application. This message is put on a queue, called a reply-to queue, which is normally a local queue to the sending application. The name of the reply-to queue is specified by the sending application as part of the message descriptor.

**Event queues:** MQSeries for Compaq NSK supports instrumentation events, which can be used to monitor queue managers independently of MQI applications. Instrumentation events can be generated in several ways, for example:

- An application attempting to put a message on a queue that is not available or does not exist
- A queue becoming full
- A channel being started

When an instrumentation event occurs, the queue manager puts an event message on an event queue. This message can then be read by a monitoring application that can inform an administrator or initiate remedial action if the event indicates a problem.

**Note:** Trigger events are different from instrumentation events in that trigger events are not caused by the same conditions, and do not generate event messages.

For more information about instrumentation events, see the *MQSeries Programmable System Management* book.

## Process definitions

A *process definition object* defines an application that is to be started in response to a trigger event on an MQSeries for Compaq NSK queue manager. See “Initiation queues” on page 11 for more information.

The process definition attributes include the application ID, the application type, and data specific to the application.

Use the MQSC command DEFINE PROCESS or the PCF command Create Process to create a process definition.

## Channels

*Channels* are objects that provide a communication path from one queue manager to another. Channels are used in distributed message queuing to move messages from one queue manager to another. Channels shield applications from the underlying communications protocols. The queue managers can exist on the same or different platforms. For queue managers to communicate with one another, you must define one channel object at the queue manager that is to send messages, and another channel object at the queue manager that is to receive them.

MQSeries for Compaq NSK supports TCP/IP and SNA LU6.2 (SNAX or ICE) transport protocols.

For information on channels and how to use them, see the *MQSeries Intercommunication* book, and also “Preparing channels and transmission queues for remote administration” on page 114.

## Clusters

In a traditional MQSeries network using distributed queuing, every queue manager is independent. If one queue manager needs to send messages to another queue manager it must have defined a transmission queue, a channel to the remote queue manager, and a remote queue definition for every queue to which it wants to send messages.

A cluster is a group of queue managers set up in such a way that the queue managers can communicate directly with one another over a single network without the need for complex transmission queue, channels and queue definitions.

For information about clusters, see “Chapter 7. Administering remote MQSeries objects” on page 111 and the *MQSeries Queue Manager Clusters* book.

## Namelists

A namelist is an MQSeries object that contains a list of other MQSeries objects. Typically, namelists are used by applications such as trigger monitors, where they are used to identify a group of queues. The advantage of using a namelist is that it is maintained independently of applications; that is, it can be updated without stopping any of the applications that use it. Also, if one application fails, the namelist is not affected and other applications can continue using it.

Namelists are also used with queue manager clusters so that you can maintain a list of clusters referenced by more than one MQSeries object.

# System default objects

The *system default objects* are a set of object definitions that are created automatically for each queue manager, when the queue manager is created.

Default object names have the stem SYSTEM.DEFAULT; for example, the default local queue is SYSTEM.DEFAULT.LOCAL.QUEUE; the default receiver channel is SYSTEM.DEFAULT.RECEIVER. You cannot rename these objects; default objects of these names are required.

### System default objects

When you define an object, any attributes that you do not specify explicitly are copied from the appropriate default object. For example, if you define a local queue, the attributes you do not specify are taken from the default queue SYSTEM.DEFAULT.LOCAL.QUEUE.

After a queue manager is created, you can use the **runmqsc** command to replace any defaults with other definitions.

See “Appendix B. System defaults” on page 297 for more information about system defaults.

# Administration

In MQSeries, you execute administration tasks by issuing *commands*. Four command sets are provided. Which set you use depends on the tasks you want to perform and how you want to perform them. The command sets are described in “Chapter 3. Using the MQSeries command sets” on page 27. Administration tasks include:

- Starting and stopping queue managers.
- Creating objects, particularly queues, for applications.
- Working with channels to create communication paths to queue managers on other (remote) systems. This process is explained in detail in the *MQSeries Intercommunication* book.

## Local and remote administration

Local administration means carrying out administration tasks on any queue managers you have defined on your local system. You can access other systems, for example through the TCP/IP terminal emulation program **telnet**, and carry out administration there. In MQSeries, you can consider this as local administration because no channels are involved, that is, the communication is managed by the operating system.

MQSeries supports administration from a single point through what is known as *remote administration*. This allows you to issue commands from your local system that are processed on another system. You do not have to log on to that system, although you do need to have the appropriate channels defined. The queue manager and command server on the target system must be running. For example, you can issue a remote command to change a queue definition on a remote queue manager.

Some commands cannot be issued in this way, in particular, creating or starting queue managers and starting command servers. To perform this type of task, you must either log onto the remote system and issue the commands from there or create a process that can issue the commands for you.

# Clients and servers

MQSeries for Compaq NSK supports client-server configurations for MQI applications. There are no MQSeries for Compaq NSK clients, only an MQSeries for Compaq NSK server; however, clients on other platforms can connect to the MQSeries for Compaq NSK server.

An *MQI client* is 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 system, that is, on the MQI server machine. A server can support normal, local MQI applications as well.

For more information, see the *MQSeries Intercommunication* book and the *MQSeries Clients* book.

## MQI applications in a client-server environment

When linked to a server, MQI client applications can issue MQI calls in the same way as local applications. The client application issues the MQCONN call to connect to a specified queue manager. Any additional MQI calls that specify the connection handle returned from the connect request are then processed by this queue manager.

You must link your applications to the appropriate client libraries. See the *MQSeries Application Programming Guide* for further information. No MQI client is currently provided for Compaq NSK; however, since Compaq NSK is an MQI Server, it accepts connections from any MQSeries MQI client running on other platforms.

# Extending queue manager facilities

The facilities provided by a queue manager can be extended by:
- User exits
- Installable services

## User exits

User exits let you insert your own programming code into a queue manager function. Two types of user exit are supported:

- *Channel exits*

  These exits change the way that channel operates. Channel exits are described in the *MQSeries Intercommunication* book.
- *Data conversion exits*

  These exits create source code fragments that can be put into application programs to convert data from one format to another. Data conversion exits are described in the *MQSeries Application Programming Guide*.
- *The cluster workload exit*

  This exit can be used to change the way that a queue manager in a cluster chooses between multiple instances of a remote queue. Call definition information is given in the *MQSeries Queue Manager Clusters* book.

**Note:** The mechanism for enabling user exits in MQSeries has changed in Version 5.1. Carefully review the description of the exit mechanism to determine the changes that you need to make to migrate user exit code from previous versions of MQSeries for Compaq NSK.

For more information about these exits, see “Appendix L. User exits” on page 341.

## Installable services

Installable services are more extensive than user exits in that they have a formalized Application Programming Interface (API) with multiple entry points.

An implementation of an installable service is called a *service component*. You can use the components supplied with the product, or you can write your own component to perform the functions that you require. Currently, the following installable services are provided:

- **Authorization service**

  The authorization service lets you build your own security facility.

  The default service component that implements the service is the Object Authority Manager (OAM), which is supplied with the product. By default, the OAM is enabled. You can use the authorization service interface to create other components to replace or augment the OAM. For more information about the OAM, see “Chapter 8. Protecting MQSeries objects” on page 125.
- **Name service**

  The name service allows queue managers to share queues by allowing applications to identify remote queues as though they were local queues.

  You can write your own name service component. See the *MQSeries Programmable System Management* book for more information.

# Security

MQSeries for Compaq NSK provides security through the Object Authority Manager (OAM) facility.

## Object Authority Manager (OAM) facility

Authorization for using MQI calls and commands and for accessing objects is provided by the Object Authority Manager (OAM), which by default is enabled. Access to MQSeries entities is controlled through MQSeries for Compaq NSK principals and user groups, and the OAM. The principal and group names that the OAM supports are resolved to Compaq NSK user and group names. In Version 5.1, all users of MQSeries must have a principal name that maps to a Compaq NSK user name. This is required regardless of whether the OAM is enabled. A command-line interface is provided to allow you to add and delete principals, and to grant and revoke authorizations, as required.

In addition, Compaq NSK security facilities can be used to control access to MQSeries commands and database files. If SAFEGUARD is installed, MQSeries is compatible with, and can take advantage of, some of the extended facilities that it provides. For more information, see “Chapter 8. Protecting MQSeries objects” on page 125.

# 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 be lost.

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 belonging to 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.

In MQSeries for Compaq NSK, all units of work are coordinated via TM/MP, or a compatible layered product that itself uses TM/MP (for example, NonStop Tuxedo in the OSS environment.) Applications can use TM/MP to coordinate units of work that include updates to any resource that is compatible with TM/MP as well as MQSeries messages. There are no XA-compliant databases in the Compaq NSK environment, but a similar interface is implemented. For example, updates to NonStop SQL databases or ENSCRIBE files may be coordinated with MQSeries messaging operations to maintain database integrity.

For more information, see “Chapter 11. Understanding transactional support and messaging” on page 165.

## Performance tuning, reliability, scalability and sizing

Facilities are provided to allow MQSeries for Compaq NSK to take advantage of the features of the Compaq NSK hardware and software that provide for scalable, high performance and reliable application infrastructure. These features include:

- Flexibility to configure queue managers and their objects across CPUs and disks to increase parallelism
- Software features to ensure that any single point of failure may be tolerated with at most a momentary interruption of service and automatic recovery to full service
- Database integrity protected by software and the underlying transactional file system
- Compatibility with products that provide disaster recovery protection.

For more information, see “Chapter 15. Scalability and performance” on page 201.

# Chapter 2. MQSeries for Compaq NSK V5.1 architecture

This chapter describes the overall architecture of MQSeries for Compaq NSK V5.1. It contains these sections:
- "Queue manager process overview"
- "Product packaging" on page 21
- "Executables" on page 21
- "Comparing Version 5.1 with Version 2.2.0.1 architecture" on page 22
- "Queue manager — functional view" on page 22
- "Queue manager process model" on page 23
- "MQSeries files and subvolumes" on page 24

The information may help you set the optimal configuration of the product for your operating environment.

"Chapter 15. Scalability and performance" on page 201 and "Chapter 16. Data integrity and availability" on page 211 also provide specific configuration guidance in the context of this architectural information.

## Queue manager process overview

The figure below shows the processes that make up an MQSeries queue manager (PATHMON is shown for completeness, since the queue manager runs in PATHWAY). The figure also provides an overview of the process IPC interactions.

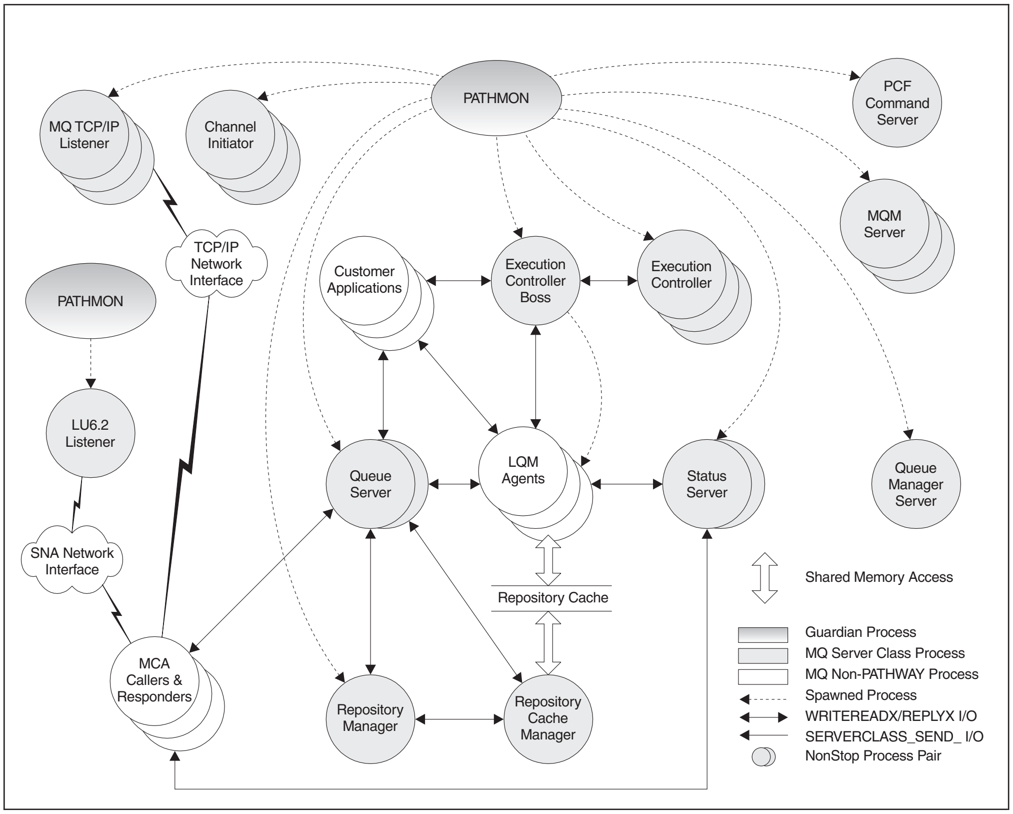


*Figure 1. MQSeries for Compaq NSK processes*

© Copyright IBM Corp. 1993, 2001

When a queue manager starts, PATHWAY creates the key queue manager processes. In the default configuration, PATHMON starts the Execution Controller Boss (ECBoss), the Execution Controllers (EC), the status and queue servers and the repository manager.

The ECBoss handles all incoming MQCONN requests and distributes them among the available Execution Controllers, providing a load balancing function by selecting the EC that is servicing the least connections.

The ECs manage and monitor the other queue manager processes and MQI applications.

The Local Queue Manager Agents (LQMA) perform the operations required to process MQI calls on behalf of applications. The Agents execute the bulk of the code that supports the MQI. The primary purpose of the agent is to separate application programs from the queue manager's critical resources, to protect against rogue or malicious applications.

The number of agent processes depends on the number of connected applications and the MinIdleLQMAgents specified in the QMINI file. When the queue manager starts, each Execution controller will start the specified number of idle agents. As agents become active, the Execution Controller starts new agents to maintain the minimum number of idle agents.

Central to the MQSeries architecture for Version 5.1 is the *queue server*. The queue server is a NonStop process pair that supports all messaging operations for local queues. When first created, a queue manager has a single default queue server. Depending on your system configuration and performance requirements, you can configure additional queue servers and assign local queues to them.

The status server handles status information for all objects other than local queues. This server subsumes the functionality of the MQSS server used in Version 2.2.0.1. Additionally, the default status server handles channel status information for the queue manager. When first created, a queue manager has a single default status server. Depending on your system configuration and performance requirements, you can configure additional status servers and assign MQSeries objects to them.

The queue manager starts the channel initiator server class automatically when it starts. This is to allow clustering operations to function without the requirement to manually start the channel initiator.

New to Version 5.1 are the *repository manager* and *repository cache server* processes for handling cluster queues. There is one repository manager process in each CPU where an Execution Controller is running. The first of the processes to start assumes the role of repository manager, which coordinates repository activities across the queue manager. Subsequent processes manage a shared memory segment containing the repository cache for the CPU in which they are executing. There can be only one repository manager per CPU. The repository cache contains information about clustered MQSeries objects, including queues and other queue managers in the cluster.

Also new to Version 5.1 is the *queue manager server*. This server handles expired messages and reporting.

The Message Channel Agents (MCA) transfer messages to and from other queue managers. In MQSeries for Compaq NSK V5.1, the MCAs are FASTPATH-bound for efficiency. For more on FASTPATH applications, see “FASTPATH binding application programs” on page 209.

The LU6.2 listener (which runs in a separate PATHWAY environment) is new for Version 5.1. “Appendix M. Setting up communications” on page 351 contains detailed information on the operation and configuration of the listener and its environment.

## Product packaging

MQSeries for Compaq NSK V5.1 provides three types of bindings: native dynamic, native static and non-native.

For native dynamic binding, MQSeries provides a native mode Shared Resource Library (MQSRLLIB). One of the design goals for MQSeries for Compaq NSK V5.1 was to incorporate as much of the product as possible into the Shared Resource Library (SRL), minimizing the product footprint. Using native dynamic binding is the preferred approach for using MQSeries, since it makes the most efficient use of system resources. From the OSS environment, this is the only binding supported.

For native static binding, MQSeries for Compaq NSK provides a re-linkable library (MQSRLLNK). This library is provided for customers who are already using an SRL and therefore cannot use the MQSeries SRL.

MQSeries provides a non-native static library for compatibility with legacy applications that cannot use the native mode bindings.

## Executables

The following table shows the MQSeries executables:

*Table 1. MQSeries executables*

| **MQSeries executable** | **Name** |
|---|---|
| Execution Controller | MQEC |
| Execution Controller Boss | MQECBOSS |
| Local Queue Manager Agent | MQLQMAG |
| Status Server | MQSTSVR |
| Queue Server | MQQSSVR |
| Repository Manager | MQREPSVR |
| Queue Manager Server | MQMGRSVR |
| Command Server | MQCMDSVR |
| MQSeries Management Server | MQMSVR |
| Channel Initiator | RUNMQCHI |
| Caller MCA | MQMCACAL |
| TCP/IP Responder MCA | MQTCPRES |
| LU6.2 Responder MCA | MQLU6RES |
| Trigger Monitor | RUNMQTRM |

## Comparing Version 5.1 with Version 2.2.0.1 architecture

The major structural change between MQSeries Version 2.2.0.1 and Version 5.1 is the introduction of queue servers. Each queue server maintains the data and files associated with one or more local queue objects. When first created, a queue manager has a single default queue server that is responsible for all local queues associated with the queue manager. You can add additional queue servers (using PATHWAY) and assign queues to them using **altmqfls**.

The queue servers are NonStop process pairs, which support all messaging operations for local queues. The queue server also supports purely memory-based non-persistent messages, providing significant performance advantages over Version 2.2.0.1. MQSeries uses an internal interface to TM/MP to integrate these memory-based messages into transactions, allowing both persistent and non-persistent messages to be included in the same transaction, without the overhead of writing the non-persistent messages to disk. The queue servers take over the function of the Version 2.2.0.1 MQSS servers for local queues only.

The MQSS servers have been superceded by status servers. The status servers are NonStop process pairs that maintain status information for objects other than local queues.

A second major change in the architecture is the introduction of shared memory segments. The queue manager Initialization file is distributed across the queue manager using read-only shared memory. This provides improved performance when connecting to a queue manager. The MQSeries Repository Cache (used to implement clustering features) uses a read/write shared memory segment in each CPU where MQSeries processes are running.

Version 5.1 incorporates more transparent support for LU6.2 channels. This version introduces an LU6.2 listener, which is responsible for starting LU6.2 responders. As a consequence of this, the ″AUTOSTART″ attribute for LU6.2 channels has been removed.

## Queue manager — functional view

The queue manager has the following major components:

**Application Interface**
Provides the environment and mechanism for execution of MQI calls.

**Queue Manager Kernel**
Provides most of the function of the MQI. For example, triggering is implemented here.

**Object Authority Manager (OAM)**
Provides access control for the queue manager and its resources. It allows specification of which users and groups are permitted to perform which operations against which resources.

**Data Abstraction and Persistence (DAP)**
Provides storage and recovery of the data held by the queue manager. DAP holds the messages.

**Message Channel Agents**
These are special applications that use the MQI for the majority of their operations. They are concerned with reliable transmission of messages between queue managers. MCAs are FASTPATH-bound.

**Command Server**
The command server is a special MQI application that is concerned with processing messages containing commands to manage the queue manager.

**Common Services**
This insulates the rest of the queue manager from the operating system. It provides a set of operating system-like services such as storage management, serialization and process management.

Figure 2 shows the relationship between the components.

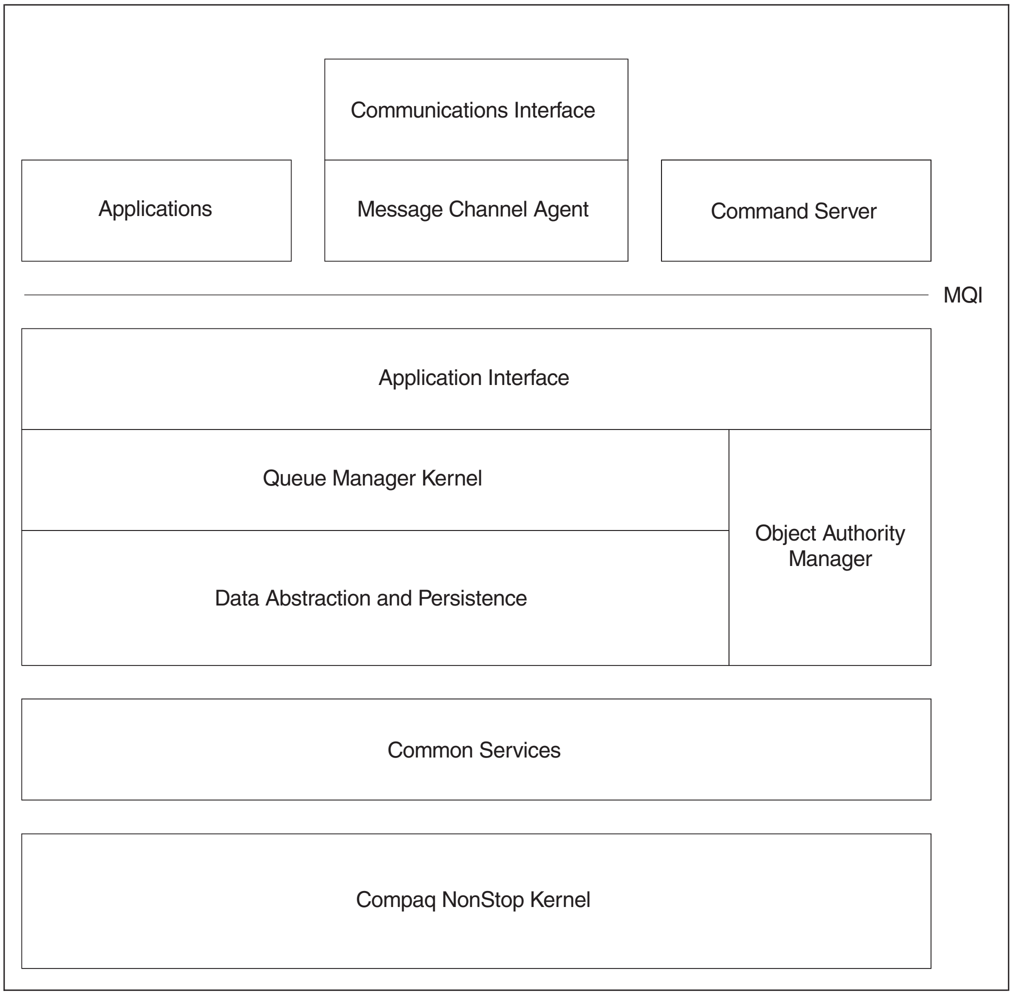


*Figure 2. Components of MQSeries for Compaq NSK V5.1*

## Queue manager process model

The application communicates with the Execution Controller Boss (ECBoss) when it needs to connect to an agent. The ECBoss selects the least used Execution Controller (EC), in terms of the smallest number of connected applications, and forwards the connection request to that EC. The EC selects an idle Local Queue Manager Agent (LQMA). The EC returns a reply to the application via the ECBoss, which then connects to the selected LQMA.

The application interface is split into two parts:

- The MQI application stub packages MQ requests and passes them to the agent process using the Inter-Process Communication Component (IPCC).

- The IPCC provides a message-passing interface between the MQI applications, the agents, the ECs and the ECBoss. Underlying the MQSeries IPCC component are standard GUARDIAN IPCs.

The application communicates with its agent process via the IPCC. The agent process performs the MQI calls on the applications behalf. The IPCC exchanges between the application and agent are synchronous request-reply messages. In addition to communicating with the agent, when performing MQPUT and MQGET operations, the application transfers message data directly to the queue server responsible for the queue.

For FASTPATH-bound applications, MQSeries code is linked directly with the user application. This provides performance benefits but carries the risk that a rogue application could disrupt the operation of the queue manager and cause data loss or other problems.

## MQSeries files and subvolumes

MQSeries uses a number of Compaq NonStop ENSCRIBE files. The location and names of files are summarized below.

The files associated with an MQSeries queue manager are distributed across a number of subvolumes.

**<qmgr> D**
Data subvolume. Used for files that hold queue manager-wide information.

**<qmgr> M**
Message data subvolume. Used for files that are associated with message data.

**<qmgr> L**
Error logs

**<qmgr> S**
Channel synchronization files.

**<qmgr>**
FFST™ files

**Object Catalog**
The object catalog (OBJCAT) contains information relating to MQSeries objects. The object catalog is located in the queue manager Data subvolume.

**Queue Files**
Each local queue has a queue file, a queue overflow file and a touch file associated with it. The files are prefixed with Q, O and T respectively—the remainder of the name is either part of the MQSeries name of the object, or a system generated name. To find out the files associated with a particular queue, use the **dspmqfls** utility. When a queue is created, the queue manager creates the associated files in the queue manager message subvolume (<qmgr>M). **altmqfls** provides a facility to relocate these files to another volume, if you need to do so for performance or space reasons.

The queue manager creates message overflow files for each message that is larger than the message overflow threshold configured for the queue. Message overflow files are located in the queue server subvolume by default. If you need to change the subvolume where the queue server creates message overflow files, use the **altmqfls** utility. A message overflow file is an unstructured, non-audited file that is dedicated to a particular message (See “Queue servers and queue files” on page 203 for more on queue file configuration). To ensure efficient data transfer, the data is written to the message overflow file using the large transfer mode (SETMODE 141), that enables transfer directly from the application process memory in 56 KB segments.

For persistent messages smaller than approximately 3 KB, the message data is stored in the queue file. For persistent messages between 3 KB and the configured message overflow threshold for the queue, the queue server writes additional records in the queue overflow file. For persistent messages above the message overflow threshold, the queue server creates a message overflow file to store the message.

If you need to partition queue or queue overflow files for size or performance reasons, “Partitioning queue files” on page 205 describes how to do this. If you partition the files, MQSeries distributes message data equally across the partitions to provide optimum performance.

“Chapter 15. Scalability and performance” on page 201 provides more information on the performance tuning options related to queues and queue files.

**Alias and Remote**

Alias and remote queues have a touch file associated with them. The touch file begins with the prefix T and is located in the queue manager data subvolume.

**Namelist Files**

Each namelist has an unstructured file associated with it. The files begin with the prefix L and are located in the queue manager data subvolume. The remainder of the file name is either part of the MQSeries object name, or a system generated value. To find out the name of the file associated with a particular namelist, use the **dspmqfls** utility.

# Chapter 3. Using the MQSeries command sets

This chapter describes the commands you can use for performing system administration tasks on MQSeries objects. Administration tasks include creating, starting, altering, viewing, stopping, and deleting queue managers, queues, processes, and channels. To perform these tasks, you must select the appropriate command.

MQSeries for Compaq NSK V5.1 provides the following administration command sets for performing administrative tasks:

- MQSC (MQSeries commands)
- PCF (Programmable Command Format) commands
- Control commands
- MQAI (MQSeries Administrator Interface)

In addition:

- Some TS/MP (PATHWAY) commands are used for administration purposes.
- The MQM (Message Queue Management) facility supports some administration tasks. The MQM is described in “Using the Message Queue Management (MQM) facility” on page 63.

This chapter introduces the MQSC, PCF, and control command sets, and provides a summary of the functions supported by each command set in “Appendix D. Comparing command sets” on page 303. How to use TS/MP commands is described in “TS/MP (PATHWAY) administration” on page 29.

## Performing administration using control commands

The following types of control commands are available:

- Commands for creating, starting, stopping, and deleting queue managers
- Commands for starting, stopping, and displaying command servers
- Utility commands associated with, for example, running MQSC commands, managing access to MQSeries objects, starting and stopping an MQSeries trace, and running trigger monitors

### Using control commands

You run control commands from the Compaq TACL prompt. Command names are not case sensitive. (Note, however, that queue manager names *are* case sensitive.) For example:

```
runmqsc
```

“Chapter 17. The MQSeries control commands” on page 227 explains the syntax and purpose of each command.

© Copyright IBM Corp. 1993, 2001

# Performing administration using MQSC commands

You can use the MQSC commands to manage queue manager objects including the queue manager, channels, queues, and process definitions. For example, you can define, alter, display, and delete a specified queue using MQSC commands.

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. These commands are summarized in "Appendix D. Comparing command sets" on page 303. For detailed information about each MQSC command, see the *MQSeries MQSC Command Reference*.

## Running MQSC commands

You can run MQSC interactively by invoking the control command **runmqsc** from the Compaq TACL prompt or running a script when a local queue manager is running. You can run the **runmqsc** command itself in three modes, depending on the flags set on the command:

- *Verification mode*, where the MQSC commands are verified on a local queue manager, but are not run.
- *Direct mode*, where the MQSC commands are run on a local queue manager.
- *Indirect mode*, where the MQSC commands are run on a remote queue manager.

For more information about using the MQSC facility and text files, see "Using the MQSC facility interactively" on page 85. For more information about the **runmqsc** command, see "runmqsc (Run MQSeries commands)" on page 273. MQSC commands are summarized in "Appendix D. Comparing command sets" on page 303.

### Using Compaq NSK Fix Command

If you run the **runmqsc** interactively (from the NSK TACL prompt), then you can also use the Compaq NSK Fix Command facility which allows you to recall and edit MQSC commands. For example:

- Typing `history` or `h` produces a list of the ten most recent commands
- Typing `!`*n* where *n* is the command number will re-execute that command
- Typing `h` *n* or `history` *n* where *n* is a number will list the n most recent commands
- Typing `fc` presents the last command entered for editing. Typing `fc` *n* where *n* is the command number presents that command for editing. Typing `fc` *string* where *string* is the beginning part or all of a previously entered command presents the last occurrence of that command for editing. The syntax is NSK standard. For example, type `d` to delete a character, `i` to insert a character and `r` to replace a character.

# Performing administration using PCF commands

PCF commands let you program administrative tasks into your applications or an administration program. PCF commands cover the same range of functions that are provided by the MQSC facility. You can write a program to issue PCF commands to any queue manager in the network from a single node. You can also centralize and automate administration tasks.

Each PCF command is a data structure that is embedded in the application data part of an MQSeries message. Each command is sent to the target queue manager using the MQI function MQPUT. The command server on the queue manager receiving the message interprets it as a command message and runs the command. To get the replies, the application issues an MQGET call and the reply data is returned as a data structure in the application data part of the MQSeries message. The application can then process the reply and act accordingly.

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

You must specify the following items to create a PCF command message:

**Message descriptor**

This is a standard MQSeries message descriptor, in which:
Message type (*MsgType*) is MQMT_REQUEST.
Message format (*Format*) is MQFMT_ADMIN.

**Application data**

Contains the PCF message including the PCF header, in which:
The PCF message type (*Type*) specifies MQCFT_COMMAND.
The command identifier specifies the command, for example, *ChangeQueue* (MQCMD_CHANGE_Q).

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

## Attributes in MQSC and PCFs

Object attributes specified in MQSC are in uppercase (for example, RQMNAME), although they are not case sensitive. These attribute names are limited to eight characters (for example, QDPHIEV). Object attributes in PCF are shown in italics, and are not limited to eight characters. The PCF equivalent of RQMNAME is *RemoteQMgrName* and of QDPHIEV is *QDepthHighEvent*.

## Escape PCFs

Escape PCFs are PCF commands that contain MQSC commands within the message text. You can use PCFs to send commands to a remote queue manager. For more information about using escape PCFs, see the *MQSeries Programmable System Management* book.

The MQAI is an administration interface to MQSeries that is now available on the Compaq NSK platform. It is an easy way to administer MQSeries; you do not have to write your own PCF messages and this avoids the problems associated with complex data structures. For more about using the MQAI, see “Using the MQAI to simplify the use of PCFs” on page 108.

# TS/MP (PATHWAY) administration

Most operations on the queue manager are accomplished by running MQSeries control commands from TACL. Some operations, however, require the use of PATHCOM to operate directly on the TS/MP server classes. Also, because of system-configuration changes, you might need to perform some administration actions on the TS/MP configuration itself.

This section summarizes these activities.

## Specifying and controlling TCP/IP listeners

To start TCP/IP listeners, you can use the MQSeries control command **runmqlsr** (described in “runmqlsr (Run listener)” on page 271), or you can use the PATHCOM commands THAW SERVER and START SERVER. To stop TCP/IP listeners, use the PATHCOM commands FREEZE SERVER and STOP SERVER. Use the PATHCOM command STATUS SERVER to display the number of TCP/IP listeners running, and their process names.

By default, each queue manager has one listener that is in server class MQS-TCPLIS00. Use the PATHCOM command ADD SERVER to create additional TCP/IP listener server classes to service more than one TCP/IP port. Each TCP/IP listener should be configured in its own TS/MP server class for maximum flexibility. If you add TCP/IP listeners, you must also add TCP/IP port definitions to the queue manager initialization file (QMINI), as described in “TCP/IP ports listened on by the queue manager” on page 49. The first listener to be started uses the first listener port defined in QMINI, the second listener uses the second listener port, and so on. For an example of the QMINI entries required to support multiple listeners, see “Configuring QMINI to support multiple TCP/IP listeners” on page 368.

## Controlling the command server

The command server is created as the TS/MP server class MQS-CMDSERV00. As an alternative to the control commands **strmqcsv**, **endmqcsv**, and **dspmqcsv**, you can use the PATHCOM commands THAW SERVER, START SERVER, FREEZE SERVER, STOP SERVER, and STATUS SERVER.

## Specifying and controlling channel initiators

The default channel initiator is created as the TS/MP server class MQS-CHANINIT00. As an alternative to using the **runmqchi** control command (described in “runmqchi (Run channel initiator)” on page 268), you can use the PATHCOM commands THAW SERVER, START SERVER, FREEZE SERVER, STOP SERVER, and STATUS SERVER to control and display the status of the channel initiator. The default channel initiator processes the default channel initiation queue, SYSTEM.CHANNEL.INITQ.

### Changing the default initiation queue for the channel initiator

In Version 5.1, the queue manager starts the default channel initiator automatically. The default channel initiator must be running to support cluster operations.

If you want to use an initiation queue other than the default (SYSTEM.CHANNEL.INITQ), you must change the PATHWAY configuration.

**Note:** You should not change the initiation queue if the queue manager is part of a cluster. Changing the default initiation queue for the default channel initiator disables support for clusters.

You can change the default initiation queue while the queue manager is running, but the channel initiator server class itself must be stopped. In PATHCOM, issue the following command against the queue manager’s PATHWAY configuration:

```
ALTER SERVER MQS-CHANINIT00, STARTUP "-q<init-queue>"
```

where <*init-queue*> is the name of the alternative initiation queue. You can then start the channel initiator and exit PATHCOM.

## Specifying and controlling trigger monitors

A single default trigger monitor is created as the TS/MP server class MQS-TRIGMON00. You can use the PATHCOM commands THAW SERVER, START SERVER, FREEZE SERVER, STOP SERVER, and STATUS SERVER to administer this server class. If you need additional trigger monitors, you can configure them as additional server classes, using MQS-TRIGMON00 as a template. You are recommended to use separate server class objects for maximum flexibility. You do not have to use TS/MP to control trigger monitors. For example, you can run the trigger monitor from TACL using the control command **runmqtrm**.

The default trigger monitor processes the default initiation queue, SYSTEM.DEFAULT.INITIATION.QUEUE. You can change this by adding or changing the STARTUP message for the server class that holds the trigger monitor. You need to do this if more than one trigger monitor is configured for the queue manager. Use the PATHCOM ALTER SERVER command to add or change the STARTUP attribute.

## Specifying the distribution of processes across CPUs

An important aspect to the distribution of work among CPUs is the CPU assigned to each EC in the queue manager. Each EC creates and manages a set of agent processes in its own CPU only. Consequently, if the EC processes are distributed among the CPUs of the system, the agent processes are similarly distributed.

By default, if multiple ECs are specified, the EC processes (each a separate server class) are distributed as evenly as possible among the available CPUs on the system. There is no built-in limit to the number of EC processes in a queue manager: the number required depends entirely on the load to be handled by the queue manager. By default, there is one EC process in the queue manager.

The default EC server class is called MQS-EC00. Specify the -e flag on the **crtmqm** command to create a queue manager with more than one EC. The number of EC processes may be changed after the queue manager has been created by adding or deleting EC process server classes, and making a corresponding modification to the `ExpectedNumECs` entry in the `ECBoss` stanza in the QMINI file.

Each EC process *must* be in its own server class. Use the MQS-EC00 server class as a template if you need to create additional EC processes manually.

Each CPU which hosts an EC must also host a repository manager. When a queue manager is created, MQSeries creates a repository manager server class (with names of the form MQS-REPSVR00) for each EC. If you manually add EC server classes in CPUs that did not previously host ECs, you should use the MQS-REPSVR00 server class as a template to create a new repository manager server class in the new CPU.

MQSeries requires a repository manager server class in each CPU where any of the following are true:

- The CPU hosts an EC, or
- The CPU hosts the MQS-QMGRSVR00 server class, or
- The CPU runs FASTPATH bound applications, or

- Users of the CPU execute any of the following MQSeries applications: **runmqsc**, **runmqchi**, **runmqchl**, or **mqrepdmp**.

The default assignment of CPUs to EC processes, or any other server class, may be changed using the PATHCOM ALTER SERVER command with the CPU attribute.

The default status server is automatically created by **crtmqm** in the server class MQS-STATUS00. By default, the only CPU assignment made is the primary in CPU 0 and the backup in CPU 1, or CPUS(0:1). The CPU assignment for the MQS-STATUS00 server can be changed using the PATHCOM ALTER SERVER command. You can specify a specific backup CPU for the status server by providing two CPU numbers separated by a colon, for example CPUS(2:12). In this case, PATHMON creates the primary in CPU 2 and the backup in CPU 12. If a specific backup CPU is not provided, the Compaq NSK operating system decides where to create the backup.

The default queue server is also automatically created by **crtmqm** in the server class MQS-QUEUE00. By default, the only CPU assignment made is the primary in CPU 0 and the backup in CPU 1, or CPUS(0:1). The CPU assignment for the MQS-QUEUE00 server can be changed using the PATHCOM ALTER SERVER command. You can specify a specific backup CPU for the queue server by providing two CPU numbers separated by a colon, for example CPUS(2:12). In this case, PATHMON creates the primary in CPU 2 and the backup in CPU 12. If a specific backup CPU is not provided, the Compaq NSK operating system decides where to create the backup.

## Addition of new status server processes

To add additional status servers to a queue manager, create a server class using the default status server class MQS-STATUS00 as a template.

The name of any new status server class should begin with the character string MQS-STATUS. If the server class names do not follow this naming convention, **strmqm** will not start them automatically on queue manager startup, and access to any objects that are configured for these status server classes will be disabled.

If additional status servers are configured, they each need unique process names. You are also recommended to configure them to run in different CPUs in order to benefit from the scalability that the status server architecture provides.

## Addition of new queue server processes

To add additional queue servers to a queue manager, create a server class using the default queue server class MQS-QUEUE00 as a template.

The name of any new queue server class should begin with the character string MQS-QUEUE. If the server class names do not follow this naming convention, **strmqm** will not start them automatically on queue manager startup, and access to any objects that are configured for these queue server classes will be disabled.

If additional queue servers are configured, they each need unique process names. You are also recommended to configure them to run in different CPUs in order to benefit from the scalability that the queue server architecture provides.

## Specifying the refresh frequency of MQM monitor panels

The MQMQMREFRESHINT PATHWAY parameter for MQS-MQMSVR00 determines the frequency with which monitor screens for channels and queues are refreshed. The default frequency is every 30 seconds. To change the frequency to every 10 seconds, for example, enter from the PATHWAY for your queue manager:

```
ALTER SERVER MQS-MQMSVR00, PARAM MQMQMREFRESHINT 10
```

## PATHWAY configuration for a queue manager

Here is an example PATHWAY configuration for a queue manager. This example was generated by issuing a sequence of INFO commands on the objects in a default queue manager configuration.

```
TCP MQS-TCP-01
  AUTORESTART 0
  CHECK-DIRECTORY OFF
  CODEAREALEN 80000
  CPUS 0:1
  DEBUG OFF
  DUMP OFF
  HIGHPIN ON
  HOMETERM \RAPTOR.$ZTN0.#PTY001C
  INSPECT OFF
  MAXINPUTMSGLEN 6000
  MAXINPUTMSGS 0
  MAXPATHWAYS 0
  MAXREPLY 32000
  MAXSERVERCLASSES 1
  MAXSERVERPROCESSES 10
  MAXTERMDATA 500000
  MAXTERMS 10
  NONSTOP 0
  POWERONRECOVERY ON
  PRI 175
  PROGRAM \RAPTOR.$SYSTEM.SYSTEM.PATHTCP2
  SERVERPOOL 32000
  STATS OFF
  TCLPROG \RAPTOR.$DEV.ZMQSEXE.POBJ
  TERMBUF 1500
  TERMPOOL 10000
```

*Figure 3. Example PATHWAY configuration (Part 1 of 13)*

```
 PROGRAM MQMC
  ERROR-ABORT OFF
  OWNER \RAPTOR.44,1
  SECURITY "N"
  TCP MQS-TCP-01
  TMF ON
  TYPE T16-6520 (BREAK OFF,ECHO ON,EXCLUSIVE OFF,INITIAL MAINC,IOPROTOCOL
0,MAXINPUTMSGS 0,TRAILINGBLANKS ON)
```

*Figure 3. Example PATHWAY configuration (Part 2 of 13)*

```
SERVER MQS-CHANINIT00
  PROCESSTYPE GUARDIAN
  AUTORESTART 10
  CPUS (0,1)
  CREATEDELAY 1 MINS
  DEBUG OFF
  DELETEDELAY 10 MINS
  HIGHPIN ON
  HOMETERM \RAPTOR.$ZTN0.#PTY001C
  LINKDEPTH 255
  MAXSERVERS 1
  NUMSTATIC 1
  OUT \RAPTOR.$ZTN0.#PTY001C
  OWNER \RAPTOR.44,1
  PARAM MQQUEMGRNAME "p101"
  PARAM MQMACHINIFILE "$DATA1.p101D.UMQSINI"
  PARAM MQDEFAULTPREFIX "$DEV"
  PRI 175
  PROGRAM \RAPTOR.$DEV.ZMQSEXE.RUNMQCHI
  SECURITY "N"
  TMF ON
  VOLUME \RAPTOR.$DATA1.P101D
```

*Figure 3. Example PATHWAY configuration (Part 3 of 13)*

```
SERVER MQS-CMDSERV00
  PROCESSTYPE GUARDIAN
  AUTORESTART 10
  CPUS (0,1)
  CREATEDELAY 1 MINS
  DEBUG OFF
  DELETEDELAY 10 MINS
  HIGHPIN ON
  HOMETERM \RAPTOR.$ZTN0.#PTY001C
  LINKDEPTH 255
  MAXSERVERS 1
  NUMSTATIC 1
  OUT \RAPTOR.$ZTN0.#PTY001C
  OWNER \RAPTOR.44,1
  PARAM MQQUEMGRNAME "p101"
  PARAM MQMACHINIFILE "$DATA1.p101D.UMQSINI"
  PARAM MQDEFAULTPREFIX "$DEV"
  PRI 175
  PROGRAM \RAPTOR.$DEV.ZMQSEXE.MQCMDSVR
  SECURITY "N"
  TMF ON
  VOLUME \RAPTOR.$DATA1.P101D
```

*Figure 3. Example PATHWAY configuration (Part 4 of 13)*

```
SERVER MQS-EC00
  PROCESSTYPE GUARDIAN
  AUTORESTART 10
  CPUS (0,1)
  CREATEDELAY 1 MINS
  DEBUG OFF
  DELETEDELAY 10 MINS
  HIGHPIN ON
  HOMETERM \RAPTOR.$ZTN0.#PTY001C
  LINKDEPTH 255
  MAXSERVERS 1
  NUMSTATIC 1
  OUT \RAPTOR.$ZTN0.#PTY001C
  OWNER \RAPTOR.44,1
  PARAM MQQUEMGRNAME "p101"
  PARAM MQMACHINIFILE "$DATA1.p101D.UMQSINI"
  PARAM MQDEFAULTPREFIX "$DEV"
  PRI 175
  PROGRAM \RAPTOR.$DEV.ZMQSEXE.MQEC
  SECURITY "N"
  TMF ON
  VOLUME \RAPTOR.$DATA1.P101D
```

*Figure 3. Example PATHWAY configuration (Part 5 of 13)*

```
SERVER MQS-ECBOSS
  PROCESSTYPE GUARDIAN
  AUTORESTART 10
  CPUS (0,1)
  CREATEDELAY 1 MINS
  DEBUG OFF
  DELETEDELAY 10 MINS
  HIGHPIN ON
  HOMETERM \RAPTOR.$ZTN0.#PTY001C
  LINKDEPTH 255
  MAXSERVERS 1
  NUMSTATIC 1
  OUT \RAPTOR.$ZTN0.#PTY001C
  OWNER \RAPTOR.44,1
  PARAM MQQUEMGRNAME "p101"
  PARAM MQMACHINIFILE "$DATA1.p101D.UMQSINI"
  PARAM MQDEFAULTPREFIX "$DEV"
  PRI 175
  PROGRAM \RAPTOR.$DEV.ZMQSEXE.MQECBOSS
  SECURITY "N"
  TMF ON
  VOLUME \RAPTOR.$DATA1.P101D
```

*Figure 3. Example PATHWAY configuration (Part 6 of 13)*

```
SERVER MQS-MQMSVR00
 PROCESSTYPE GUARDIAN
 AUTORESTART 0
 CPUS (0,1)
 CREATEDELAY 1 MINS
 DEBUG OFF
 DELETEDELAY 10 MINS
 HIGHPIN ON
 HOMETERM \RAPTOR.$ZTN0.#PTY001C
 LINKDEPTH 255
 MAXSERVERS 1
 NUMSTATIC 1
 OUT \RAPTOR.$ZTN0.#PTY001C
 OWNER \RAPTOR.44,1
 PARAM MQQUEMGRNAME "p101"
 PARAM MQMACHINIFILE "$DATA1.p101D.UMQSINI"
 PARAM MQDEFAULTPREFIX "$DEV"
 PARAM MQMPAGESTORETRIEVE "20"
 PRI 175
 PROGRAM \RAPTOR.$DEV.ZMQSEXE.MQMSVR
 SECURITY "N"
 TMF ON
 VOLUME \RAPTOR.$DATA1.P101D
```

*Figure 3. Example PATHWAY configuration (Part 7 of 13)*

```
SERVER MQS-STATUS00
 PROCESSTYPE GUARDIAN
 AUTORESTART 0
 CPUS (0:1)
 CREATEDELAY 1 MINS
 DEBUG OFF
 DELETEDELAY 10 MINS
 HIGHPIN ON
 HOMETERM \RAPTOR.$ZTN0.#PTY001C
 LINKDEPTH 255
 MAXSERVERS 1
 NUMSTATIC 1
 OUT \RAPTOR.$ZTN0.#PTY001C
 OWNER \RAPTOR.44,1
 PARAM MQQUEMGRNAME "p101"
 PARAM MQMACHINIFILE "$DATA1.p101D.UMQSINI"
 PARAM MQDEFAULTPREFIX "$DEV"
 PRI 176
 PROCESS $P01S
 PROGRAM \RAPTOR.$DEV.ZMQSEXE.MQSTSVR
 SECURITY "N"
 TMF ON
 VOLUME \RAPTOR.$DATA1.P101D
```

*Figure 3. Example PATHWAY configuration (Part 8 of 13)*

```
SERVER MQS-TCPLIS00
  PROCESSTYPE GUARDIAN
  AUTORESTART 10
  CPUS (0,1)
  CREATEDELAY 1 MINS
  DEBUG OFF
  DELETEDELAY 10 MINS
  HIGHPIN ON
  HOMETERM \RAPTOR.$ZTN0.#PTY001C
  LINKDEPTH 255
  MAXSERVERS 1
  NUMSTATIC 1
  OUT \RAPTOR.$ZTN0.#PTY001C
  OWNER \RAPTOR.44,1
  PARAM MQQUEMGRNAME "p101"
  PARAM MQMACHINIFILE "$DATA1.p101D.UMQSINI"
  PARAM MQDEFAULTPREFIX "$DEV"
  PRI 175
  PROGRAM \RAPTOR.$DEV.ZMQSEXE.RUNMQLSR
  SECURITY "N"
  TMF ON
  VOLUME \RAPTOR.$DATA1.P101D
```

*Figure 3. Example PATHWAY configuration (Part 9 of 13)*

```
SERVER MQS-TRIGMON00
  PROCESSTYPE GUARDIAN
  AUTORESTART 10
  CPUS (0,1)
  CREATEDELAY 1 MINS
  DEBUG OFF
  DELETEDELAY 10 MINS
  HIGHPIN ON
  HOMETERM \RAPTOR.$ZTN0.#PTY001C
  LINKDEPTH 255
  MAXSERVERS 1
  NUMSTATIC 1
  OUT \RAPTOR.$ZTN0.#PTY001C
  OWNER \RAPTOR.44,1
  PARAM MQQUEMGRNAME "p101"
  PARAM MQMACHINIFILE "$DATA1.p101D.UMQSINI"
  PARAM MQDEFAULTPREFIX "$DEV"
  PRI 175
  PROGRAM \RAPTOR.$DEV.ZMQSEXE.RUNMQTRM
  SECURITY "N"
  TMF ON
  VOLUME \RAPTOR.$DATA1.P101D
```

*Figure 3. Example PATHWAY configuration (Part 10 of 13)*

```
SERVER MQS-QUEUE00
  PROCESSTYPE GUARDIAN
  AUTORESTART 10
  CPUS (0:1)
  CREATEDELAY 1 MINS
  DEBUG OFF
  DELETEDELAY 10 MINS
  HIGHPIN ON
  HOMETERM \RAPTOR.$ZTN0.#PTY001C
  LINKDEPTH 255
  MAXSERVERS 1
  NUMSTATIC 1
  OUT \RAPTOR.$ZTN0.#PTY001C
  OWNER \RAPTOR.44,1
  PARAM MQQUEMGRNAME "p101"
  PARAM MQMACHINIFILE "$DATA1.p101D.UMQSINI"
  PARAM MQDEFAULTPREFIX "$DEV"
  PRI 176
  PROGRAM \RAPTOR.$DEV.ZMQSEXE.MQQSSVR
  SECURITY "N"
  TMF ON
  VOLUME \RAPTOR.$DATA1.P101D
```

*Figure 3. Example PATHWAY configuration (Part 11 of 13)*

```
SERVER MQS-QMGRSVR
  PROCESSTYPE GUARDIAN
  AUTORESTART 10
  CPUS (0,1)
  CREATEDELAY 1 MINS
  DEBUG OFF
  DELETEDELAY 10 MINS
  HIGHPIN ON
  HOMETERM \RAPTOR.$ZTN0.#PTY001C
  LINKDEPTH 255
  MAXSERVERS 1
  NUMSTATIC 1
  OUT \RAPTOR.$ZTN0.#PTY001C
  OWNER \RAPTOR.44,1
  PARAM MQQUEMGRNAME "p101"
  PARAM MQMACHINIFILE "$DATA1.p101D.UMQSINI"
  PARAM MQDEFAULTPREFIX "$DEV"
  PRI 175
  PROGRAM \RAPTOR.$DEV.ZMQSEXE.MQMGRSVR
  SECURITY "N"
  TMF ON
  VOLUME \RAPTOR.$DATA1.P101D
```

*Figure 3. Example PATHWAY configuration (Part 12 of 13)*

```
SERVER MQS-REPSVR00
  PROCESSTYPE GUARDIAN
  AUTORESTART 10
  CPUS (0,1)
  CREATEDELAY 1 MINS
  DEBUG OFF
  DELETEDELAY 10 MINS
  HIGHPIN ON
  HOMETERM \RAPTOR.$ZTN0.#PTY001C
  LINKDEPTH 255
  MAXSERVERS 1
  NUMSTATIC 1
  OUT \RAPTOR.$ZTN0.#PTY001C
  OWNER \RAPTOR.44,1
  PARAM MQQUEMGRNAME "p101"
  PARAM MQMACHINIFILE "$DATA1.p101D.UMQSINI"
  PARAM MQDEFAULTPREFIX "$DEV"
  PARAM MQREPMANAGER "YES"
  PRI 175
  PROGRAM \RAPTOR.$DEV.ZMQSEXE.MQREPSVR
  SECURITY "N"
  TMF ON
  VOLUME \RAPTOR.$DATA1.P101D
```

*Figure 3. Example PATHWAY configuration (Part 13 of 13)*

## Changing the parameters of PATHWAY server classes

To alter the parameters of PATHWAY server classes:

1. Stop the queue manager by issuing the **endmqm** command. This also stops the PATHMON process.
2. Go to the subvolume `queue manager`D, which contains the PATHCTL file. For example:

```
>VOLUME $DATA2.MT01D
```

3. Start a PATHMON with the same name as the queue manager's PATHMON and with the NOWAIT option. For example:

```
>PATHMON /NAME $MT01, NOWAIT/
```

4. Start a PATHCOM against the new PATHMON. For example:

```
>PATHCOM $MT01
```

5. Load the existing PATHWAY configuration for the queue manager by issuing the following command from the PATHCOM prompt:

```
>START PATHWAY COOL
```

6. Make the required changes using PATHCOM commands.

7. Shut down the PATHWAY system by issuing the following command:

```
>SHUTDOWN2
```

8. Start the queue manager using the **strmqm** command.

## Adding user-defined server classes to an MQSeries PATHWAY

You can add your own server class definitions to the MQSeries PATHWAY configuration using PATHCOM. However, this is not recommended: servers *must* be well behaved, or **endmqm** does not function correctly. Note also that user-defined server class definitions are lost when a queue manager is deleted. To minimize inconvenience, you are recommended to create a reusable script.

# Chapter 4. Managing queue managers

This chapter describes all aspects of the management of MQSeries queue managers.

The following sections are in this chapter:

## Getting started

Before you use messages and queues, you must create at least one queue manager. Once you complete the installation process, you can use the MQSeries control commands to create a queue manager, create MQSeries principals, and start the queue manager.

When you create a queue manager, the required default objects and system objects are automatically created. Default objects form the basis of any object definitions that you make; system objects are required for queue manager operation. See "Restoring the default and system objects" on page 61 for information about restoring the default system objects.

See "Chapter 3. Using the MQSeries command sets" on page 27 for more information about commands that you can use with MQSeries for Compaq NSK, and the different methods of invoking them.

## Guidelines for creating a queue manager

A queue manager manages the resources associated with it, such as the queues that it owns. A queue manager provides queuing services to applications for Message Queuing Interface (MQI) calls and commands to create, modify, display, and delete MQSeries objects. You create a queue manager using the **crtmqm** command. Here is a list of items to consider when creating a queue manager:

- **Specify a unique queue manager name.**

© Copyright IBM Corp. 1993, 2001

## Creating queue managers

When you create a queue manager, you must ensure that no other queue manager has the same name in your network. Queue manager names are not checked at create time, and non-unique names prevent you from creating channels for distributed queuing.

You can ensure uniqueness by prefixing each queue manager name with its own node name. For example, if a node is called `accounts`, you can name your queue manager `accounts.saturn.queue.manager`, where `saturn` identifies a particular queue manager and `queue.manager` is an extension you can give to all queue managers. Alternatively, you can omit this extension; however, `accounts.saturn` and `accounts.saturn.queue.manager` are *different* queue manager names.

**Note:** Queue manager names in control commands are case sensitive. For example, you can create two queue managers with the names. `jupiter.queue.manager` and `JUPITER.queue.manager`.

- **Limit the number of queue managers.**

  In MQSeries for Compaq NSK, you can create as many queue managers as resources allow. However, because each queue manager requires its own resources, it is often more efficient to have one queue manager with 100 queues than ten queue managers with ten queues each. Many nodes can be run with a single queue manager; however, larger servers can run with multiple queue managers. There can be special requirements of either performance, or functionality that would require multiple queue managers.

- **Specify a default queue manager.**

  Each node should have a default queue manager, though it is possible to configure MQSeries on a node without one.

  The default queue manager is the queue manager that applications connect to if they do not specify a queue manager name in an MQCONN call. It is also the queue manager that processes MQSC commands when you invoke the **runmqsc** command without specifying a queue manager name.

  Specifying a queue manager as the default *replaces* any existing default queue manager specification for the node.

  If you change the default queue manager it can affect other users or applications. The change has no effect on currently-connected applications because they can use the handle from their original connect call in any further MQI calls. This handle ensures that the calls are directed to the same queue manager. Any applications connecting after the change connect to the new default queue manager.

  To create a default queue manager, specify the -q flag on the **crtmqm** command. For a detailed description of this command and its parameters, see "crtmqm (Create queue manager)" on page 242.

- **Specify a dead-letter queue.**

  The dead-letter queue is a local queue where messages are put if they cannot be routed to their correct destination.

  **Attention:** You should have a dead-letter queue on each queue manager in your network. Failure to do so can result in application program errors, which causes channels to be closed and causes replies to administration commands to fail. For example, if an application attempts to put a message on a queue on another queue manager, but the wrong queue name is given, the channel is stopped, and the message remains on the transmission queue. Other applications cannot then use this channel for their messages.

  The channels are not affected if the queue managers have dead-letter queues. The undelivered message is put on the dead-letter queue at the receiving end, leaving the channel and its transmission queue available.

Therefore, when you create a queue manager you should use the -u flag to specify the name of the dead-letter queue. You can also use an MQSC command to alter the attributes of a queue manager and specify the dead-letter queue to be used. See "Altering queue manager attributes" on page 87 for an example of an MQSC ALTER command.

When you find messages on a dead-letter queue, you can use the dead-letter queue handler, which is supplied with MQSeries, to process these messages. See "Chapter 9. MQSeries dead-letter queue handler" on page 145 for further information about the dead-letter queue handler, and how to reduce the number of messages that might otherwise be placed on the dead-letter queue.

- **Specify a default transmission queue.**

  A transmission queue is a local queue on which messages in transit to a remote queue manager are queued pending transmission. The default transmission queue is the queue that is used when no transmission queue is explicitly defined. Each queue manager can be assigned a default transmission queue.

  When you create a queue manager you should use the -d flag to specify the name of the default transmission queue. The -d flag does not actually create the queue, which you have to create at a later time. See "Working with local queues" on page 92 for more information.

- **Back up configuration files after creating a queue manager.**

  The MQSeries configuration file (MQSINI) is created when you install MQSeries. This file contains a list of queue managers that is updated each time you create or delete a queue manager. There is one MQSINI file per installation. By default, MQSINI is located in $SYSTEM.ZMQSSYS.

  A queue manager configuration file (QMINI) is automatically created when you create a new queue manager. This file contains configuration parameters for the queue manager.

  You should make a backup of these files. If you create another queue manager that causes problems, you can reinstate the backups when you have removed the source of the problem. You should back up your configuration files each time you create a new queue manager.

  For more information about configuration files, see "Chapter 13. Configuration files" on page 173.

## Backing up configuration files after creating a queue manager

There are two configuration files to back up, MQSINI and QMINI:

1. The MQSeries configuration file (MQSINI) is created when you install MQSeries. This file contains a list of queue managers that is updated each time you create or delete a queue manager. There is one MQSINI file per installation. By default, MQSINI is located in $SYSTEM.ZMQSSYS.
2. A queue manager configuration file (QMINI) is automatically created when you create a new queue manager. This file contains configuration parameters for the queue manager.

You should make a backup of these files. If you create another queue manager that causes problems, you can reinstate the backups when you have removed the source of the problem. You should back up your configuration files each time you create a new queue manager.

# Modifying queue manager properties

Many of the properties of a queue manager can be modified when the queue manager is created. Some properties can also be modified after the queue manager is created, though you are usually required to stop and restart the queue manager before the changes can take effect.

The remainder of this section describes some queue manager properties that you might want to change.

## Home volume of the queue manager

This is the volume where all databases, including queues, are created. (However, individual queues may be moved to a different volume after creation using the **altmqfls** control command, as described in "altmqfls (Alter queue file attributes)" on page 230.)

The default value is taken from the `QMDefaultVolume` entry of the `AllQueueManagers` stanza in the MQSINI file. It is overridden by the -p *`DefaultPrefix`* parameter of the **crtmqm** command, if specified.

The home volume can be specified *only* when a queue manager is created. It cannot be changed after creation.

## Number of EC processes per queue manager

By default, there is one EC process per queue manager. You specify the number of EC processes for a queue manager on the -e *`NumECs`* parameter of the **crtmqm** command.

Each EC is responsible for a subset of the server processes that perform messaging and queuing for applications and channels in the same CPU as the EC itself. You are recommended to have 1 EC per CPU, unless the number of applications per EC is large, in which case having an additional EC running on the CPU would be beneficial. For large installations, for example, more EC processes are desirable (often distributed across multiple CPUs) so that large numbers of applications and channels can be handled concurrently.

During queue manager creation, a TS/MP server class is created for each EC specified on the **crtmqm** command. The specified EC server classes are distributed across all CPUs in the system, in a round-robin fashion. For example, specifying eight EC processes in a four-CPU system would result in two EC processes per CPU by default.

The `ExpectedNumECs` field of the `ECBoss` stanza in the QMINI file of the queue manager is set to the number of EC processes specified on creation. This value must be consistent with the TS/MP configuration at all times.

It is possible to change the number of EC processes in a queue manager after creation by adding or deleting TS/MP server classes, and modifying the `ExpectedNumECs` entry of the `ECBoss` stanza in the QMINI file. This can be done only while the queue manager is stopped.

### System load balancing

The EC Boss is responsible for distributing the workload of a queue manager among the ECs. The processing load of a queue manager can be distributed among multiple CPUs in a balanced way, given an appropriate configuration of ECs.

When a new connection request arrives from a local application, or when a channel is to be started, the EC Boss allocates the request to the EC with the smallest number of active LQMAs and MCAs.

## Home terminal of the queue manager

All Compaq NSK processes, including the queue manager server processes, have a home terminal. The terminal *must* exist and be in the paused state. In general, the queue manager home terminal is not used for output. The home terminal can be any valid terminal device, including the Compaq Virtual Hometerm Service (VHS) product.

Compaq NonStop Kernel allows up to 255 primary openers of a physical terminal. Therefore, careful planning is required to ensure that this limit is not exceeded. You are recommended to use Compaq's VHS product if support for more than 255 openers is required.

You identify a queue manager's home terminal on the -o *`HomeTerminalName`* parameter of the **crtmqm** command. There is no default; this parameter is mandatory.

The HOMETERM and OUT attributes of all TS/MP server classes are set to the specified terminal device. These attributes may be altered at any time when server classes are in the stopped state, normally when the queue manager is stopped.

The `HomeTerminalName` entry in the `Configuration` stanza in the QMINI file must also be modified in order to change the home terminal of a queue manager.

## The PATHMON process name for the queue manager

Each queue manager runs under its own TS/MP (Pathway) configuration. The controlling process for this is the PATHMON process. A unique name must be specified for each queue manager. Furthermore, the name must be unique within the system.

You specify the PATHMON process name on the -n *`PATHMONProcessName`* parameter of the **crtmqm** command. There is no default; this parameter is mandatory.

Specify a process name that is unique in the system, and is easy to associate with the queue manager it controls.

You can change the PATHMON process name for a queue manager, as follows:

1. Stop the queue manager.
2. Set your default volume and subvolume to the location of the queue manager data files (normally `<QMgr name>D`).
3. Modify the `PathmonProcName` entry in the queue manager's QMINI file to specify the new process name.
4. Run PATHMON up for the queue manager, using the *new* process name.

   From TACL, execute the following command:

```
PATHMON /name $<newname>, nowait/
```

5. Run PATHCOM against the newly named PATHMON.

From TACL, execute the following command:

```
PATHCOM $<newname>
```

6. Load the queue manager Pathway configuration and confirm the change of name of the PATHMON process.

   From PATHCOM, execute the following command:

   ```
   START PATHWAY COOL
   ```

   As the configuration is loading, you will be warned that the name of the new PATHMON process is different from the one stored in the configuration file. After this, you will be asked to confirm whether you want to proceed. Type `y` at the prompt, and the configuration loading will complete.
7. Save the new PATHWAY configuration information back to the database.

   From PATHCOM, execute the following commands:

   ```
   SHUTDOWN2
   EXIT
   ```

The `PathmonProcName` entry in the `Configuration` stanza of the QMINI file must also be changed.

The PATHMON process name change is now complete. The next **strmqm** will start the queue manager using the new PATHMON process name.

## The CCSID of the queue manager

This is the Coded Character Set ID (CCSID) of the character set that is used by the queue manager to store information about messages.

You specify the CCSID on the -l *CCSID* parameter of the **crtmqm** command. The default is 819.

The CCSID of the queue manager can be changed at any time after queue manager creation using **runmqsc**, the MQM facility, or PCF commands.

## Controlling EBCDIC data conversion

The way in which EBCDIC new line (NL) characters are handled during conversion can be controlled using the `ConvEBCDICNewline` item in the `AllQueueManagers` stanza of the MQSINI configuration file.

**ConvEBCDICNewline=NL_TO_LF|TABLE|ISO**

EBCDIC code pages contain a new line (NL) character that is not supported by ASCII code pages; although some ISO variants of ASCII do contain an equivalent. Use the `ConvEBCDICNewline` attribute to specify the method MQSeries is to use when converting the EBCDIC NL character into ASCII format.

**NL_TO_LF**

Specify NL_TO_LF if you want the EBCDIC NL character (X’15’) converted to the ASCII line feed character, LF (X'0A'), for all EBCDIC to ASCII conversions. NL_TO_LF is the default.

**TABLE**

Specify TABLE if you want the EBCDIC NL character converted according to the conversion tables used on your platform for all EBCDIC to ASCII conversions. Note that the effect of this type of conversion may vary from platform to platform and from language to language; while on the same platform, the behavior may vary if you use different CCSIDs.

**ISO** Specify ISO if you want:

- ISO CCSIDs to be converted using the TABLE method.
- All other CCSIDs are to be converted using the NL_TO_CF method.

Possible ISO CCSIDs are shown in Table 2.

*Table 2. List of possible ISO CCSIDs*

| **CCSID** | **Code Set** |
|---|---|
| 819 | ISO8859-1 |
| 912 | ISO8859-2 |
| 915 | ISO8859-5 |
| 1089 | ISO8859-6 |
| 813 | ISO8859-7 |
| 916 | ISO8859-8 |
| 920 | ISO8859-9 |
| 1051 | roman8 |

If the ASCII CCSID is not an ISO subset, `ConvEBCDICNewline` defaults to NL_TO_LF.

For more information about data conversion, see the *MQSeries Application Programming Guide*.

## The EMS Collector for the queue manager

The queue manager can be configured to use an alternative collector if required. EMS Events are sent to $0 by default. The `EMSCollectorName` entry in the `Configuration` stanza in the QMINI file specifies the name of the EMS Collector for this queue manager.

The EMS collector can be changed at any time by modifying the value of this entry, though it does not take effect until the queue manager has been restarted.

## The pool of agents kept ready by each EC in the queue manager

For each of the four basic types of agent, an EC can maintain a pool of idle agent processes, ready to be assigned to new work. The size of these pools can be configured in order to achieve an appropriate balance between response time to new work and resource utilization. The values of the following fields of the `Configuration` stanza in the QMINI file can be modified to specify a different number of processes to be kept idle:

*MinIdleMCALU62Responders*
Specifies the minimum number of SNA LU 6.2 responder MCAs to maintain in an idle state. The default value is 0.

*MinIdleMCATCPResponders*
Specifies the minimum number of TCP/IP responder MCAs to maintain in an idle state. The default value is 0.

*MinIdleMCACallers*
Specifies the minimum number of caller MCAs (not protocol specific) to maintain in an idle state. The default value is 0.

*MinIdleLQMAgents*
Specifies the minimum number of local queue manager agents (LQMAs) to maintain in an idle state. The default value is 1.

Note that the number of processes specified in these fields applies to each EC, not to each queue manager. Therefore, for a two-EC queue manager, there is a minimum of two idle LQMAs by default.

These values can be changed at any time, though the change does not take effect until the queue manager is restarted.

## Maximum idle agents and process reuse

By default, a queue manager allows up to 10 agent processes of each type to be idle. This value is controlled by the `MaxIdleAgents` entry in the `Configuration` stanza of the QMINI file.

The `MaxIdleAgentReuse` entry determines the number of times an agent process can be reused before it is replaced by a new agent process. By default, `MaxIdleAgentReuse` is set to 10.

## Process priority of queue manager processes

The priorities may need to be changed to balance resources between MQSeries and other applications. The process priorities of the TS/MP server classes may be changed by ALTERing the TS/MP objects when the queue manager is stopped.

The process priorities of the agent processes may be changed by editing the `MCAAgentPriority` and `LQMAgentPriority` fields of the `EC` stanza of the QMINI file.

Apart from the status servers and queue servers, which have a default process priority of 176, the TS/MP configured processes are all given a default priority of 175. By default, both MCAs and LQMAs have a process priority of 165.

Ensure that the status servers and queue servers have the highest priority, followed by the EC Boss and EC, which in turn must have a higher process priority than the MCAs and LQMAs.

## Maximum number of channels for the queue manager

There is a limit to the number of channels that may be controlled at any one time for a queue manager. If the limit is too high, performance may be affected as this parameter dictates the size of the channel status table, on which numerous search operations are performed. If the limit is too low, you may not be able to control enough channels for your application. The `MaxChannels` field of the `Channels` stanza in the QMINI file defines the maximum number of channels that can be controlled simultaneously.

The default on creation is 10. There is no way to override the default on creation.

The MaxChannels entry in the Channels stanza of the QMINI file can be changed at any time, though the change does not take effect until the queue manager is restarted.

## Maximum number of active channels for the queue manager

There is a limit to the number of concurrently active (running) channels in a queue manager. This may be used to control the peak demand on system resources by channels. The MaxActiveChannels entry in the Channels stanza in the QMINI file defines the maximum number of active channels for the queue manager.

The default on creation is 10. There is no way to override the default on creation.

The MaxActiveChannels entry in the QMINI file can be changed at any time, though the change does not take effect until the queue manager is restarted.

## Guardian segment IDs used by MQSeries

MQSeries allocates Guardian memory segments both in its own processes and in the application program's process. The Guardian Segment IDs used by MQSeries for these segments are allocated from a range defined in the MQSINI configuration file. The NSKSegidRange item in the AllQueueManagers stanza of the MQSINI configuration file defines the range of Segment IDs used by MQSeries when allocating memory segments.

When MQSeries is installed, this range is set, by default, to NSKSegidRange=10-20.

Since these memory segments are also allocated in the application program's process, you should ensure that applications do not allocate segments in this range. You can force MQSeries to allocate its segments in a different range by editing the NSKSegidRange item in your MQSINI file. After editing this item, you must stop and restart all queue managers before the new value will take effect.

The range defined by the NSKSegidRange parameter must be wholly contained within the Guardian Segment ID limits of 0-1023. The NSKSegidRange parameter must define a range containing at least 10 Segment IDs for use by MQSeries.

## Default TCP/IP port

The TCPPort entry in the TCPConfig stanza in QMINI defines the default port number for outgoing channels. By default, port number 1414 is used. This default is overridden by port-number values specified in the CONNAME field for a channel.

## TCP/IP ports listened on by the queue manager

A queue manager with TCP/IP channels may be configured to listen for incoming connections on one or more TCP/IP ports. The TCPNumListenerPorts and entries in the TCPConfig stanza in the QMINI file define how many ports to listen on, and the numbers of the ports assigned to this queue manager. For examples of the QMINI entries, see "TCP/IP communications example" on page 367.

There can be multiple queue managers on a single system. Each queue manager on a system must be assigned nonoverlapping sets of TCP/IP ports to listen on. The set of TCP/IP ports for each queue manager may be just one port, where the rate of incoming TCP/IP connect requests is low, or may be more than one port for large configurations. The default TCP/IP port is 1414 and, by default, a queue manager is created to listen on only this port.

The list of listening ports may be changed by editing the `TCPConfig` stanza in the QMINI file and restarting the queue manager. In order to listen on more than one port, a queue manager must also be configured with additional TCP/IP listener server classes using TS/MP. This operation is performed manually using PATHCOM.

Alternatively, a port number can be specified on the **runmqlsr** command (described in “runmqlsr (Run listener)” on page 271). The `TCPListenerPort` values are overridden by a Listener server class program if the parameter MQLISTENPORTNUM is present in the environment of that Listener program.

## TCP/IP process used by the queue manager

The interface to the Compaq TCP/IP product is via a server process, known as the TCP/IP process. By default, the system default, $ZTC0, is used. There is no way to override this default when the queue manager is created. A queue manager ’s channels can be configured to use a specific TCP/IP process, if the system default is not sufficient. The TCP/IP Listener’s TS/MP server class configuration must be manually changed if the system default TCP/IP process, $ZTC0, is not sufficient or correct. This server class configuration can also be manually changed to enable a Listener to listen on a specific port and override the ports defined in the QMINI file.

Server class MQS-TCPLIS00 must have the DEFINE TCPIP^PROCESS^NAME added to reference the required alternative TCP/IP process name. Note that, if you have multiple ECs, you must update all of them. Refer to the Compaq NSK TCP/IP product manuals for further information.

The change to the TS/MP server classes can be made only when the queue manager is stopped.

MQSeries for Compaq NSK can support multiple TCP/IP process per queue manager. Also, multiple TCP/IP ports and listener processes are supported.

## Reconfiguring the MQS-TCPLISnn server class for a nondefault TCP/IP process and port

To reconfigure the MQS-TCPLISnn server class for a nondefault TCP/IP process and port, follow the general instructions in “Changing the parameters of PATHWAY server classes” on page 39.

To specify a nondefault TCP/IP process, use the following PATHCOM commands:

```
ALTER SERVER MQS-TCPLIS00, (DEFINE =TCPIP^PROCESS^NAME, FILE $ZZZZ)
```

where `$ZZZZ` is the name of the required TCP/IP process.

To specify a specific port, use the following PATHCOM commands:

```
ALTER SERVER MQS-TCPLIS00, PARAM MQLISTENPORTNUM "nnnn"
```

where `nnnn` is the number of the port on which you want to listen.

For listeners running from TACL (**runmqlsr**) that require a different TCP/IP process name from the default $ZTC0, add the following to the TACL environment from where that listener is going to run:

```
ADD DEFINE =TCPIPˆPROCESSˆNAME, FILE $ZZZZ)
```

where `$ZZZZ` is the name of the required TCP/IP process.

**Note:** If individual TCP/IP listener server classes want to use different TCP/IP processes with each port, they must define both the port and TCP/IP process name in each server class instance, otherwise there is no guarantee which TCP/IP process a port defined in the QMINI file will use.

If the parameter MQLISTENPORTNUM is not defined in the listener program's environment the Listener obtains the port from the QMINI file.

## Swap space allocation

MQSeries for Compaq NSK allocates swap space according to the `ExtPoolSize` values for the various executables in the QMINI configuration file. Therefore, if your queue manager is using the default QMINI file, which allocates 300 KB to each executable by default, and is running 10 outbound channels, 20 agents, and 10 TCP responders, you need at least 12 MB (40 * 300 KB) of swap space. To reduce this requirement, you can lower the values in the QMINI file. The `ExtPoolSize` values are the minimum additional memory allocated when the initial memory allocation is exhausted. The value does not have to be larger than the maximum message size for the queue manager.

A way of controlling the swap allocation of MQSeries executables is to alter the MQ Pathway server classes by adding:

```
DEFINE =_DEFAULTS, CLASS DEFAULTS, VOLUME volume.qmD subvolume, SWAP volume
```

By adding the DEFINE, agent processes created by the EC server also inherit the defined swap volume thereby creating some scalability of swap utilization when multiple EC server classes are used.

## Default status server name

A unique process name must be specified for the default status server process pair when you create a queue manager. You specify the default status server name on the mandatory -s *StatusServerName* parameter of the **crtmqm** command. There is no default value for this parameter. Specify a process name that is both unique in the system and easy to associate with the queue manager to which it belongs.

You can change the default status server process name for a queue manager as follows:

1. Stop the queue manager.
2. Set your default volume and subvolume to the location of the queue manager data files (usually <QMgrName>D).

3. Edit the DefaultStatusServerName entry in the Configuration stanza of the queue manager's QMINI file to record the new process name.
4. Run PATHMON for the queue manager by entering the following command from TACL:

   ```
       PATHMON /name $<pmon>, nowait/
   ```

   where <pmon> is the name of the PATHMON process for the queue manager.
5. Run PATHCOM against the PATHMON process by entering the following command from TACL:

   ```
       PATHCOM $<pmon>
   ```

6. Load the queue manager Pathway configuration by entering the following command from PATHCOM:

   ```
       START PATHWAY COOL
   ```

7. Alter the server MQS-STATUS00 and reset the process by entering the following commands from PATHCOM:

   ```
       ALTER SERVER MQS-STATUS00, DELETE PROCESS $<oldname>
       ALTER SERVER MQS-STATUS00, PROCESS $<newname>
   ```

8. Save the altered configuration back to disk by entering the following commands from PATHCOM:

   ```
       SHUTDOWN2
       EXIT
   ```

The default status server process name change is now complete. The next **strmqm** command starts the queue manager using the new default status server process name.

## Adding and removing nondefault status servers

To add additional status servers, use the existing default status server as a template. The queue manager does not need to be stopped to allow you to add a new status server.

The following procedure adds a new status server:

**Note:** This procedure assumes that the queue manager is running. If this is not the case, you must start PATHMON and load the PATHWAY configuration before starting this procedure. You must also omit step 5 on page 55, and save the Pathway configuration to disk at the end using a SHUTDOWN2 command.

1. Run PATHCOM against the PATHMON process by entering the following command from TACL:

```
PATHCOM $<pmon>
```

2. Create a working set of attributes based on the default status server class as a template by entering the following commands from PATHCOM:

```
RESET SERVER
SET LIKE MQS-STATUS00
```

3. Modify the working set for the new server by entering the following commands from PATHCOM:

```
SET SERVER CPUS(n:m)
RESET SERVER PROCESS $<default status server process name>
SET SERVER PROCESS $<new status server name>
```

4. Add the new server, giving it a new server class name by entering the following command from PATHCOM:

```
ADD SERVER MQS-STATUSxx
```

5. Start the new status server, so that it can be used, and exit from PATHCOM by entering the following commands:

```
START SERVER MQS-STATUSxx
EXIT
```

The recommended naming convention for additional server classes is MQS-STATUS01, MQS-STATUS02, and so on. However, there is no requirement to use this convention. Provided that the server class name begins with the character string MQS-STATUS, the server class will be started by **strmqm**.

Once a status server has been added and started (either explicitly using PATHCOM or implicitly by **strmqm**), objects can be reassigned to the new status server using **altmqfls**. For more information about reassigning objects, see “Reassigning objects to status servers and queue servers” on page 96.

Before removing a status server, check that all objects configured against this status server have been either deleted or reassigned to a different status server. You must **not** delete the default status server, otherwise the queue manager will become inoperable.

## Default queue server name

A unique process name must be specified for the default queue server process pair when you create a queue manager. You specify the default queue server name on the mandatory -v QueueServer parameter of the **crtmqm** command. There is no default value for this parameter. Specify a process name that is both unique in the system and easy to associate with the queue manager to which it belongs.

You can change the default queue server process name for a queue manager as follows:

1. Stop the queue manager.
2. Set your default volume and subvolume to the location of the queue manager data files (usually <QMgrName>D).
3. Edit the `DefaultQueueServerName` entry in the Configuration stanza of the queue manager′s QMINI file to record the new process name.
4. Run PATHMON for the queue manager by entering the following command from TACL:

   ```
   PATHMON /name $<pmon>, nowait/
   ```

   where `<pmon>` is the name of the PATHMON process for the queue manager.
5. Run PATHCOM against the PATHMON process by entering the following command from TACL:

   ```
   PATHCOM $<pmon>
   ```

6. Load the queue manager Pathway configuration by entering the following command from PATHCOM:

   ```
   START PATHWAY COOL
   ```

7. Alter server mqs-queue*nn* and reset the process by entering the following commands from PATHCOM:

   ```
   ALTER SERVER MQS-QUEUEnn, DELETE PROCESS $<oldname>
   ALTER SERVER MQS-QUEUEnn, PROCESS $<newname>
   ```

8. Save the altered configuration back to disk by entering the following commands from PATHCOM:

   ```
   SHUTDOWN2
   EXIT
   ```

The default status server process name change is now complete. The next **strmqm** command starts the queue manager using the new default status server process name.

## Adding and removing nondefault queue servers

To add additional queue servers, use the existing default queue server as a template. The queue manager does not need to be stopped to allow you to add a new queue server.

The following procedure adds a new queue server:

**Note:** This procedure assumes that the queue manager is running. If this is not the case, you must start PATHMON and load the PATHWAY configuration before starting this procedure. You must also omit step 5, and save the PATHWAY configuration to disk at the end using a SHUTDOWN2 command.

1. Run PATHCOM against the PATHMON process by entering the following command from TACL:

```
PATHCOM $<pmon>
```

2. Create a working set of attributes based on the default queue server class as a template by entering the following commands from PATHCOM:

```
RESET SERVER
SET LIKE MQS-QUEUEnn
```

3. Modify the working set for the new server by entering the following commands from PATHCOM:

```
SET SERVER CPUS(n:m)
RESET SERVER PROCESS $<default queue server process name>
SET SERVER PROCESS $<new queue server name>
```

4. Add the new server, giving it a new server class name by entering the following command from PATHCOM:

```
ADD SERVER MQS-QUEUEnn
```

5. Start the new queue server, so that it can be used, and exit from PATHCOM by entering the following commands:

```
START SERVER MQS-QUEUEnn
EXIT
```

The recommended naming convention for additional server classes is MQS-QUEUE01, MQS-QUEUE02, and so on. However, there is no requirement to use this convention. Provided that the server class name begins with the character string MQS-QUEUE, the server class will be started by **strmqm**.

### Creating queue managers

Once a queue server has been added and started (either explicitly using PATHCOM or implicitly by **strmqm**), objects can be reassigned to the new queue server using **altmqfls**. For more information about reassigning objects, see "Reassigning objects to status servers and queue servers" on page 96.

Before removing a queue server, check that all objects configured against this queue server have been either deleted or reassigned to a different queue server. You must **not** delete the default queue server, otherwise the queue manager will become inoperable.

## Volume structure

Files for MQSeries for Compaq NSK are distributed over several subvolumes. The volume in which these subvolumes reside is selected when you create the queue manager: it is either taken from the default volume value in MQSINI or specified on the -p `DefaultPrefix` parameter of the **crtmqm** command.

There are five subvolumes per queue manager. The contents of the subvolumes are determined by the final character of the subvolume name. For example, for a queue manager called QMGR resident on a volume $DATA, the following subvolumes would be present:

| | |
|---|---|
| **$DATA.QMGR** | FFST subvolume |
| **$DATA.QMGRD** | Queue manager data files subvolume |
| **$DATA.QMGRL** | Queue manager error logs subvolume |
| **$DATA.QMGRM** | Message queue subvolume |
| **$DATA.QMGRS** | Channel synchronization subvolume |

If the queue manager name is more than seven characters, the subvolume names are transformed or shortened. The MQSINI file stanzas `QMVolume` and `QMSubvolume` for the queue manager are used to record the locations and names of these subvolumes.

### Queue manager FFST subvolume

The FFST subvolume contains first failure support files. These files are all prefixed with the letters FD. They indicate serious problems with the MQSeries system, such as resource shortage, internal MQSeries errors, or problems with the Compaq NSK system.

### Queue manager data files subvolume

**AMQRFN***xx* Are the Repository Cache Shared Memory files for each CPU (number *xx*).

**CCHDEFS** Is the client connection channel definition file.

**CCSIDMEM** Is the Read Only shared memory file for data conversion CCSID support.

**CHDEFS** Is the channel definition file. This file contains configuration information for the channels that are defined for a queue manager.

**Lxxxxxxx** Are the namelist files. Lxxxxxxx is derived from the object name; otherwise it is a generated value.

**OAMDB** Is the authorization database.

**OBJCAT** Is the object catalog.

**ABJCAT** Is the alternate key file for OBJCAT that contains an index by object name.

**PATHCTL** Is the PATHWAY control file.

**PRIDB** Is the MQSeries principal (user) database.

**PRIDBA** Is the alternate key file for PRIDB.

**QMINI** Is the queue manager initialization file.

**QMINIMEM** Is the Read Only shared memory file for the queue manager wide static configuration information.

**REPMGR** Is a file used for coordinating the startup of multiple repository server processes.

**RDFPURGE** Is the database used by the queue manager to record logically deleted files that will eventually be removed by **cleanrdf**.

**RUNTIME** Is the file for ECBOSS and EC recovery coordination.

**SHUTDOWN** Is the file that controls **endmqm**.

**STATABLE** Is the channel status table file. This file holds dynamic information associated with channel status. Used to save channel status information over a queue manager shutdown and restart.

**TRACEOPT** The TRACEOPT file contains the current trace settings for a queue manager in the form of an unformatted bit-map record. The control commands **strmqtrc** and **endmqtrc** modify the contents of the file, using the CONTROL 27 mechanism to notify all processes of the update.

**UMQSINI** Is a snapshot of the unstructured MQSINI file at queue manager startup.

## Queue manager error log subvolume

The error log subvolume contains the error and trace logging files. The TR prefix identifies trace files. (You can change the prefix by editing the `TracePrefix` entry in the QMINI file.) Trace files contain diagnostic information, and are created only if tracing is switched on using either the MQM facility or the **strmqtrc** control command.

The error logs have names in the format MQERRLGn, where n is 1, 2, or 3. MQERRLG1 is always the current error log. Its contents are moved to MQERRLG2 when MQERRLG1 is full; MQERRLG2 is moved to MQERRLG3 when MQERRLG1 is next emptied. MQERRLG3 is overwritten if necessary. There are never more than three error logs, so they must be sized correctly to avoid loss of useful error information.

## Queue manager message queue subvolume

The message subvolume contains files associated with the storage of messages on local queues. The file names are in the following format:

**Qxxxxxxx** Are the queue files themselves, that contain persistent messages.

**Txxxxxxx** Are the Touch files. If an object is altered, the Touch files change the object date stamp. Txxxxxxx is derived from the object name; otherwise it is a generated value.

**Oxxxxxxx** Is the queue overflow file.

**xxxxxxx** May be the queue name if it is a unique, short name; otherwise it is a generated value. (See "Object name transformation".)

In addition to the files described above, with specific file name prefix letters, message overflow files are created in the message queue subvolume by default. Message overflow files are created to store large messages and, although they have no specific file name prefix, they can be readily identified because they are unaudited, unstructured files. The location of new message overflow files may be changed using **altmqfls** on a per-queue basis by specifying the --msgofsubvol parameter. (See "dspmqfls (Display MQSeries file attributes)" on page 253 for more information.)

## Queue manager channel synchronization subvolume

The queue manager synchronization subvolume contains internal databases that record the status of units of work (or batches of messages) transmitted or received over the channels that are owned by the queue manager.

Once channels have been used on a queue manager, the subvolume contains the following files:

**SYNCHIDX**
The synchronization index file. Contains an entry for each synchronization file created by the queue manager.

**Sxxxxxxx**
Individual synchronization files. There is one file for each unique combination of local and remote channel that has been used in the queue manager. These files record the identities of the messages that have been transmitted or received within a batch of messages. The information is used in the resynchronization of channels following failure and the resolution of in-doubt channels.

## Object name transformation

Object names are not necessarily valid file system names. Therefore, the object names might need to be transformed. The method used is different from that used for queue-manager names because, although there may be only a few queue-manager names per system, there can be a large number of other objects for each queue manager. Only process definitions, namelists and queues are represented using separate files in the file system; channels and other objects are not affected by these considerations because they are stored as records in databases that hold multiple object definitions.

When a new name is generated by the transformation process there is no relationship with the original object name. You can use the **dspmqfls** command to convert between real and transformed object names: **dspmqfls** displays the names of the main files associated with an MQSeries object.

# Working with queue managers

MQSeries provides control commands for creating, starting, ending, and deleting queue managers. You can also display a queue manager's attributes using the MQSC command DISPLAY QMGR and change them using ALTER QMGR. See "Displaying queue manager attributes" on page 86 and "Altering queue manager attributes" on page 87.

Ensure that the environment variable PMSEARCHLIST specifies the location of your MQSeries executables before you attempt to use the control commands. For more on this environment variable, see “Appendix C. Setting TACL environment variables for MQSeries for Compaq NSK” on page 299.

## Creating a default queue manager

You create a default queue manager using the **crtmqm** command. The **crtmqm** command specified with a -q flag:

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

```
 crtmqm -q -d MY.DEF.XMITQ -u SYSTEM.DEAD.LETTER.QUEUE -n $PMON -o $TRM01 -s $MQSS
-v $MQQS saturn.queue.manager
```

where:

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

**-d MY.DEF.XMIT.Q**
is the name of the default transmission queue.

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

**-n $PMON**
Is the process name of PATHMON for the queue manager.

**-o $TRM01**
Is the home terminal name (must be paused).

**-s $MQSS**
Is the process name of the default status server.

**-v $MQQS**
Is the process name of the default queue server.

**saturn.queue.manager**
Is the name of this queue manager. For **crtmqm**, this name must be the last parameter in the command.

A queue manager with the name and options you specified is created. By creating a queue manager, you also automatically created the following:

- MQSeries default principal for the creator of the queue manager
- Status server for the queue manager
- Queue server for the queue manager
- A configuration file for the queue manager

You can now start the queue manager. For more information, see “Starting a queue manager” on page 60.

You may want to alter some of the attributes of a queue manager. You can do this using the MQM or the control command strmqm. For more information, see Figure 5 on page 65 or “strmqm (Start queue manager)” on page 285.

### Working with queue managers

You should backup the two configuration files that were created when the queue manager was created. If you create another queue manager that causes problems, you can reinstate the backups. For more information, see "Backing up configuration files after creating a queue manager" on page 43.

## Creating MQSeries principals

The **crtmqm** command automatically creates a principal for the user that created the queue manager. This principal (also known as the default principal) is always called mqm for compatibility with other MQSeries implementations.

Once you have created a queue manager you may define principals for other users of MQSeries. This step may be performed at any time (whether or not the queue manager has been started). If no other users are required for the queue manager, this step can be omitted.

To create an MQSeries principal named `MQPRINCIPAL` corresponding to Compaq NSK user `MQM.MQUSER`, enter the command:

```
altmqusr -m saturn.queue.manager -p MQPRINCIPAL -u MQM.MQUSER
```

To display all the principals currently created, enter the command:

```
dspmqusr -m saturn.queue.manager
```

Remember that if you do not create a principal entry for a user, any attempt to access the queue manager by that user (whether the OAM is enabled or not) will result in an authorization error. This change was introduced in MQSeries for Compaq NSK Version 2.2.0.1 and is a part of all later versions.

## Running cleanrdf for an RDF-enabled queue manager

If you are running a queue manager in the RDF environment and have enabled RDF-specific behavior using the MQRDF PARAM, you should run the **cleanrdf** utility periodically, as follows:

- After making any configuration changes (such as creating or deleting objects, or making changes to the QMINI file), **cleanrdf** should be run.
- If your application creates and deletes objects as part of its normal operation, especially if it uses dynamic queues, **cleanrdf** should be run during normal operation at a frequency depending upon the rate of object deletion. NetBatch or other scheduling software should be used.

## Starting a queue manager

Although you have created a queue manager, it cannot process commands or MQI calls until it is started. Start the queue manager by entering 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.

## Restoring the default and system objects

Default and system objects are automatically created when the queue manager is created, however, the objects can be replaced by other object definitions at any time. To restore the default and system objects to a queue manager named saturn.queue.manager, issue the **strmqm** command with the **-c** option:

```
strmqm -c saturn.queue.manager
```

The queue manager is started, the default and system objects that were created when the queue manager was created are restored, then the queue manager is stopped.

## Looking at object files

Each MQSeries queue, queue manager, or process object is represented by a file. Because the names of these objects are not necessarily valid file names, the queue manager converts the object name into a valid file name, where necessary. This process is described in “Object name transformation” on page 58.

## 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
```

By default, this command performs a *controlled* or *quiesced* shutdown of the specified queue manager. This process might take a while to complete—a controlled shutdown waits until *all* connected applications have disconnected and until all running channels have stopped.

“Immediate and preemptive queue manager shutdowns” describes optional flags for the **endmqm** command that specify how the shutdown is to be carried out.

### If you have problems

Problems in shutting down a queue manager are often caused by applications. For example, when applications:

- Do not check MQI return codes properly.
- Do not request a notification of a quiesce.

### Immediate and preemptive queue manager shutdowns

If a shutdown of a queue manager is slow, or the queue manager does not stop, you can terminate the **endmqm** command using BREAK followed by STOP. You can then issue another **endmqm** command, but this time with a flag specifying either an immediate or a preemptive shutdown.

For an *immediate shutdown* any current MQI calls are allowed to complete, but any new calls fail. This type of shutdown does not wait for applications to disconnect from the queue manager. For an immediate shutdown, the command is:

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

If an immediate shutdown does not work, try a *preemptive* shutdown by specifying the -p flag. For example:

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

> **Attention**
>
> 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 this method still does not work, see “Stopping a queue manager manually” on page 307 for an alternative.

For a detailed description of the **endmqm** command and its options, see “endmqm (End queue manager)” on page 263.

## Restarting a queue manager

To restart a queue manager, use the command:

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

## Making an existing queue manager the default

When you create a default queue manager, the name of the default queue manager is inserted in the *DefaultQueueManager* stanza in the MQSeries configuration file (MQSINI). The stanza and its contents are automatically created if they do not exist.

You might need to edit this stanza:

- To make an existing queue manager the default. To perform this task you have to change the queue manager name in this stanza to the name of the new default queue manager. You must perform this step manually using a text editor.
- If you do not have a default queue manager on the node, and you want to make an existing queue manager the default. To perform this task, you must create the *DefaultQueueManager* stanza—with the required name—yourself.
- If you accidentally make another queue manager the default and want to revert to the original default queue manager. To perform this task, edit the *DefaultQueueManager* stanza in the MQSeries configuration file, replacing the name of the unwanted default queue manager with that of the one you do want.

The default queue manager changes to the one specified. All subsequent connection attempts where the application does not specify a queue manager will connect to the new default queue manager.

See “Chapter 13. Configuration files” on page 173 for information about configuration files.

When the stanza contains the required information, stop the queue manager and restart it.

## Deleting a queue manager

To delete a queue manager, first stop it, then use the following command:

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

**Attention**

Use caution if deleting a queue manager as you also delete all the resources associated with it, including all queues and their messages and all object definitions. Also, all files in the queue manager subvolumes might be purged (even if they were not created by MQSeries).

For a description of the **dltmqm** command and its options, see “dltmqm (Delete queue manager)” on page 246. You should ensure that only trusted administrators have the authority to use this command.

If the usual methods for deleting a queue manager do not work, see “Removing queue managers manually” on page 307 for an alternative.

## Using the Message Queue Management (MQM) facility

The Message Queue Management (MQM) facility of MQSeries for Compaq NSK, V5.1 runs as a PATHWAY SCOBOL requester under the Terminal Control Process (TCP). It uses an MQM server class server, which invokes the C language API.

There is a separate instance of the MQM for each queue manager configured on a system, because each queue manager is controlled under its own PATHWAY configuration. Consequently, MQM is limited to the management of the queue manager to which it belongs.

**Note:** By default, a maximum of 10 users may use the MQM facility concurrently. To change this limit to 20, for example, enter from the PATHWAY of the queue manager:

```
ALTER TCP MQS-TCP-01, MAXTERMS 20
```

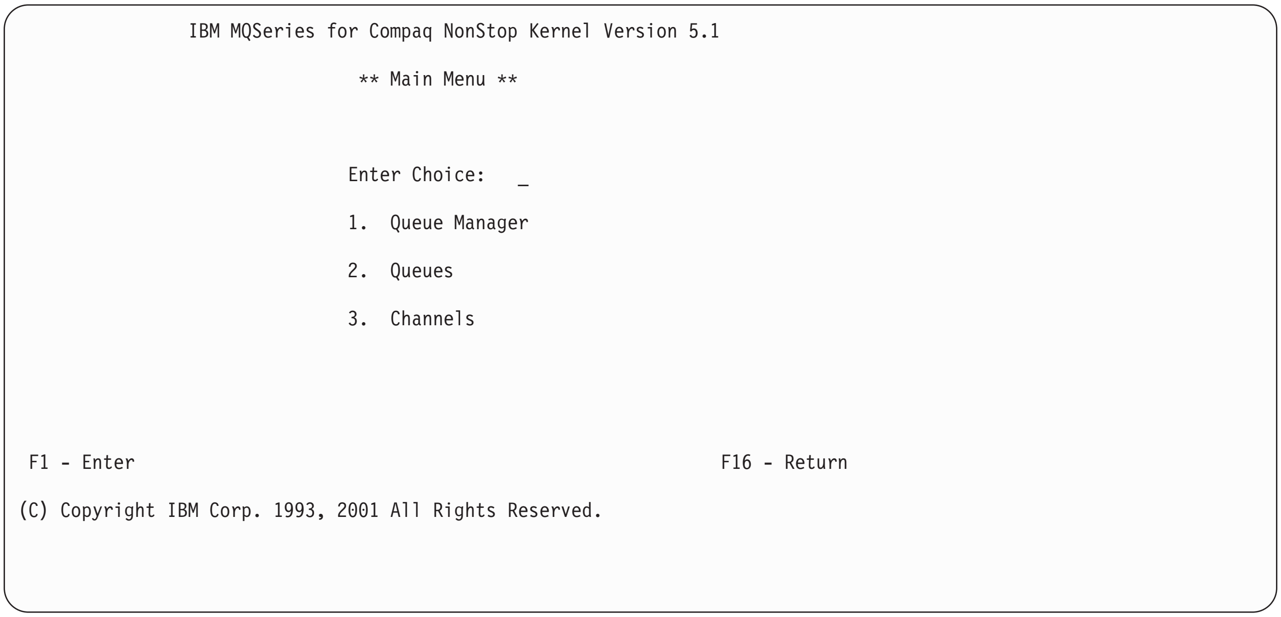
For more information, see “Chapter 3. Using the MQSeries command sets” on page 27.

**Note:**

To invoke MQM, enter `run mqmc` from the queue manager’s PATHCOM prompt.

### Message Queue Management (MQM)

The MQM Main Menu is as follows:



```
               IBM MQSeries for Compaq NonStop Kernel Version 5.1

                               ** Main Menu **


                              Enter Choice:   _

                              1.  Queue Manager

                              2.  Queues

                              3.  Channels



 F1 - Enter                                                      F16 - Return

(C) Copyright IBM Corp. 1993, 2001 All Rights Reserved.
```

*Figure 4. The MQSeries for Compaq NSK MQM Main Menu*

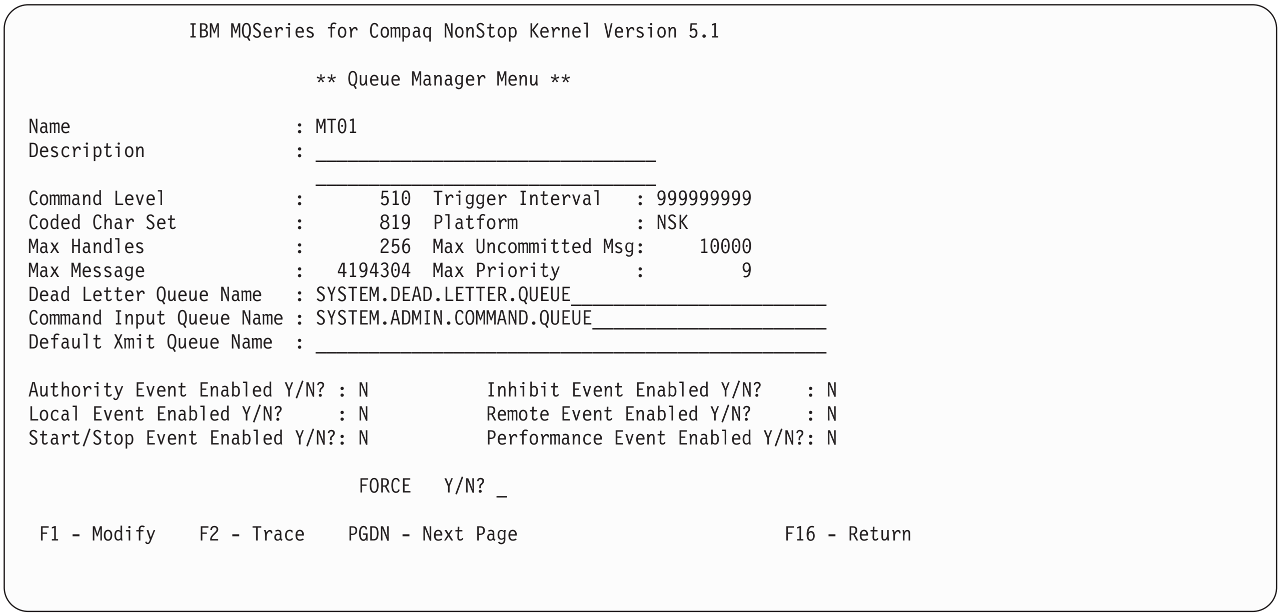
You can select the following submenus from the MQM Main Menu:

1. Queue Manager
2. Queues
3. Channels

These submenus are described in the remainder of this chapter. You can return to the MQM Main Menu at any time by pressing Alt+F6. You can return to the previous screen by pressing the Return key (F16). When selected from the MQM Main Menu, F16 exits from the MQM facility.

## Using the Queue Manager Menu

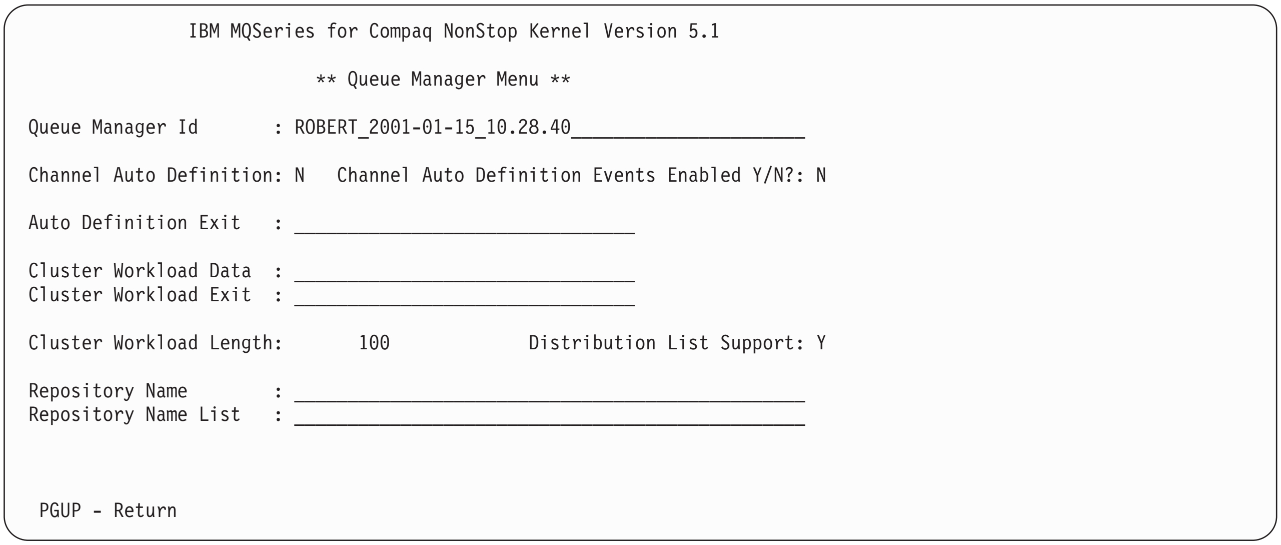
To select the `Queue Manager` option, type `1` in the Enter Choice field on the MQM Main Menu, then press the Enter key (F1). The Queue Manager Menu panel is displayed:



```
                 IBM MQSeries for Compaq NonStop Kernel Version 5.1

                             ** Queue Manager Menu **

Name                     : MT01
Description              : ________________________________
                           ________________________________
Command Level            :       510  Trigger Interval   : 999999999
Coded Char Set           :       819  Platform           : NSK
Max Handles              :       256  Max Uncommitted Msg:     10000
Max Message              :   4194304  Max Priority       :         9
Dead Letter Queue Name   : SYSTEM.DEAD.LETTER.QUEUE________________________
Command Input Queue Name : SYSTEM.ADMIN.COMMAND.QUEUE______________________
Default Xmit Queue Name  : ________________________________________________

Authority Event Enabled Y/N? : N           Inhibit Event Enabled Y/N?    : N
Local Event Enabled Y/N?     : N           Remote Event Enabled Y/N?     : N
Start/Stop Event Enabled Y/N?: N           Performance Event Enabled Y/N?: N

                               FORCE   Y/N? _

 F1 - Modify    F2 - Trace    PGDN - Next Page                         F16 - Return
```

Figure 5. The Queue Manager Menu panel

Press PGDN key to display the second panel of information.



```
                 IBM MQSeries for Compaq NonStop Kernel Version 5.1

                             ** Queue Manager Menu **

Queue Manager Id       : ROBERT_2001-01-15_10.28.40______________________

Channel Auto Definition: N   Channel Auto Definition Events Enabled Y/N?: N

Auto Definition Exit   : ________________________________

Cluster Workload Data  : ________________________________
Cluster Workload Exit  : ________________________________

Cluster Workload Length:       100            Distribution List Support: Y

Repository Name        : ________________________________________________
Repository Name List   : ________________________________________________



 PGUP - Return
```

Figure 6. The Queue Manager Menu panel 2

You can use the Queue Manager Menu panel to:

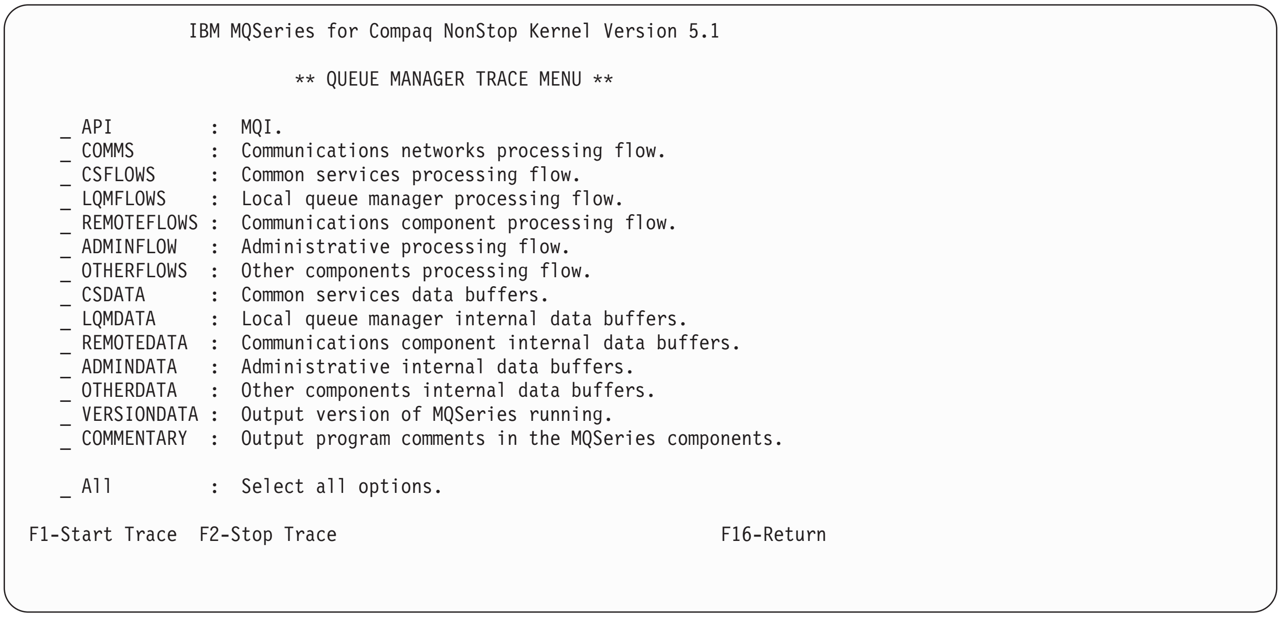
- Alter some attributes of the queue manager
- Control tracing of MQSeries objects

## Altering queue manager attributes

Overtype those values you want to alter on the Queue Manager Menu panel, and press the Modify key (F1). You are prevented from overtyping those values that cannot be modified.

## Tracing MQSeries objects

Press the Trace key (F2) to display the QUEUE MANAGER TRACE MENU:



```
              IBM MQSeries for Compaq NonStop Kernel Version 5.1

                        ** QUEUE MANAGER TRACE MENU **

   _ API         :  MQI.
   _ COMMS       :  Communications networks processing flow.
   _ CSFLOWS     :  Common services processing flow.
   _ LQMFLOWS    :  Local queue manager processing flow.
   _ REMOTEFLOWS :  Communications component processing flow.
   _ ADMINFLOW   :  Administrative processing flow.
   _ OTHERFLOWS  :  Other components processing flow.
   _ CSDATA      :  Common services data buffers.
   _ LQMDATA     :  Local queue manager internal data buffers.
   _ REMOTEDATA  :  Communications component internal data buffers.
   _ ADMINDATA   :  Administrative internal data buffers.
   _ OTHERDATA   :  Other components internal data buffers.
   _ VERSIONDATA :  Output version of MQSeries running.
   _ COMMENTARY  :  Output program comments in the MQSeries components.

   _ All         :  Select all options.

 F1-Start Trace  F2-Stop Trace                                    F16-Return
```

*Figure 7. The QUEUE MANAGER TRACE MENU*

The following trace options are available:

**API**
Output data for trace points associated with the MQI and major queue manager components.

**COMMS**
Output data for trace points associated with data flowing over communications networks.

**CSFLOWS**
Output data for trace points associated with processing flow in common services.

**LQMFLOWS**
Output data for trace points associated with processing flow in the local queue manager.

**REMOTEFLOWS**
Output data for trace points associated with processing flow in the communications component.

**ADMINFLOW**
Output data for trace points associated with administrative internal data buffers.

**OTHERFLOWS**
Output data for trace points associated with other components' processing flow.

**CSDATA**
Output data for trace points associated with internal data buffers in common services.

**LQMDATA**
Output data for trace points associated with internal data buffers in the local queue manager.

**REMOTEDATA**
Output data for trace points associated with internal data buffers in the communications component.

**ADMINDATA**
Output data for trace points associated with internal data buffers in the communications component.

**OTHERDATA** Output data for trace points associated with other components' internal data buffers.

**VERSIONDATA**
Output data for trace points associated with the version of MQSeries that is running.

**COMMENTARY**
Output data for trace points associated with comments in the MQSeries components.

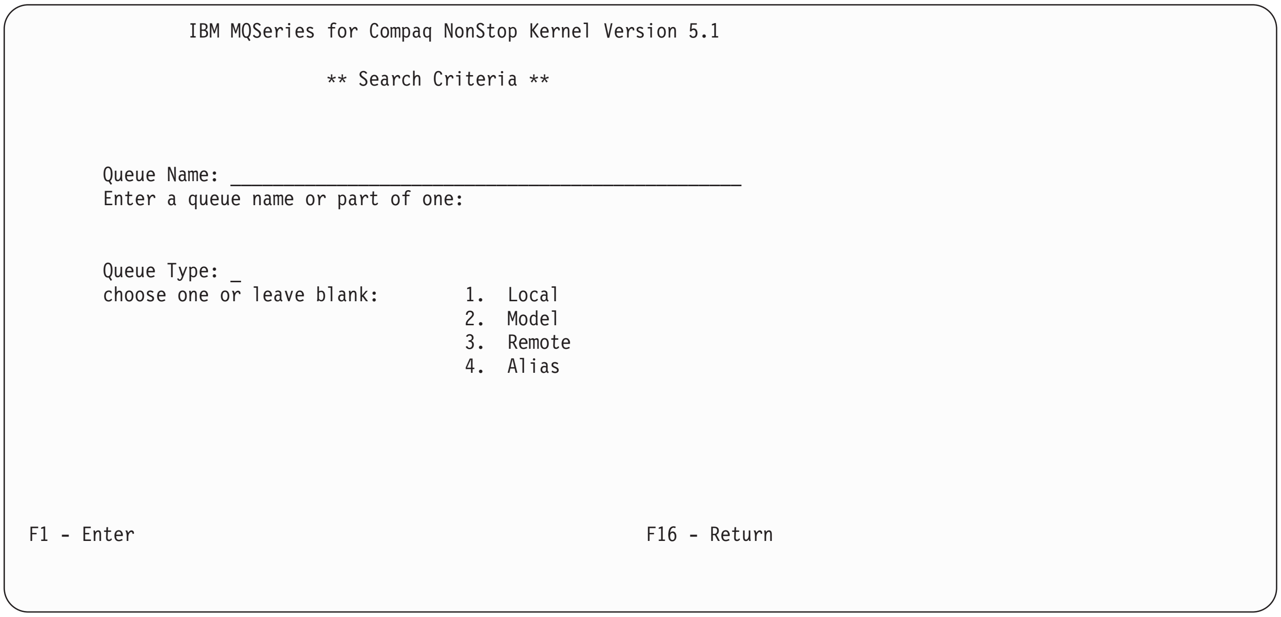
**ALL** Trace points are enabled and a full trace is generated.

Type any character against the names of the components for which you want to start (or stop) tracing.

To start tracing of the selected components, press the Start Trace key (F1). To stop tracing of the selected components, press the Stop Trace key (F2).

## Using the Queues menu

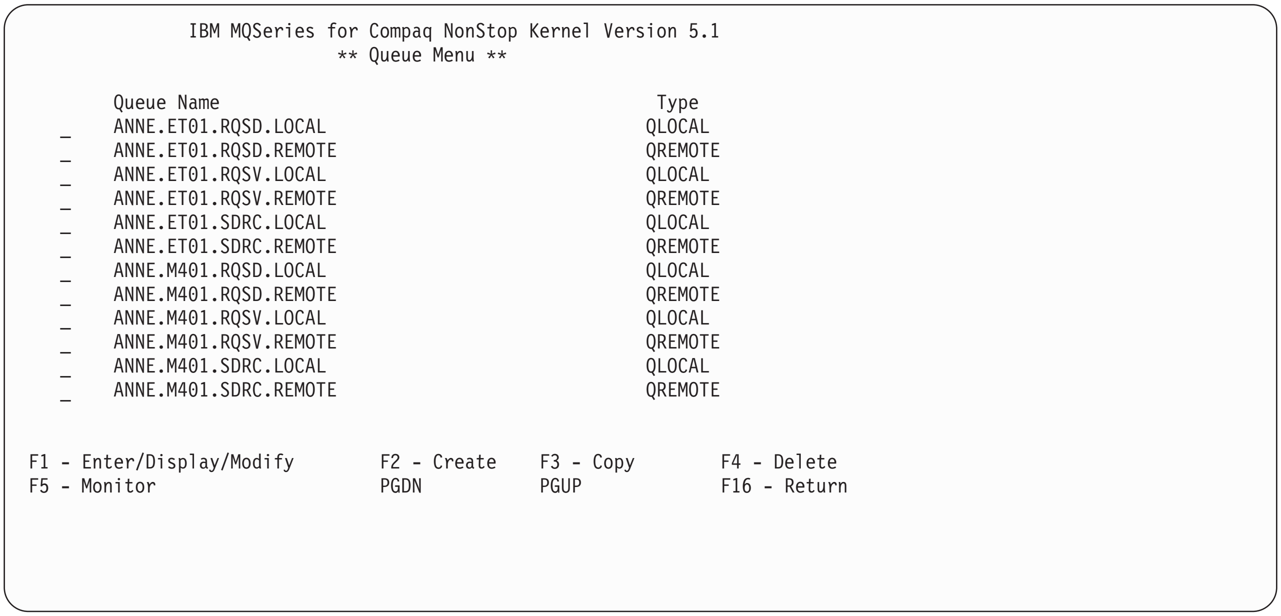
To select the `Queues` option, type `2` in the Enter Choice field on the MQM Main Menu, then press the Enter key (F1). The Search Criteria panel is displayed:



```
             IBM MQSeries for Compaq NonStop Kernel Version 5.1

                          ** Search Criteria **



        Queue Name: ________________________________________________
        Enter a queue name or part of one:


        Queue Type: _
        choose one or leave blank:       1.  Local
                                         2.  Model
                                         3.  Remote
                                         4.  Alias






 F1 - Enter                                         F16 - Return
```

*Figure 8. The Search Criteria panel (queue)*

In the Queue Name field of the Search Criteria panel, type a partial or complete queue name. You may also provide a Queue Type identifier if you wish to limit your search to queues of one type. Press the Enter key (F1). The Queue menu, which you use to display, modify, create, copy, delete, and monitor MQSeries queues, is displayed.



```
               IBM MQSeries for Compaq NonStop Kernel Version 5.1
                             ** Queue Menu **

        Queue Name                                         Type
   _    ANNE.ET01.RQSD.LOCAL                              QLOCAL
   _    ANNE.ET01.RQSD.REMOTE                             QREMOTE
   _    ANNE.ET01.RQSV.LOCAL                              QLOCAL
   _    ANNE.ET01.RQSV.REMOTE                             QREMOTE
   _    ANNE.ET01.SDRC.LOCAL                              QLOCAL
   _    ANNE.ET01.SDRC.REMOTE                             QREMOTE
   _    ANNE.M401.RQSD.LOCAL                              QLOCAL
   _    ANNE.M401.RQSD.REMOTE                             QREMOTE
   _    ANNE.M401.RQSV.LOCAL                              QLOCAL
   _    ANNE.M401.RQSV.REMOTE                             QREMOTE
   _    ANNE.M401.SDRC.LOCAL                              QLOCAL
   _    ANNE.M401.SDRC.REMOTE                             QREMOTE


F1 - Enter/Display/Modify        F2 - Create    F3 - Copy        F4 - Delete
F5 - Monitor                     PGDN           PGUP             F16 - Return
```

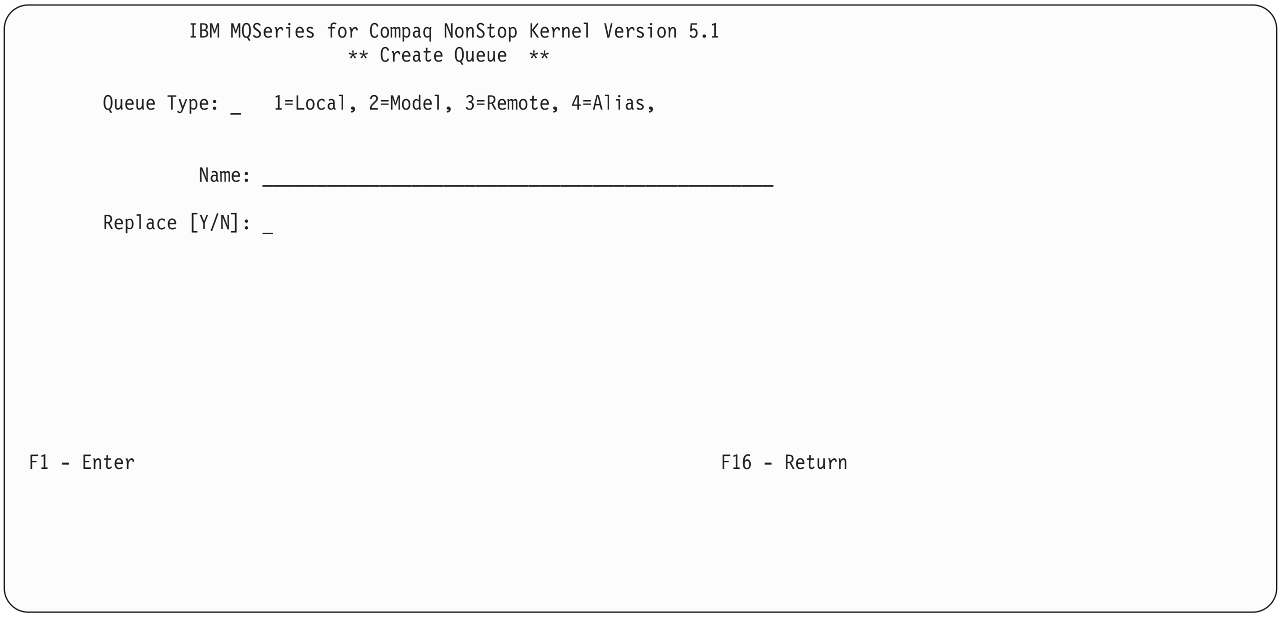
*Figure 9. The Queue Menu*

**Note:** You can create, modify, and delete queues only on the queue manager associated with the MQM requester that you are using.

Use the PGUP and PGDN keys to scroll the list of queues.

## Creating a queue

From the Queue Menu, press the Create key (F2) to display the Create Queue panel:



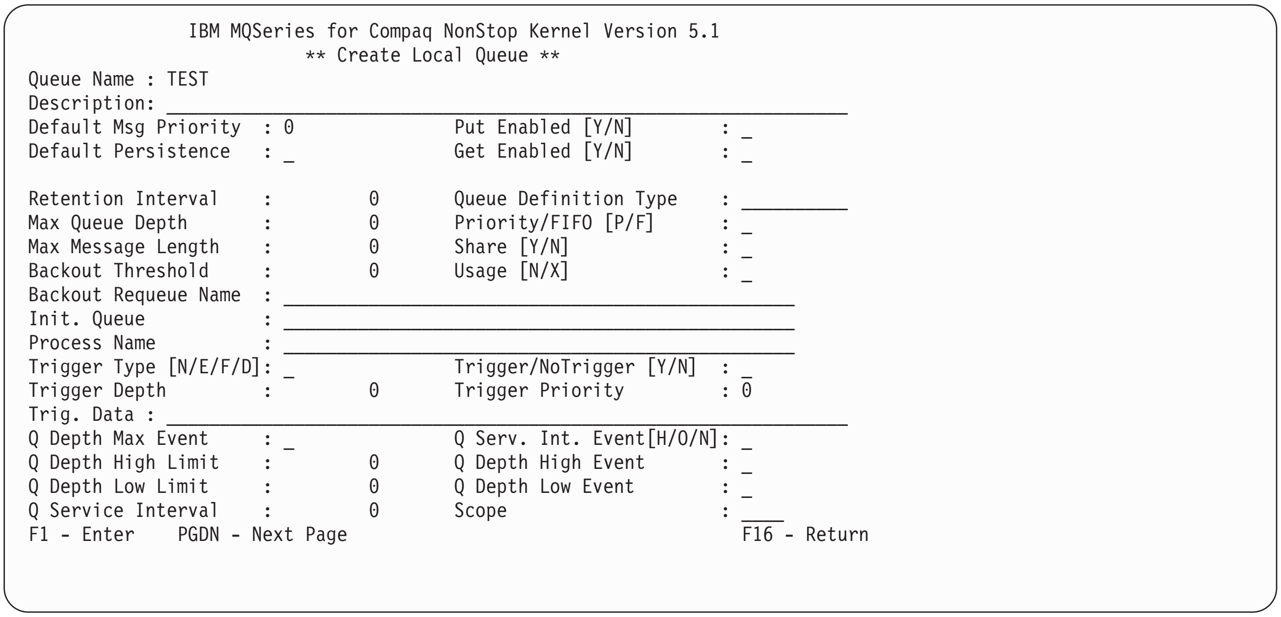
```
               IBM MQSeries for Compaq NonStop Kernel Version 5.1
                              ** Create Queue  **

       Queue Type: _   1=Local, 2=Model, 3=Remote, 4=Alias,


                Name: ________________________________________________

       Replace [Y/N]: _







F1 - Enter                                                       F16 - Return
```

*Figure 10. The Create Queue panel*

To create a new queue definition:

1. Type 1 (for a local queue), 2 (for a model queue), 3 (for a remote queue), or 4 (for an alias queue) in the Queue Type field.
2. Type the queue manager name in the Name field.
3. If the queue is to replace an existing queue of the same name and type, type Y in the Replace field.
4. Press the Enter key (F1).

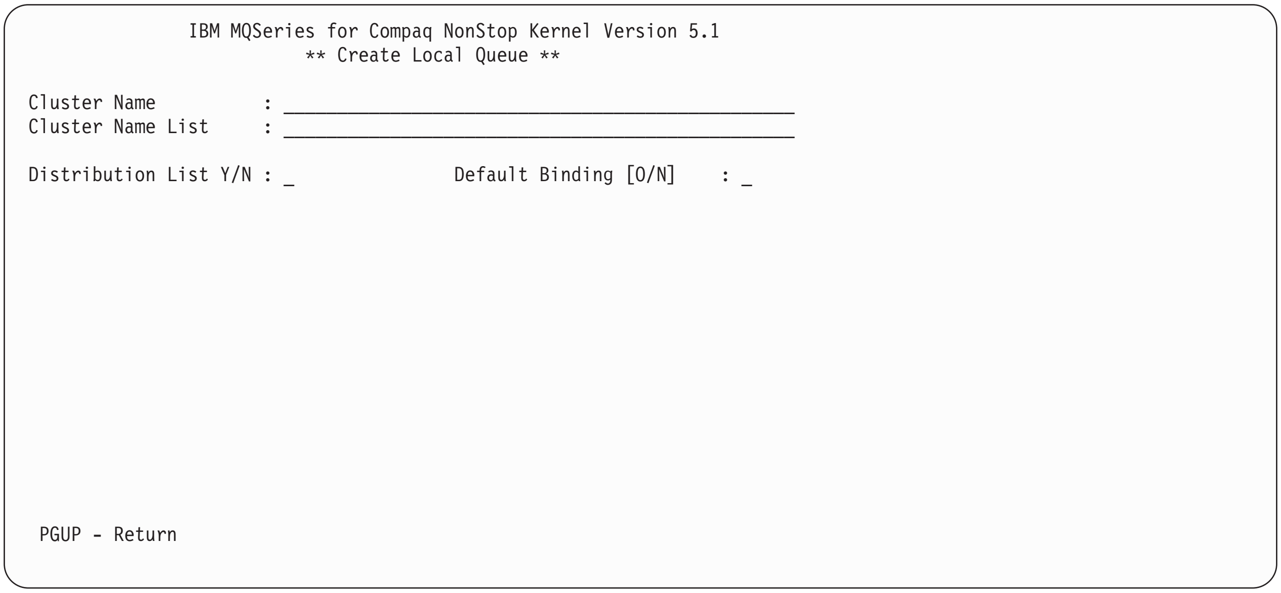
If you create a local queue, the Create Local Queue panel is displayed:



```
               IBM MQSeries for Compaq NonStop Kernel Version 5.1
                          ** Create Local Queue **
Queue Name : TEST
Description: ________________________________________________________________
Default Msg Priority  : 0               Put Enabled [Y/N]        : _
Default Persistence   : _               Get Enabled [Y/N]        : _

Retention Interval    :         0      Queue Definition Type    : __________
Max Queue Depth       :         0      Priority/FIFO [P/F]      : _
Max Message Length    :         0      Share [Y/N]              : _
Backout Threshold     :         0      Usage [N/X]              : _
Backout Requeue Name  : ________________________________________________
Init. Queue           : ________________________________________________
Process Name          : ________________________________________________
Trigger Type [N/E/F/D]: _               Trigger/NoTrigger [Y/N]  : _
Trigger Depth         :         0      Trigger Priority         : 0
Trig. Data : ________________________________________________________________
Q Depth Max Event     : _               Q Serv. Int. Event[H/O/N]: _
Q Depth High Limit    :         0      Q Depth High Event       : _
Q Depth Low Limit     :         0      Q Depth Low Event        : _
Q Service Interval    :         0      Scope                    : ____
F1 - Enter    PGDN - Next Page                                     F16 - Return
```

*Figure 11. The Create Local Queue panel*

Press the PGDN key to display the second panel of information.

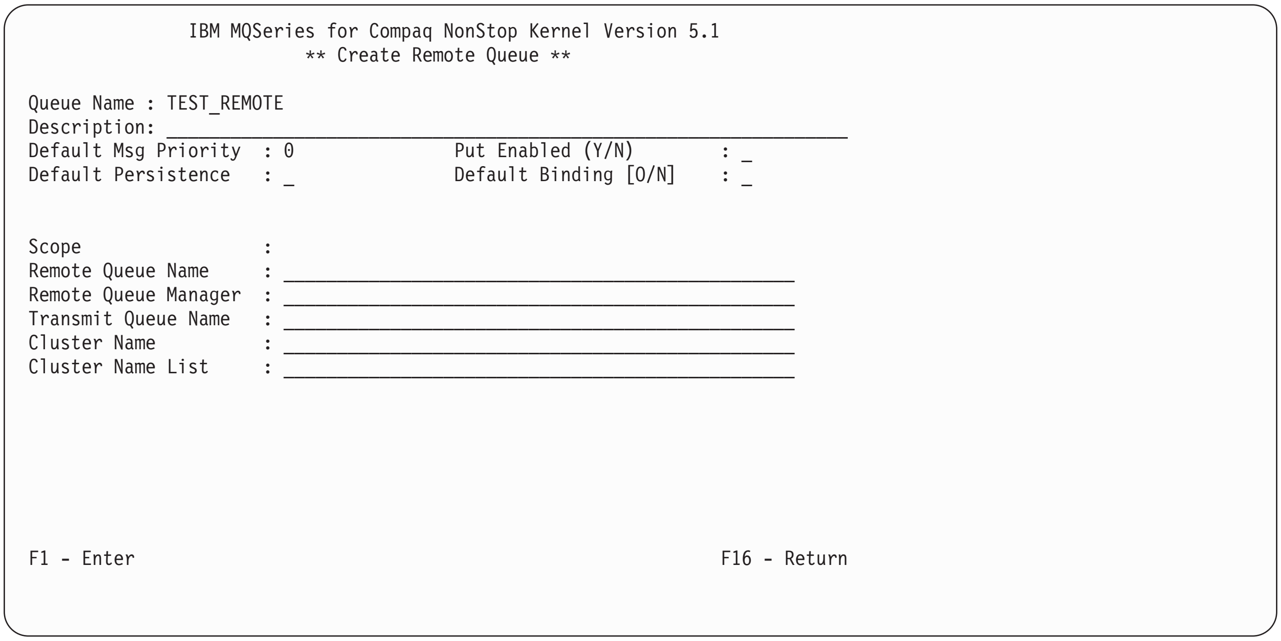


```
               IBM MQSeries for Compaq NonStop Kernel Version 5.1
                          ** Create Local Queue **

Cluster Name          : ________________________________________________
Cluster Name List     : ________________________________________________

Distribution List Y/N : _               Default Binding [O/N]    : _















 PGUP - Return
```

*Figure 12. The Create Local Queue panel 2*

Complete the panel, and press the Enter key (F1).

If you create a remote queue, the Create Remote Queue panel is displayed:



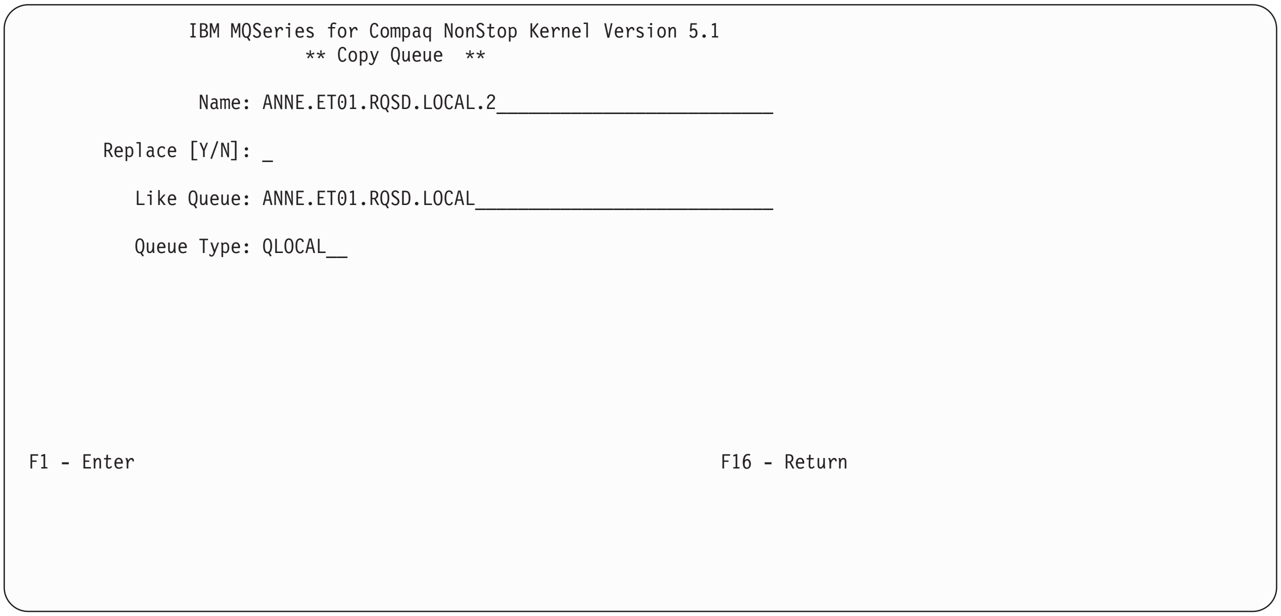
```
               IBM MQSeries for Compaq NonStop Kernel Version 5.1
                          ** Create Remote Queue **

Queue Name : TEST_REMOTE
Description: ________________________________________________________________
Default Msg Priority  : 0               Put Enabled (Y/N)        : _
Default Persistence   : _               Default Binding [O/N]    : _


Scope                 :
Remote Queue Name     : ________________________________________________
Remote Queue Manager  : ________________________________________________
Transmit Queue Name   : ________________________________________________
Cluster Name          : ________________________________________________
Cluster Name List     : ________________________________________________







F1 - Enter                                                    F16 - Return
```

*Figure 13. The Create Remote Queue panel*

Complete the panel, and press the Enter key (F1).

## Copying a queue

From the Queue Menu, press the Copy key (F3) to define a new queue by copying an existing definition. The Copy Queue panel is displayed:



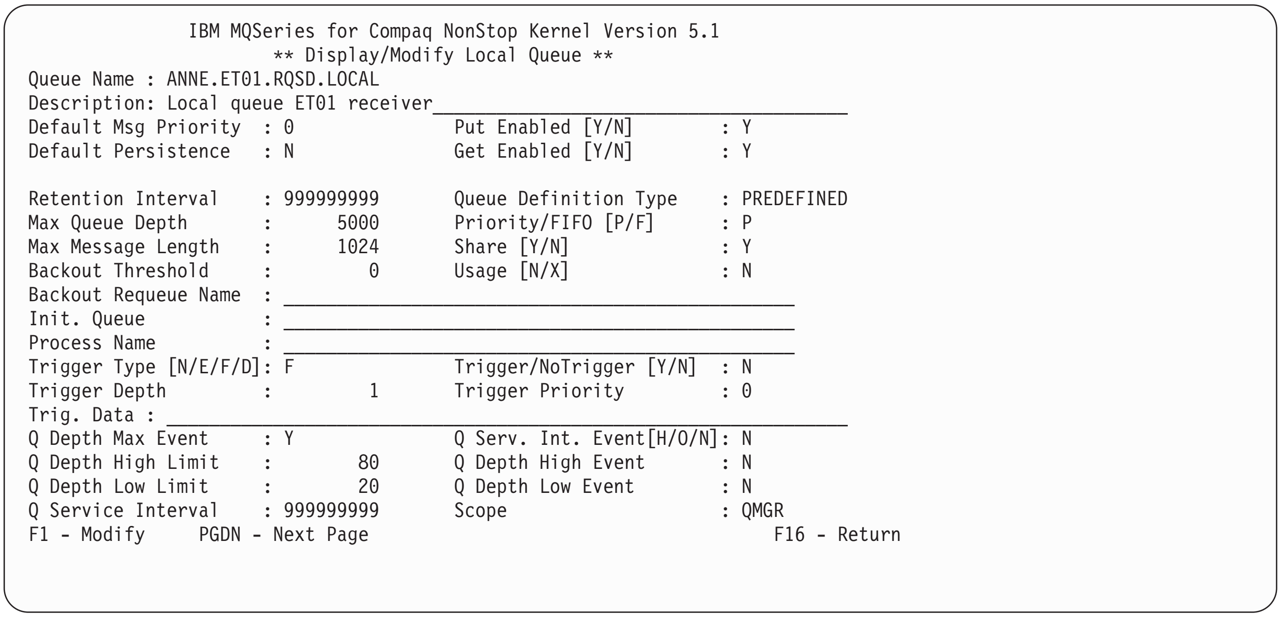
```
               IBM MQSeries for Compaq NonStop Kernel Version 5.1
                          ** Copy Queue  **

                Name: ANNE.ET01.RQSD.LOCAL.2__________________________

        Replace [Y/N]: _

           Like Queue: ANNE.ET01.RQSD.LOCAL____________________________

           Queue Type: QLOCAL__






F1 - Enter                                                      F16 - Return
```

*Figure 14. The Copy Queue panel*

Type the name of the new queue definition in the Name field; type Y in the Replace field if the new queue is to replace an existing queue of the same name and type; type the name of the definition you are copying in the Like Queue field; type the queue type in the Queue Type field. Press the Enter key (F1).

## Modifying a queue

From the Queue Menu, press the Modify key (F1) to display the Display/Modify Local Queue panel:



```
               IBM MQSeries for Compaq NonStop Kernel Version 5.1
                       ** Display/Modify Local Queue **
Queue Name : ANNE.ET01.RQSD.LOCAL
Description: Local queue ET01 receiver_______________________________________
Default Msg Priority  : 0               Put Enabled [Y/N]        : Y
Default Persistence   : N               Get Enabled [Y/N]        : Y

Retention Interval    : 999999999       Queue Definition Type    : PREDEFINED
Max Queue Depth       :      5000       Priority/FIFO [P/F]      : P
Max Message Length    :      1024       Share [Y/N]              : Y
Backout Threshold     :         0       Usage [N/X]              : N
Backout Requeue Name  : ________________________________________________
Init. Queue           : ________________________________________________
Process Name          : ________________________________________________
Trigger Type [N/E/F/D]: F               Trigger/NoTrigger [Y/N]  : N
Trigger Depth         :         1       Trigger Priority         : 0
Trig. Data : _________________________________________________________________
Q Depth Max Event     : Y               Q Serv. Int. Event[H/O/N]: N
Q Depth High Limit    :        80       Q Depth High Event       : N
Q Depth Low Limit     :        20       Q Depth Low Event        : N
Q Service Interval    : 999999999       Scope                    : QMGR
F1 - Modify     PGDN - Next Page                                     F16 - Return
```

*Figure 15. The Display/Modify Local Queue panel*

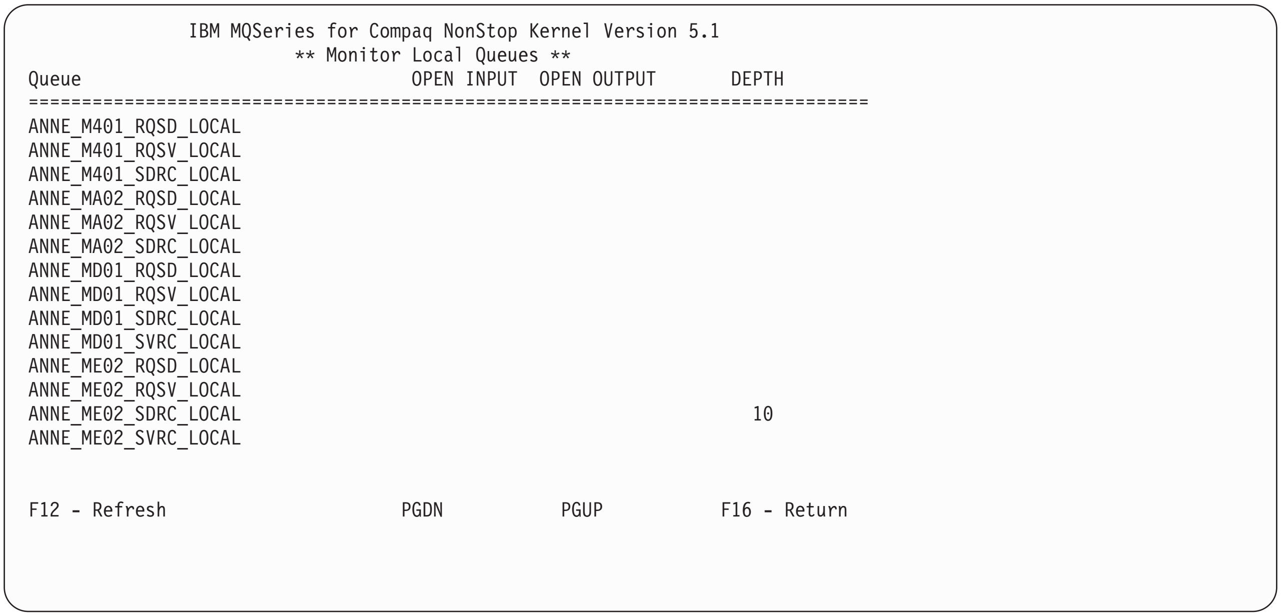
Overtype those values you want to modify, and press the Modify key (F1). You are prevented from overtyping those values that cannot be modified.

### Deleting a queue

On the Queue Menu, enter any character against the name of the queue that you want to delete. Press the Delete key (F4), then press F4 again to confirm deletion.

### Monitoring a queue

Press the Monitor key (F5) from the Queue Menu to display the Monitor Local Queues panel:



```
               IBM MQSeries for Compaq NonStop Kernel Version 5.1
                         ** Monitor Local Queues **
Queue                               OPEN INPUT  OPEN OUTPUT       DEPTH
===============================================================================
ANNE_M401_RQSD_LOCAL
ANNE_M401_RQSV_LOCAL
ANNE_M401_SDRC_LOCAL
ANNE_MA02_RQSD_LOCAL
ANNE_MA02_RQSV_LOCAL
ANNE_MA02_SDRC_LOCAL
ANNE_MD01_RQSD_LOCAL
ANNE_MD01_RQSV_LOCAL
ANNE_MD01_SDRC_LOCAL
ANNE_MD01_SVRC_LOCAL
ANNE_ME02_RQSD_LOCAL
ANNE_ME02_RQSV_LOCAL
ANNE_ME02_SDRC_LOCAL                                                    10
ANNE_ME02_SVRC_LOCAL


F12 - Refresh                      PGDN           PGUP           F16 - Return
```

*Figure 16. The Monitor Local Queues panel*

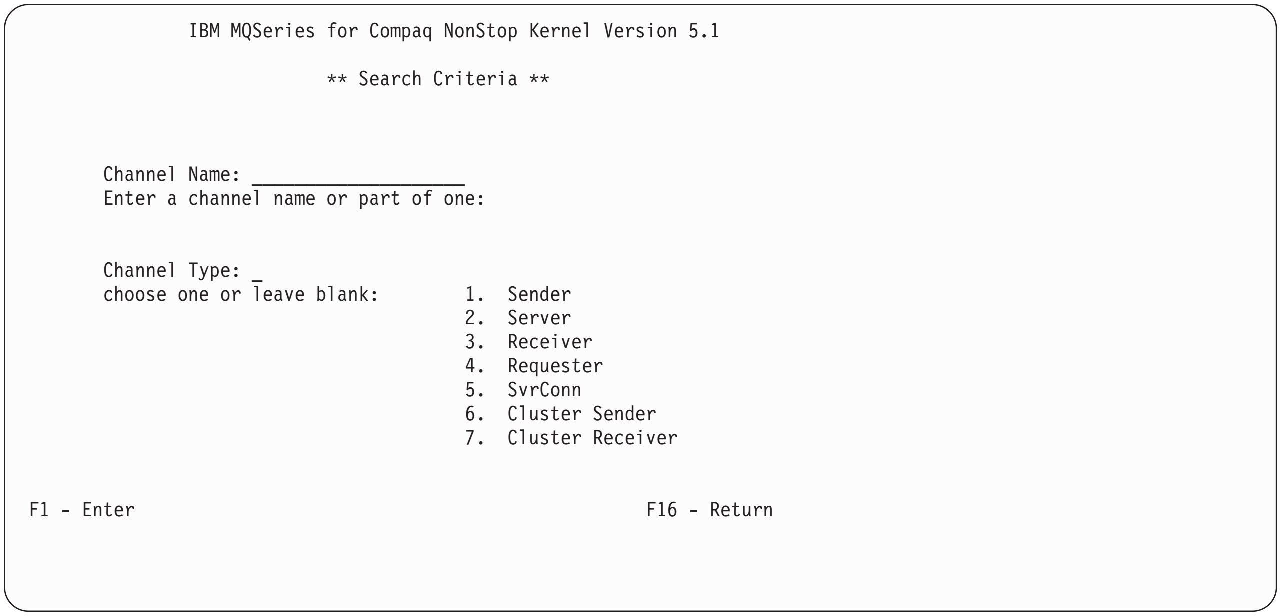
In this example, the queues are open neither for input nor for output. One queue, ANNE_ME02_SDRC_LOCAL, contains 10 messages.

The MQMQMREFRESHINT PATHWAY parameter for MQS-MQMSVR00 determines the frequency with which monitor screens for channels and queues are refreshed. The default frequency is every 30 seconds. To change the frequency to every 10 seconds, for example, enter from the PATHWAY for your queue manager:

```
ALTER SERVER MQS-MQMSVR00, PARAM MQMQMREFRESHINT 10
```

## Using the Channels menu

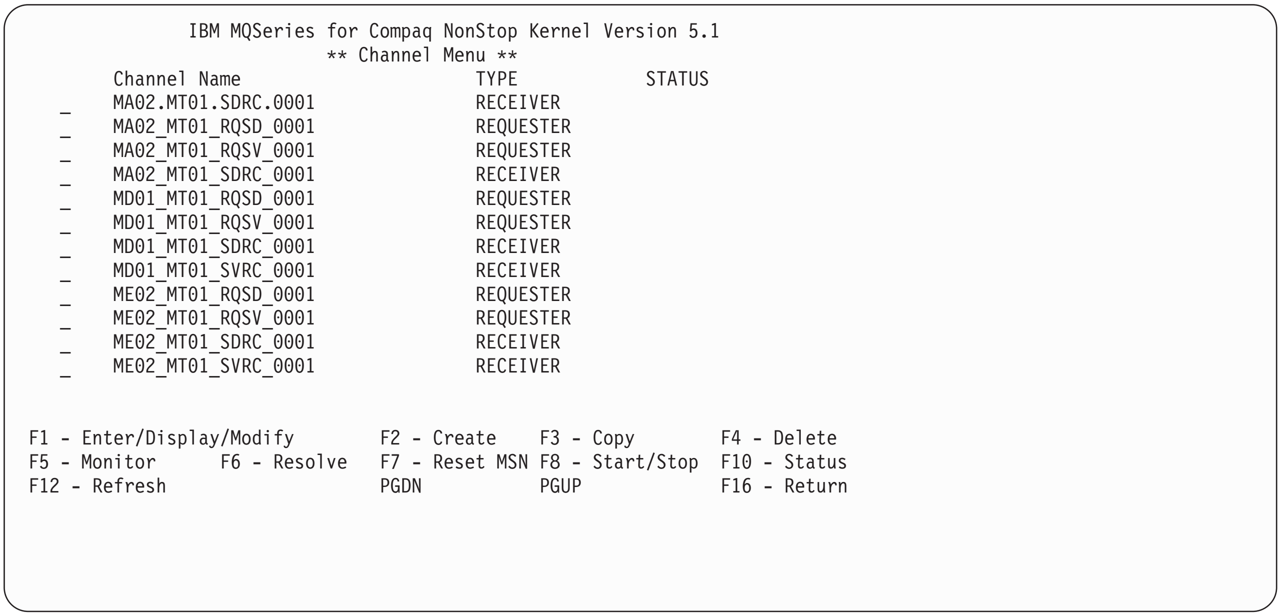
To select the `Channels` option, type 3 in the Enter Choice field on the MQM Main Menu, then press the Enter key (F1). The channel Search Criteria panel is displayed:



```
               IBM MQSeries for Compaq NonStop Kernel Version 5.1

                            ** Search Criteria **



       Channel Name: ____________________
       Enter a channel name or part of one:


       Channel Type: _
       choose one or leave blank:        1.  Sender
                                         2.  Server
                                         3.  Receiver
                                         4.  Requester
                                         5.  SvrConn
                                         6.  Cluster Sender
                                         7.  Cluster Receiver


F1 - Enter                                                F16 - Return
```

Figure 17. The Search Criteria panel (channel)

In the Channel Name field, type a partial or complete channel name. In the Channel Type field, you may enter a number between 1 and 5 to identify the type of channel you are interested in. Press the Enter key (F1) to display the Channel Menu:



```
               IBM MQSeries for Compaq NonStop Kernel Version 5.1
                            ** Channel Menu **
        Channel Name                      TYPE            STATUS
   _    MA02.MT01.SDRC.0001               RECEIVER
   _    MA02_MT01_RQSD_0001               REQUESTER
   _    MA02_MT01_RQSV_0001               REQUESTER
   _    MA02_MT01_SDRC_0001               RECEIVER
   _    MD01_MT01_RQSD_0001               REQUESTER
   _    MD01_MT01_RQSV_0001               REQUESTER
   _    MD01_MT01_SDRC_0001               RECEIVER
   _    MD01_MT01_SVRC_0001               RECEIVER
   _    ME02_MT01_RQSD_0001               REQUESTER
   _    ME02_MT01_RQSV_0001               REQUESTER
   _    ME02_MT01_SDRC_0001               RECEIVER
   _    ME02_MT01_SVRC_0001               RECEIVER


F1 - Enter/Display/Modify        F2 - Create    F3 - Copy        F4 - Delete
F5 - Monitor      F6 - Resolve   F7 - Reset MSN F8 - Start/Stop  F10 - Status
F12 - Refresh                    PGDN           PGUP             F16 - Return
```

Figure 18. The Channel Menu

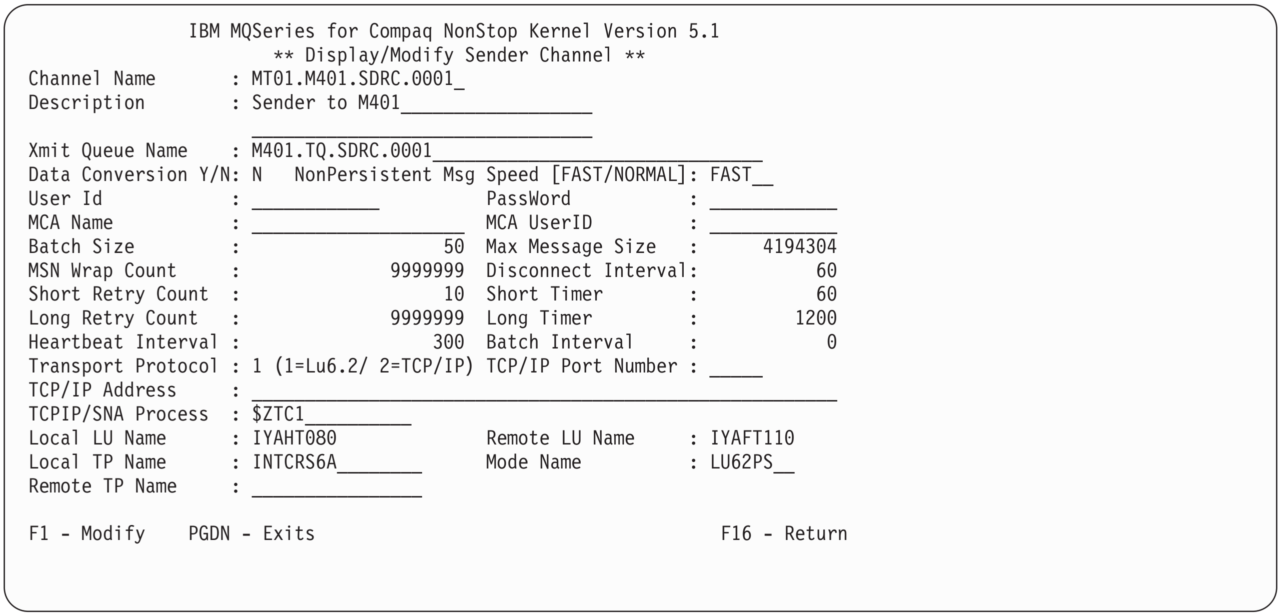
The Channel Menu displays a list of channels that match your search criteria. From the Channel Menu you can:

- Display and modify channel status.
- Create a new channel definition.

- Copy a channel definition.
- Delete a channel definition.
- Monitor channel status.
- Resolve a channel.
- Reset a message sequence number (MSN).
- Start or stop a channel.

## Modifying a channel

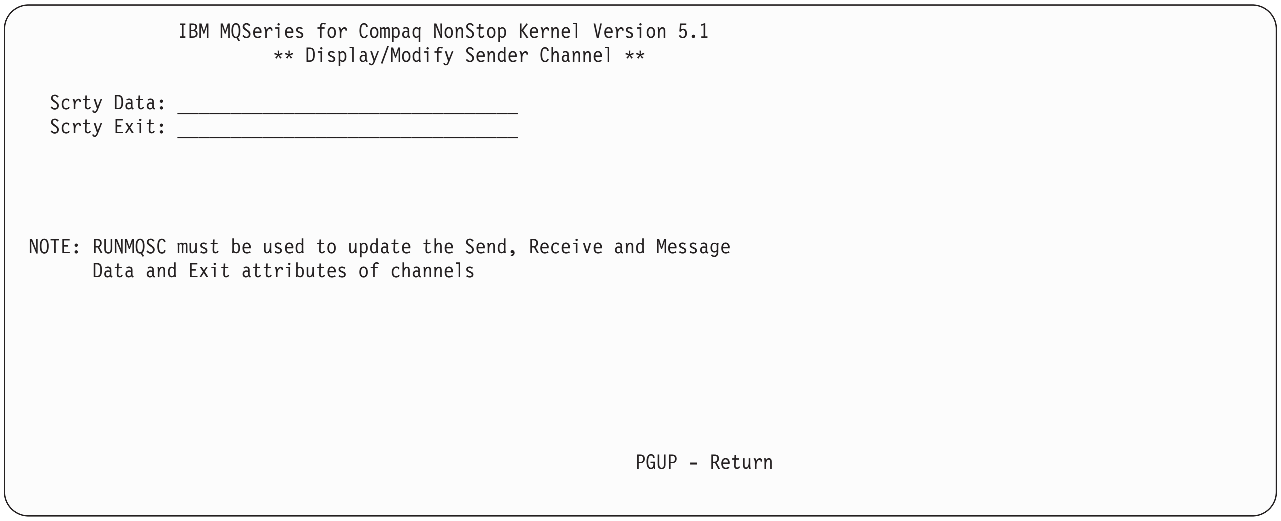
On the Channel Menu, type any character against the channel you want to modify, and press the Enter/Display/Modify key (F1). The appropriate panel is displayed. For example, if you select a sender channel, the Display/Modify Sender Channel panel is displayed:



```
               IBM MQSeries for Compaq NonStop Kernel Version 5.1
                       ** Display/Modify Sender Channel **
Channel Name       : MT01.M401.SDRC.0001_
Description        : Sender to M401__________________
                     ________________________________
Xmit Queue Name    : M401.TQ.SDRC.0001_______________________________
Data Conversion Y/N: N   NonPersistent Msg Speed [FAST/NORMAL]: FAST__
User Id            : ____________          PassWord           : ____________
MCA Name           : ____________________  MCA UserID         : ____________
Batch Size         :                   50  Max Message Size   :      4194304
MSN Wrap Count     :              9999999  Disconnect Interval:           60
Short Retry Count  :                   10  Short Timer        :           60
Long Retry Count   :              9999999  Long Timer         :         1200
Heartbeat Interval :                  300  Batch Interval     :            0
Transport Protocol : 1 (1=Lu6.2/ 2=TCP/IP) TCP/IP Port Number : _____
TCP/IP Address     : _______________________________________________________
TCPIP/SNA Process  : $ZTC1__________
Local LU Name      : IYAHT080              Remote LU Name     : IYAFT110
Local TP Name      : INTCRS6A________      Mode Name          : LU62PS__
Remote TP Name     : ________________

F1 - Modify    PGDN - Exits                                      F16 - Return
```

*Figure 19. The Display/Modify Sender Channel panel (1)*

Press the PGDN key to display the second panel of information:



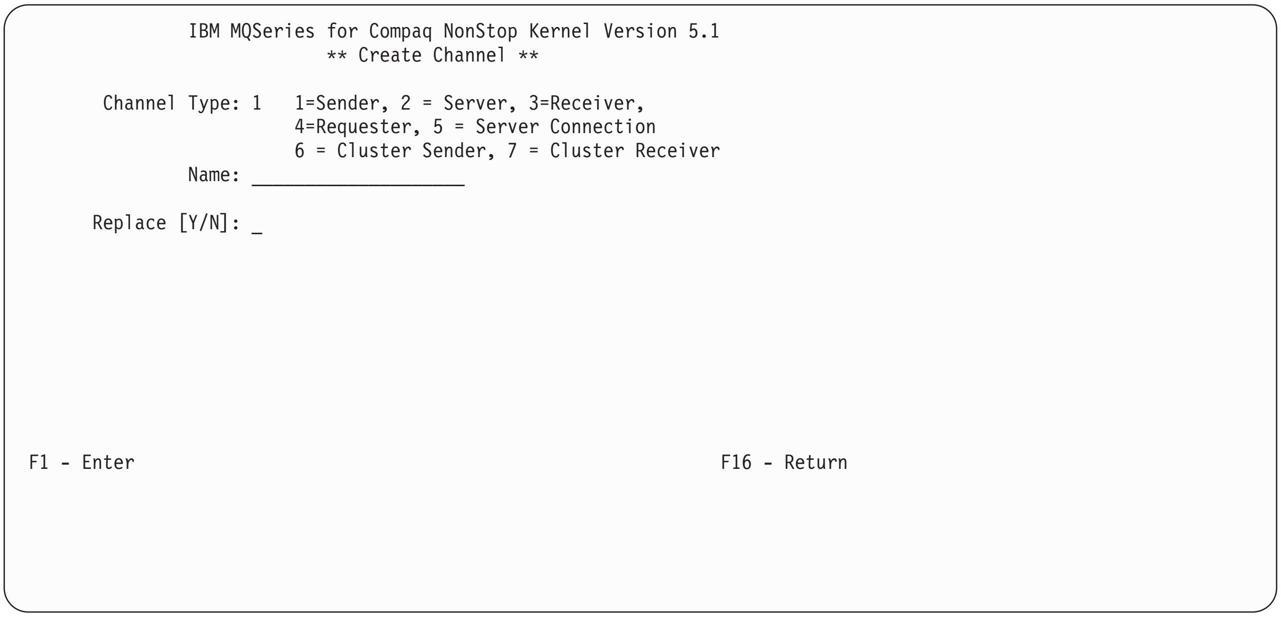
```
              IBM MQSeries for Compaq NonStop Kernel Version 5.1
                       ** Display/Modify Sender Channel **

    Scrty Data: ________________________________
    Scrty Exit: ________________________________




 NOTE: RUNMQSC must be used to update the Send, Receive and Message
       Data and Exit attributes of channels









                                                PGUP - Return
```

*Figure 20. The Display/Modify Sender Channel panel (2)*

Overtype those values you want to modify, and press the Modify key (F1). You are prevented from overtyping those values that cannot be modified.

## Creating a channel definition

From the Channel Menu, press the Create key (F2) to display the Create Channel panel:



```
               IBM MQSeries for Compaq NonStop Kernel Version 5.1
                            ** Create Channel **

         Channel Type: 1   1=Sender, 2 = Server, 3=Receiver,
                           4=Requester, 5 = Server Connection
                           6 = Cluster Sender, 7 = Cluster Receiver
                 Name: ____________________

        Replace [Y/N]: _









 F1 - Enter                                                  F16 - Return
```

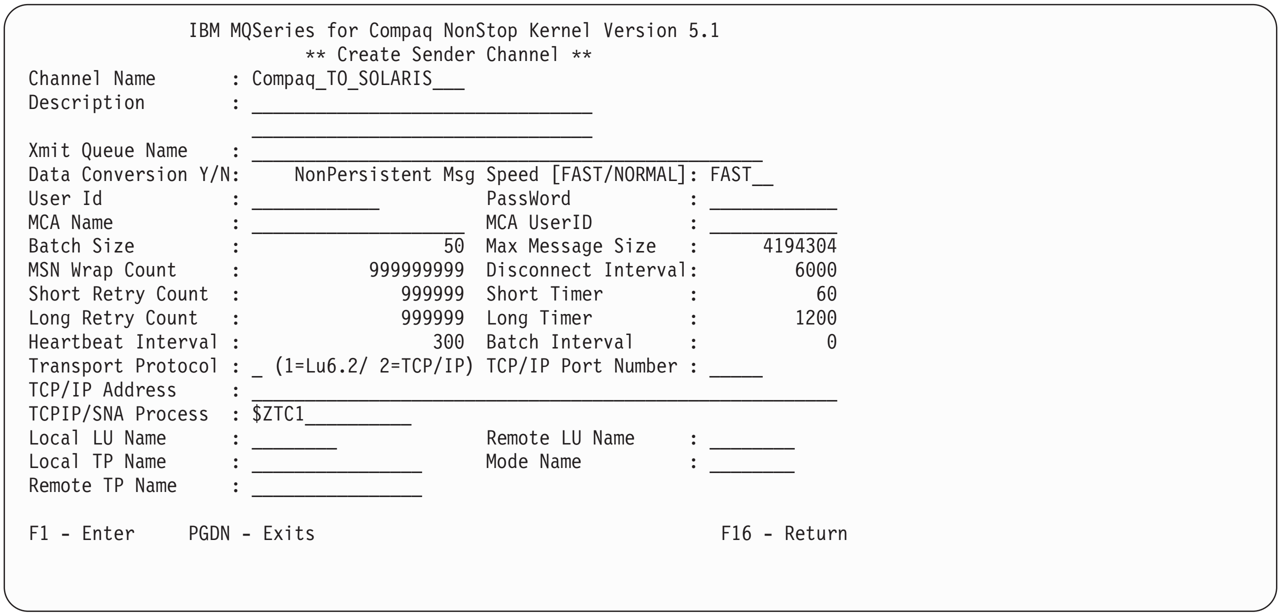
*Figure 21. The Create Channel panel*

To create a new channel definition:

1. Type 1 (for a sender channel), 2 (for a server channel), 3 (for a receiver channel), 4 (for a requester channel), or 5 (for a server connection) in the Channel Type field.
2. Type the name of the channel definition in the Name field.

3. Press the Enter key (F1).
4. Type Y in the `Replace` field if the definition is to replace an existing definition of the same name and type.

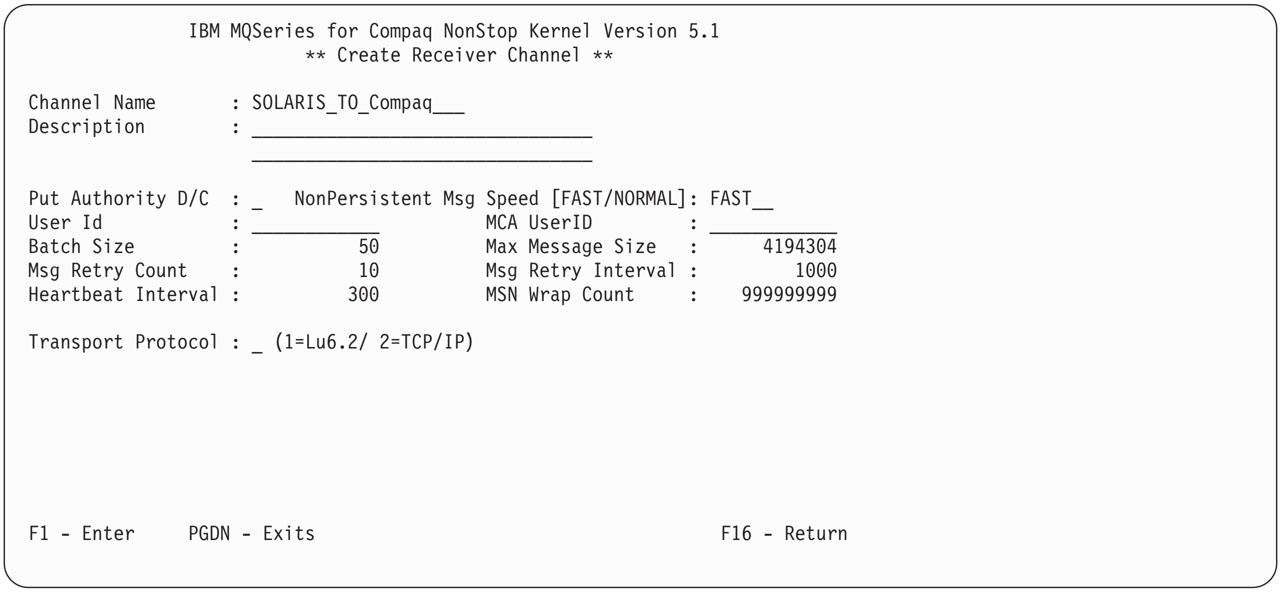
If you enter a 1 in the Channel Type field, the Create Sender Channel panel is displayed:



```
               IBM MQSeries for Compaq NonStop Kernel Version 5.1
                          ** Create Sender Channel **
Channel Name       : Compaq_TO_SOLARIS___
Description        : ________________________________
                     ________________________________
Xmit Queue Name    : ________________________________________________
Data Conversion Y/N:     NonPersistent Msg Speed [FAST/NORMAL]: FAST__
User Id            : ____________          PassWord           : ____________
MCA Name           : ____________________  MCA UserID         : ____________
Batch Size         :                   50  Max Message Size   :      4194304
MSN Wrap Count     :            999999999  Disconnect Interval:         6000
Short Retry Count  :               999999  Short Timer        :           60
Long Retry Count   :               999999  Long Timer         :         1200
Heartbeat Interval :                  300  Batch Interval     :            0
Transport Protocol : _ (1=Lu6.2/ 2=TCP/IP) TCP/IP Port Number : _____
TCP/IP Address     : _______________________________________________________
TCPIP/SNA Process  : $ZTC1__________
Local LU Name      : ________              Remote LU Name     : ________
Local TP Name      : ________________      Mode Name          : ________
Remote TP Name     : ________________

F1 - Enter     PGDN - Exits                                      F16 - Return
```

*Figure 22. The Create Sender Channel panel*

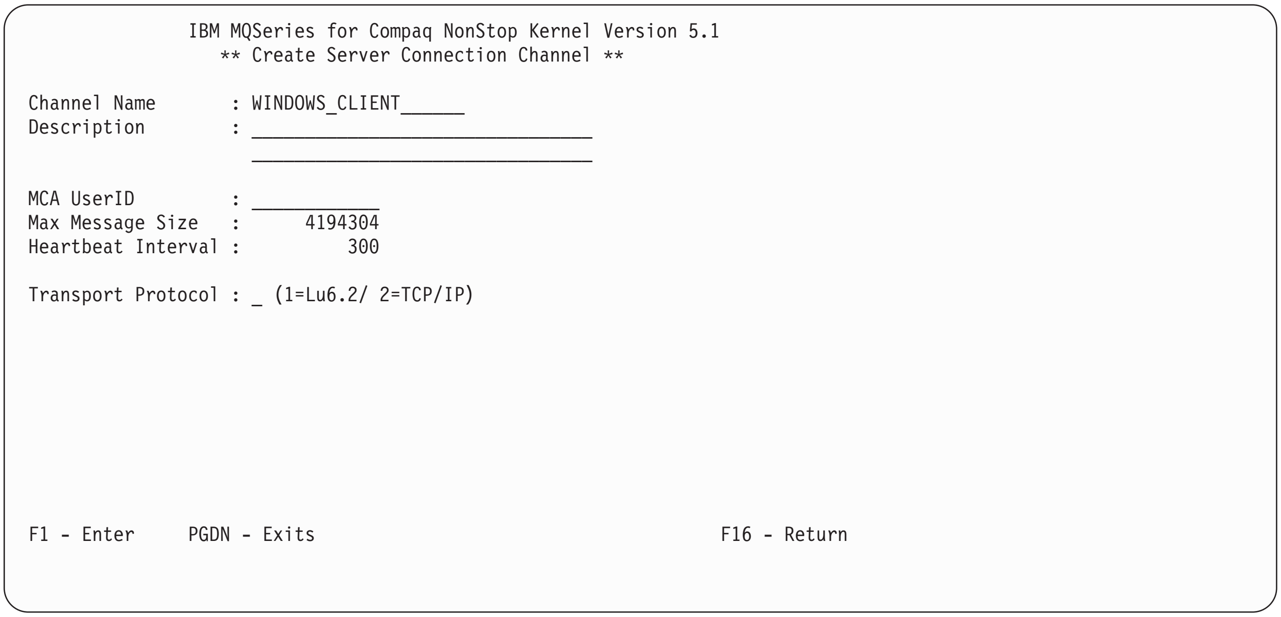
If you enter a 3 in the Channel Type field, the Create Receiver Channel panel is displayed:



```
               IBM MQSeries for Compaq NonStop Kernel Version 5.1
                          ** Create Receiver Channel **

Channel Name       : SOLARIS_TO_Compaq___
Description        : ________________________________
                     ________________________________

Put Authority D/C  : _   NonPersistent Msg Speed [FAST/NORMAL]: FAST__
User Id            : ____________          MCA UserID         : ____________
Batch Size         :           50          Max Message Size   :      4194304
Msg Retry Count    :           10          Msg Retry Interval :         1000
Heartbeat Interval :          300          MSN Wrap Count     :    999999999

Transport Protocol : _ (1=Lu6.2/ 2=TCP/IP)






F1 - Enter     PGDN - Exits                                      F16 - Return
```

*Figure 23. The Create Receiver Channel panel*

If you enter a 5 in the Channel Type field, the Create Server Connection Channel panel is displayed:

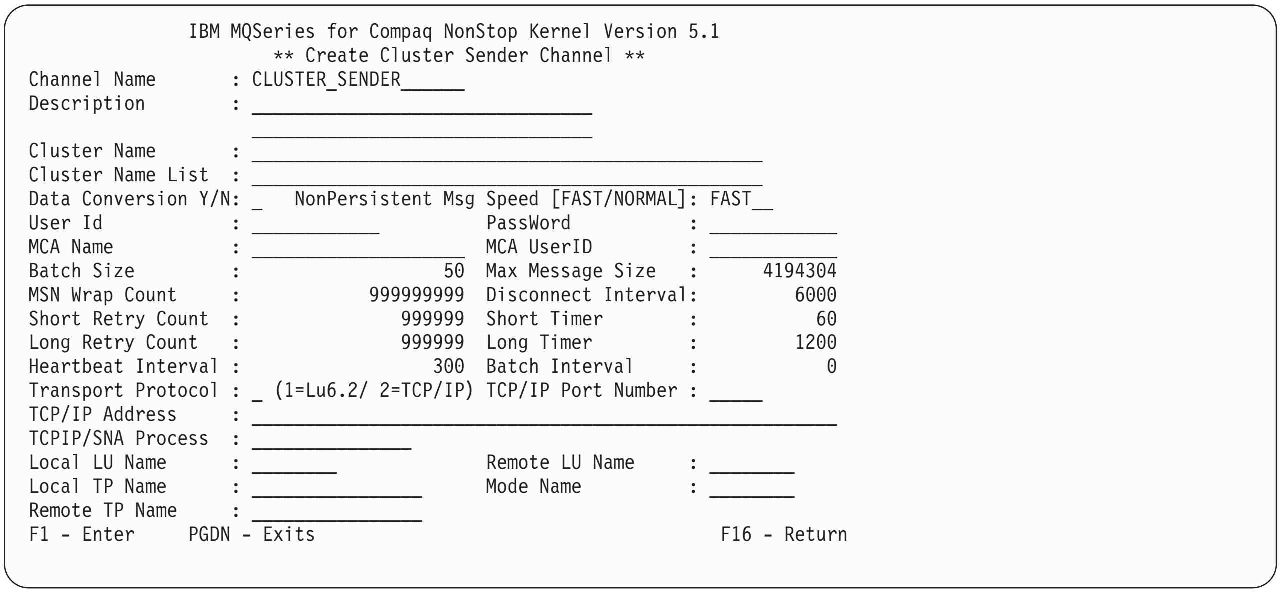


```
               IBM MQSeries for Compaq NonStop Kernel Version 5.1
                  ** Create Server Connection Channel **

Channel Name       : WINDOWS_CLIENT______
Description        : ________________________________
                     ________________________________

MCA UserID         : ____________
Max Message Size   :      4194304
Heartbeat Interval :          300

Transport Protocol : _ (1=Lu6.2/ 2=TCP/IP)









F1 - Enter     PGDN - Exits                                     F16 - Return
```

Figure 24. The Create Server Connection Channel panel

To create a new channel definition, complete the requested panel and press the Enter key (F1).

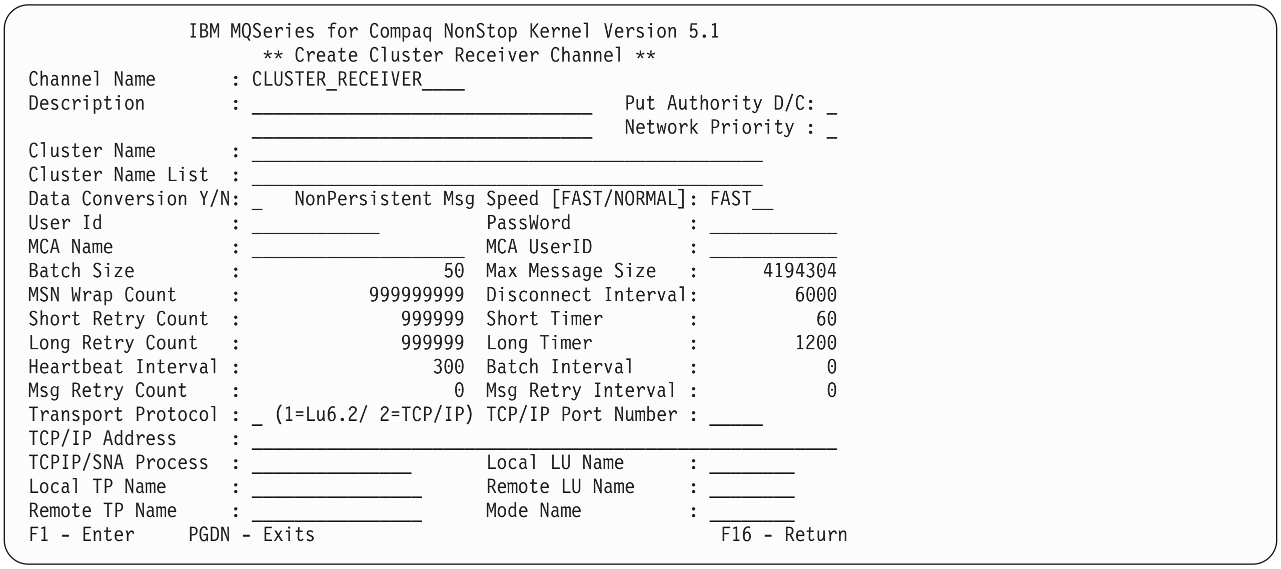
If you enter a 6 in the Channel Type field, the Create Cluster Sender Channel panel is displayed:



```
               IBM MQSeries for Compaq NonStop Kernel Version 5.1
                       ** Create Cluster Sender Channel **
Channel Name       : CLUSTER_SENDER______
Description        : ________________________________
                     ________________________________
Cluster Name       : ________________________________________________
Cluster Name List  : ________________________________________________
Data Conversion Y/N: _   NonPersistent Msg Speed [FAST/NORMAL]: FAST__
User Id            : ____________          PassWord           : ____________
MCA Name           : ____________________  MCA UserID         : ____________
Batch Size         :                   50  Max Message Size   :      4194304
MSN Wrap Count     :            999999999  Disconnect Interval:         6000
Short Retry Count  :               999999  Short Timer        :           60
Long Retry Count   :               999999  Long Timer         :         1200
Heartbeat Interval :                  300  Batch Interval     :            0
Transport Protocol : _ (1=Lu6.2/ 2=TCP/IP) TCP/IP Port Number : _____
TCP/IP Address     : _______________________________________________________
TCPIP/SNA Process  : _______________
Local LU Name      : ________              Remote LU Name     : ________
Local TP Name      : ________________      Mode Name          : ________
Remote TP Name     : ________________
F1 - Enter     PGDN - Exits                                     F16 - Return
```

Figure 25. The Create Cluster Sender Channel panel

If you enter a 7 in the Channel Type field, the Create Cluster Receiver Channel panel is displayed:

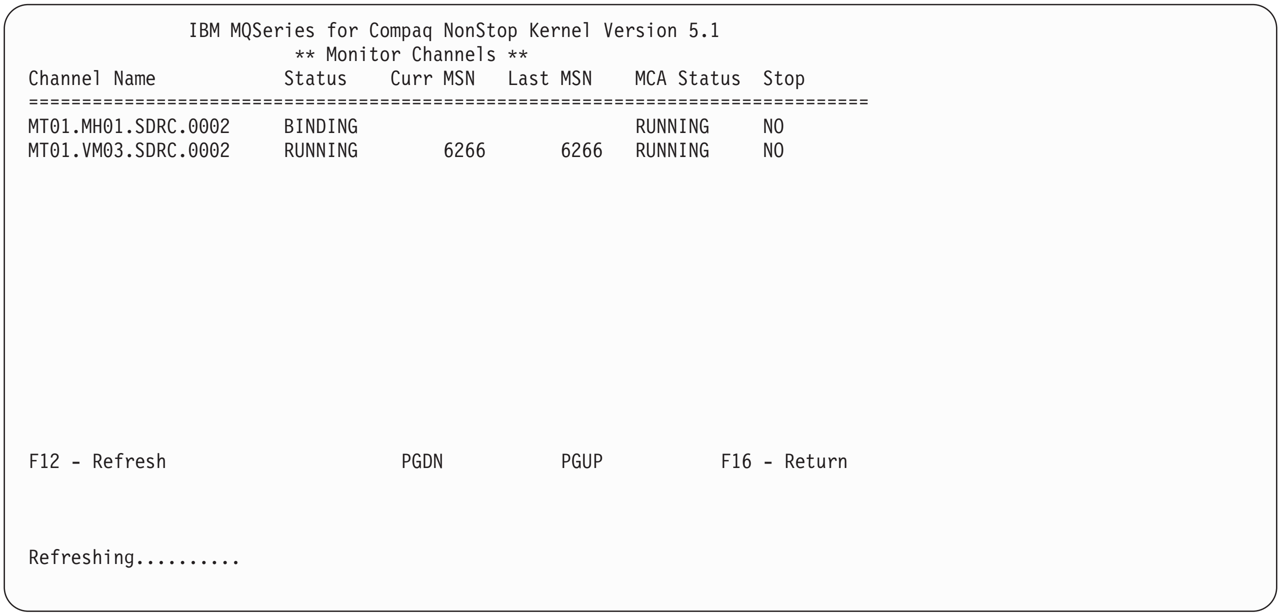


```
               IBM MQSeries for Compaq NonStop Kernel Version 5.1
                      ** Create Cluster Receiver Channel **
Channel Name       : CLUSTER_RECEIVER____
Description        : ________________________________   Put Authority D/C: _
                     ________________________________   Network Priority : _
Cluster Name       : ________________________________________________
Cluster Name List  : ________________________________________________
Data Conversion Y/N: _   NonPersistent Msg Speed [FAST/NORMAL]: FAST__
User Id            : ____________          PassWord          : ____________
MCA Name           : ____________________  MCA UserID        : ____________
Batch Size         :                   50  Max Message Size  :     4194304
MSN Wrap Count     :            999999999  Disconnect Interval:       6000
Short Retry Count  :               999999  Short Timer        :         60
Long Retry Count   :               999999  Long Timer         :       1200
Heartbeat Interval :                  300  Batch Interval     :          0
Msg Retry Count    :                    0  Msg Retry Interval :          0
Transport Protocol : _ (1=Lu6.2/ 2=TCP/IP) TCP/IP Port Number : _____
TCP/IP Address     : _______________________________________________________
TCPIP/SNA Process  : _______________       Local LU Name      : ________
Local TP Name      : ________________      Remote LU Name     : ________
Remote TP Name     : ________________      Mode Name          : ________
F1 - Enter     PGDN - Exits                                  F16 - Return
```

*Figure 26. The Create Cluster Receiver Channel panel*

## Monitoring a channel

Press the Monitor key (F5) from the Channel Menu panel to display the Monitor Channels panel:



```
               IBM MQSeries for Compaq NonStop Kernel Version 5.1
                         ** Monitor Channels **
Channel Name            Status    Curr MSN   Last MSN    MCA Status  Stop
===============================================================================
MT01.MH01.SDRC.0002     BINDING                          RUNNING     NO
MT01.VM03.SDRC.0002     RUNNING        6266       6266   RUNNING     NO












F12 - Refresh                      PGDN           PGUP           F16 - Return




Refreshing..........
```

*Figure 27. The Monitor Channels panel*

The MQMQMREFRESHINT PATHWAY parameter for MQS-MQMSVR00 determines the frequency with which monitor screens for channels and queues are refreshed. The default frequency is every 30 seconds. To change the frequency to every 10 seconds, for example, enter from the PATHWAY for your queue manager:

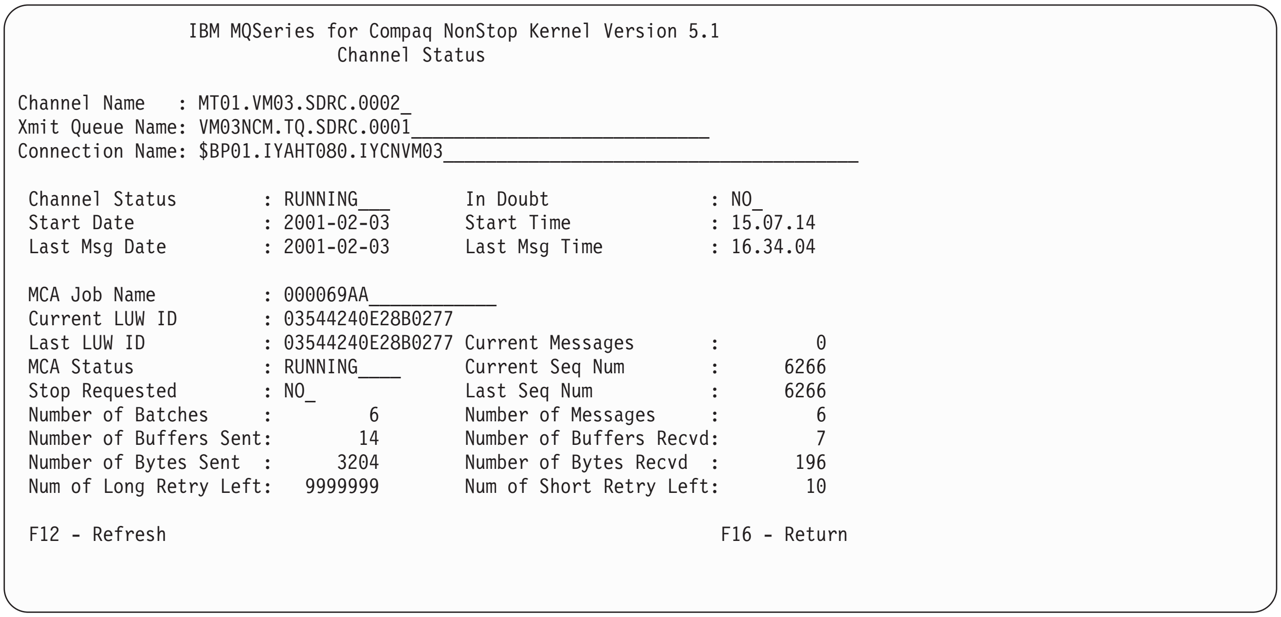
```
ALTER SERVER MQS-MQMSVR00, PARAM MQMQMREFRESHINT 10
```

## Deleting a channel

On the Channel Menu, select a channel to delete by typing any character against the channel name. Press the Delete key (F4) to delete the channel, then press F4 again to confirm the deletion request.

## Displaying channel status

Press the Status key (F10) from the Channel Menu panel to display the Channel Status panel:

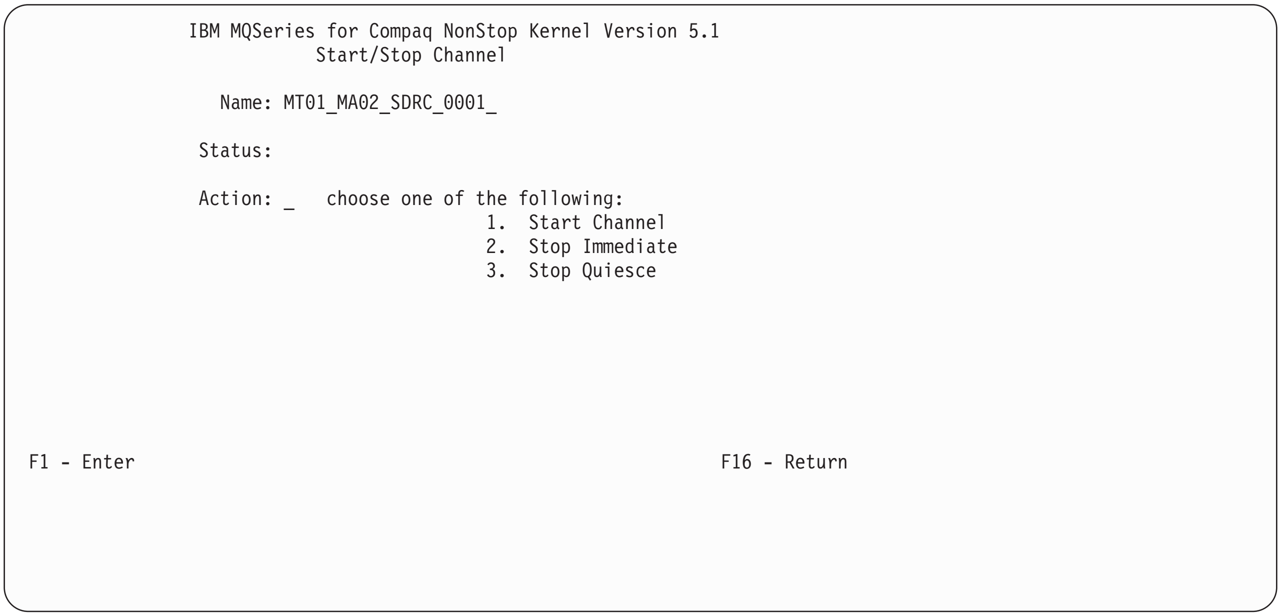


```
                IBM MQSeries for Compaq NonStop Kernel Version 5.1
                              Channel Status

Channel Name   : MT01.VM03.SDRC.0002_
Xmit Queue Name: VM03NCM.TQ.SDRC.0001____________________________
Connection Name: $BP01.IYAHT080.IYCNVM03_______________________________________

 Channel Status        : RUNNING___       In Doubt              : NO_
 Start Date            : 2001-02-03       Start Time            : 15.07.14
 Last Msg Date         : 2001-02-03       Last Msg Time         : 16.34.04

 MCA Job Name          : 000069AA____________
 Current LUW ID        : 03544240E28B0277
 Last LUW ID           : 03544240E28B0277 Current Messages      :          0
 MCA Status            : RUNNING____      Current Seq Num       :       6266
 Stop Requested        : NO_              Last Seq Num          :       6266
 Number of Batches     :          6       Number of Messages    :          6
 Number of Buffers Sent:         14       Number of Buffers Recvd:         7
 Number of Bytes Sent  :       3204       Number of Bytes Recvd :        196
 Num of Long Retry Left:    9999999       Num of Short Retry Left:        10

 F12 - Refresh                                            F16 - Return
```

*Figure 28. The Channel Status panel*

## Starting and stopping a channel

Press the Start/Stop key (F8) from the Channel Menu to display the Start/Stop Channel panel:



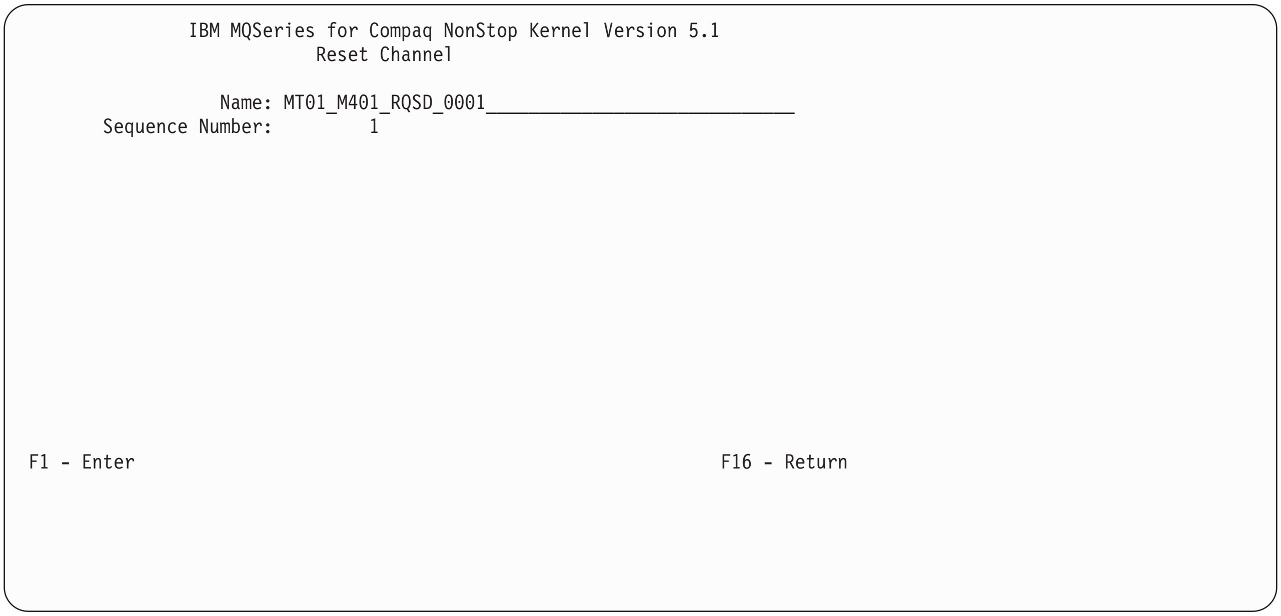
```
              IBM MQSeries for Compaq NonStop Kernel Version 5.1
                        Start/Stop Channel

                 Name: MT01_MA02_SDRC_0001_

               Status:

               Action: _   choose one of the following:
                                          1.  Start Channel
                                          2.  Stop Immediate
                                          3.  Stop Quiesce






F1 - Enter                                                      F16 - Return
```

*Figure 29. The Start/Stop Channel panel*

Type the name of the channel in the Name field, and type a number between 1 and 3 in the Action field. Press the Enter key (F1).

## Resetting a Message Sequence Number (MSN)

From the Channel Menu, press the Reset MSN key (F7) to display the Reset Channel panel:



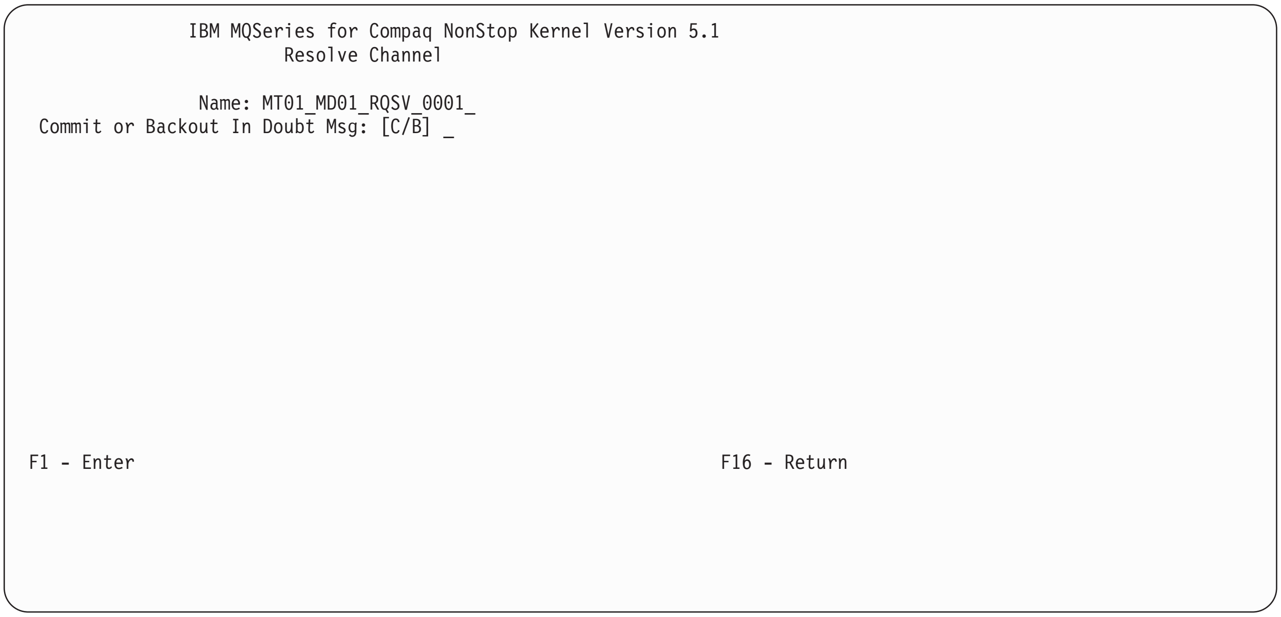
```
              IBM MQSeries for Compaq NonStop Kernel Version 5.1
                        Reset Channel

                 Name: MT01_M401_RQSD_0001____________________________
      Sequence Number:         1










F1 - Enter                                                      F16 - Return
```

*Figure 30. The Reset Channel panel*

The MSN ensures nonduplication of messages, and ensures that messages are stored in the same order as they are transmitted. This screen lets you reset the sequence number of a channel if necessary.

## Resolving a channel

From the Channel Menu, press the Resolve key (F6) to display the Resolve Channel panel.



```
               IBM MQSeries for Compaq NonStop Kernel Version 5.1
                        Resolve Channel

                Name: MT01_MD01_RQSV_0001_
 Commit or Backout In Doubt Msg: [C/B] _















F1 - Enter                                                    F16 - Return
```

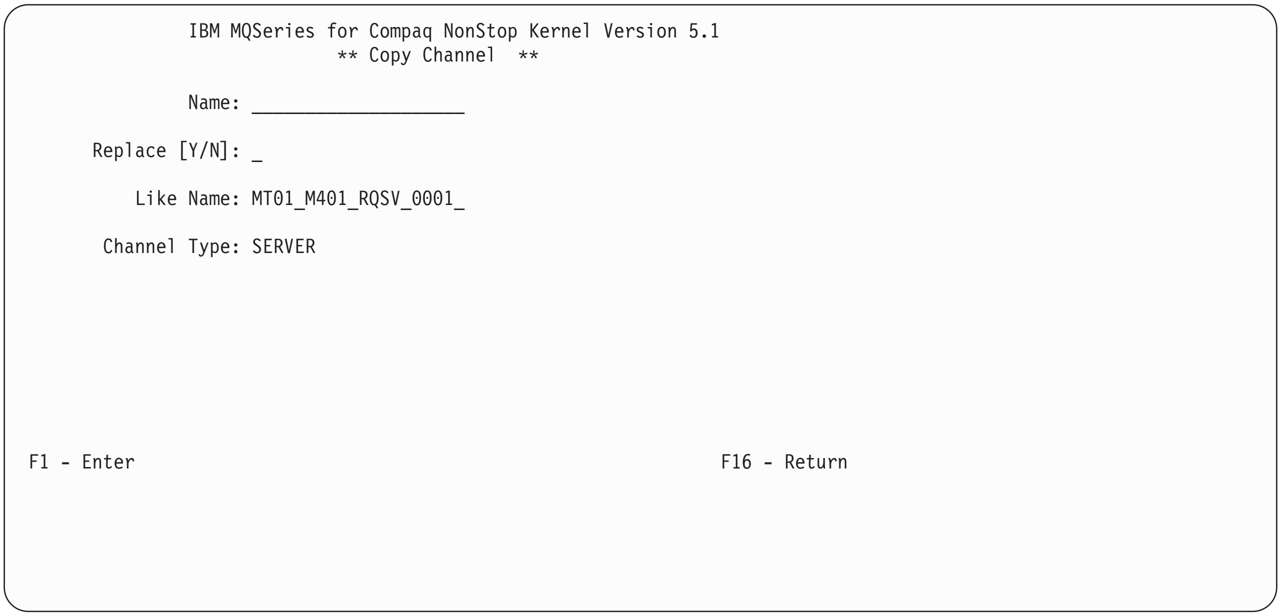
*Figure 31. The Resolve Channel panel*

You can:

- Backout the in-doubt message batch (B)
- Commit the in-doubt message batch (C)

## Copying a channel

On the Channel Menu, press the Copy key (F3). The Copy Channel panel is displayed:



```
              IBM MQSeries for Compaq NonStop Kernel Version 5.1
                           ** Copy Channel  **

              Name: ____________________

     Replace [Y/N]: _

         Like Name: MT01_M401_RQSV_0001_

      Channel Type: SERVER






F1 - Enter                                                        F16 - Return
```

*Figure 32. The Copy Channel panel*

Type the name of the new channel in the Name field; type the name of the channel definition you are copying in the Like Name field; type the channel type in the Channel Type field. Press the Enter key (F1) to copy the channel definition.

# Chapter 5. Administering local MQSeries objects

This chapter explains how to administer local MQSeries objects to support application programs that use the Message Queuing Interface (MQI). The MQI lets application programs access message queuing services.

Local administration is when you create, display, change, copy, and delete MQSeries objects.

This chapter contains these sections:

- "Supporting application programs that use the MQI"
- "Performing local administration tasks using MQSC commands" on page 84
- "Running MQSC commands from text files" on page 87
- "Resolving problems with MQSC" on page 91
- "Working with local queues" on page 92
- "Working with alias queues" on page 100
- "Working with model queues" on page 102
- "Managing objects for triggering" on page 103

## Supporting application programs that use the MQI

MQI application programs need specific objects before they can run successfully. An MQI application can remove messages from a queue, process them, and send the results to another queue on the same queue manager.

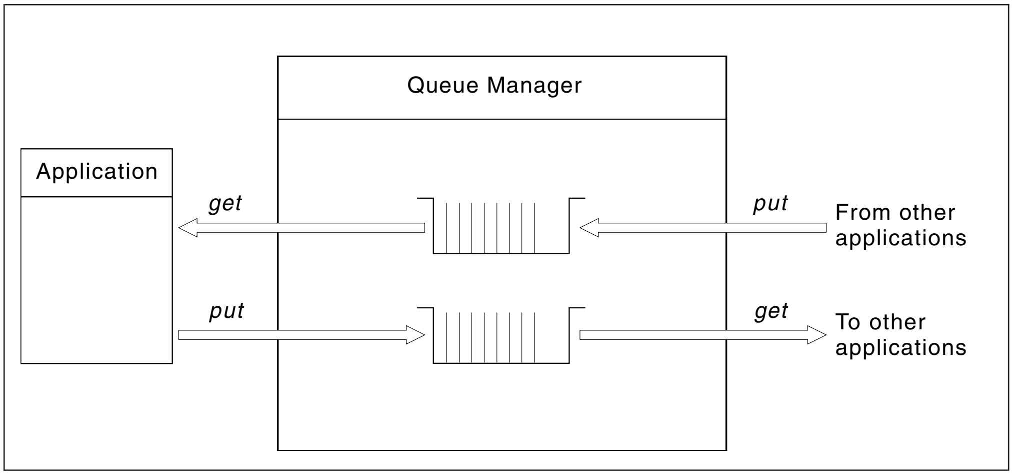


*Figure 33. Queues, messages, and applications*

Whereas applications can put (using MQPUT) messages on local or remote queues, they can only get (using MQGET) messages directly from local queues.

Before this application can be run, these conditions must be satisfied:

- The queue manager must exist and be running.
- The first application queue, from which the messages are to be removed, must be defined.
- The second queue, on which the application puts the messages, must also be defined (unless it is a dynamic queue).

© Copyright IBM Corp. 1993, 2001

### Supporting applications

- The application must be able to connect to the queue manager. To perform this task, it must be linked to the product code. See “Appendix I. Building and running applications” on page 323 for more information.
- The applications that put the messages on the first queue must also connect to a queue manager. If they are remote, they must also be set up with transmission queues and channels.

# Performing local administration tasks using MQSC commands

MQSeries commands (MQSC) let you manipulate MQSeries objects. You can issue commands using the **runmqsc** command at the command prompt.

See “Appendix G. MQSC supported by MQSeries for Compaq NSK” on page 311 for more information about using MQSC in the MQSeries for Compaq NSK environment.

You can use MQSeries script commands (MQSC) to manage queue manager objects, including the queue manager itself, clusters, channels, queues, namelists and process definitions. This section deals with queue managers, queues and process definitions; for information about administering channel objects, see DQM implementation in the *MQSeries Intercommunication* book.

You issue MQSC commands to a queue manager using the **runmqsc** command. You can do this interactively, issuing commands from the keyboard, or you can redirect standard input to run a sequence of commands from an ASCII text file. In both cases, the format of the commands is the same.

You can run the **runmqsc** command in three modes, depending on the flags set on the command:

- *Verification mode*, where the MQSC commands are verified on a local queue manager, but are not actually run.
- *Direct mode*, where the MQSC commands are run on a local queue manager.
- *Indirect mode*, where the MQSC commands are run on a remote queue manager.

Object attributes specified in MQSC are shown in this book in upper case (for example, RQMNAME) although they are not case sensitive. MQSC attribute names are limited to eight characters.

## Before you start

Before you begin, you must create and then start the queue manager, which runs the MQSC commands. See “Creating a default queue manager” on page 59 for more information.

### MQSeries object names

In examples, we use some long names for objects. This is to help you identify what type of object it is you are dealing with.

When you are issuing MQSC commands, you need only specify the local name of the queue. In our examples, we use queue names such as: `ORANGE.LOCAL.QUEUE`

The LOCAL.QUEUE part of the name is simply to illustrate that this queue is a local queue. It is *not* required for the names of local queues in general.

We also use the name `saturn.queue.manager` as a queue manager name.

The `queue.manager` part of the name is simply to illustrate that this object is a queue manager. It is *not* required for the names of queue managers in general.

You do not have to use these names, but if you do not, you must modify any commands in examples that specify them.

### Case sensitivity on MQSC commands

MQSC commands and their attributes can be in upper case or lower case letters and are not case sensitive. Object names on the other hand are case sensitive (that is, input-queue and INPUT-QUEUE are different objects). Object names in MQSC commands are folded to upper case (that is, QUEUE and queue are not differentiated), unless the names are put in single quotation marks. If quotation marks are not used, upper case letters are used for the object name. See the *MQSeries MQSC Command Reference* for more information.

However, some arguments of the **runmqsc** command, which invokes the MQSC facility, are case sensitive; see "Using control commands" on page 27.

## Using the MQSC facility interactively

To enter commands interactively, open a TACL session and enter:

```
runmqsc
```

In this example, a queue manager name has not been specified, therefore the MQSCs are processed by the default queue manager. You can enter any MQSC command. For example:

```
MQSC>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.

The **runmqsc** command also supports the standard Compaq NSK history and fix command facilities. For example:

- Typing `history` or `h` produces a list of the ten most recent commands
- Typing `!`*n* where *n* is the command number will re-execute that command
- Typing `h` *n* or `history` *n* where *n* is a number will list the n most recent commands
- Typing `fc` presents the last command entered for editing. Typing `fc` *n* where *n* is the command number presents that command for editing. Typing `fc` *string* where *string* is the beginning part or all of a previously entered command presents the last occurrence of that command for editing. The syntax is NSK standard. For example, type `d` to delete a character, `i` to insert a character and `r` to replace a character.

## Feedback from MQSCs

When you issue MQSCs, the queue manager provides confirmation or error messages. For example:

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

The first message confirms that a queue has been successfully created. The second message indicates that you have made a syntax error. If you have not entered the command correctly, refer to the *MQSeries MQSC Command Reference* for the correct syntax.

## Ending interactive input to MQSC

If you are using MQSC interactively, you can exit by entering the EOF character CTRL+Y, or by typing `exit` or `quit` or `end` and pressing Enter.

If you are redirecting input from other sources, such as a text file, MQSC terminates when the end of file is reached.

## Displaying queue manager attributes

To display the attributes of the queue manager specified on the **runmqsc** command, use the following MQSeries command:

```
MQSC>DISPLAY QMGR ALL
```

A typical output is displaying in Figure 34:

```
    1 : dis qmgr all
AMQ8408: Display Queue Manager details.
   DESCR( )                               DEADQ(SYSTEM.DEAD.LETTER.QUEUE)
   DEFXMITQ(MY.DEFAULT.XMIT.QUEUE)        CHADEXIT( )
   CLWLEXIT( )                            CLWLDATA( )
   REPOS( )                               REPOSNL( )
   COMMANDQ(SYSTEM.ADMIN.COMMAND.QUEUE)   QMNAME(saturn.queue.manager)
   CRDATE(2001-03-12)                     CRTIME(09.24.30)
   ALTDATE(2001-03-12)                    ALTTIME(09.26.27)
   QMID(SIMONW_2001-03-12_09.24.30)       TRIGINT(10000)
   MAXHANDS(256)                          MAXUMSGS(10000)
   AUTHOREV(ENABLED)                      INHIBTEV(ENABLED)
   LOCALEV(ENABLED)                       REMOTEEV(ENABLED)
   PERFMEV(ENABLED)                       STRSTPEV(ENABLED)
   CHAD(DISABLED)                         CHADEV(ENABLED)
   CLWLLEN(100)                           MAXMSGL(100000000)
   CCSID(819)                             MAXPRTY(9)
   CMDLEVEL(510)                          PLATFORM(NSK)
   SYNCPT                                 DISTL(YES)
```

*Figure 34. Example output for QMGR ALL*

The ALL parameter on the DISPLAY QMGR command causes all the queue manager attributes to be displayed. The output tells us the queue manager name (`saturn.queue.manager`), and the names of the dead-letter queue

(SYSTEM.DEAD.LETTER.QUEUE) and the command queue (SYSTEM.ADMIN.COMMAND.QUEUE). Note that, if you do not specify the name of a dead-letter queue on the **crtmqm** command, you must alter the queue manager to associate a dead-letter queue with the queue manager.

You should confirm that these queues are created by entering the command:

```
DISPLAY QUEUE (*)
```

## Using a queue manager that is not the default

You can specify a queue manager name when executing the **runmqsc** command to run MQSCs on a local queue manager (other than the default). For example, to run MQSCs on queue manager named `jupiter.queue.manager`, use this command:

```
runmqsc jupiter.queue.manager
```

All the MQSCs you enter are processed by this queue manager providing the queue manager is on the same node and is already running.

You can also run MQSC commands on a remote queue manager; see "Issuing MQSC commands remotely" on page 117.

## Altering queue manager attributes

To alter the attributes of the queue manager specified with the **runmqsc** command, use the MQSC ALTER QMGR, specifying the attributes and values that you want to change. For example, use the following commands to alter the attributes of `jupiter.queue.manager`:

```
runmqsc jupiter.queue.manager

ALTER QMGR DEADQ (ANOTHERDLQ) INHIBTEV (ENABLED)
```

The ALTER QMGR command changes the dead-letter queue used, and enables inhibit events.

# Running MQSC commands from text files

Running MQSCs interactively is appropriate for quick tests; however, if you have long commands, or commands that you want to repeat, you should take input from a text file.

To perform this task, create a text file containing the MQSCs using your text editor. When you use the **runmqsc** command, use the TACL IN and OUT redirection operators, or the flags -i and -o on **runmqsc**. For example, the following command runs a sequence of commands contained in the text file `mymqscin`:

```
runmqsc /IN  mymqscin/
```

or

```
runmqsc -i mymqscin
```

Similarly, you can redirect the output to a file. A file containing the MQSCs for input is called an *MQSC file*. The output file containing replies from the queue manager is called the *report file*.

To redirect both input and output on the **runmqsc** command, use this command:

```
runmqsc /IN mymqscin, OUT mymqscou/
```

or

```
runmqsc -i mymqscin -o mymqscou
```

This command invokes the MQSC commands contained in the file `mymqscin`. Because a queue manager name is not specified, the MQSC commands are run against the default queue manager. The output is sent to the report file `mymqscou`. Figure 35 on page 89 shows an extract from the MQSC command file `mymqscin`, and Figure 36 on page 90 shows the corresponding extract of the output in `mymqscou`.

To redirect input and output on the **runmqsc** command for a queue manager (`saturn.queue.manager`) that is not the default, use the command:

```
runmqsc /IN mymqscin, OUT mymqscou/ saturn.queue.manager
```

or

```
runmqsc -i mymqscin -o mymqscou  saturn.queue.manager
```

## Using MQSC command files

MQSC command files are written as EDIT files (Compaq file type code 101). Figure 35 on page 89 is an extract from an MQSC file showing an MQSeries command (DEFINE QLOCAL) with its attributes. The *MQSeries MQSC Command Reference* contains a description of each MQSC command and its syntax.

```
 .
 .
 .
DEFINE QLOCAL(ORANGE.LOCAL.QUEUE) REPLACE  +
       DESCR(' ') +
       PUT(ENABLED) +
       DEFPRTY(0) +
       DEFPSIST(NO) +
       GET(ENABLED) +
       MAXDEPTH(5000) +
       MAXMSGL(1024) +
       DEFSOPT(SHARED) +
       NOHARDENBO +
       USAGE(NORMAL) +
       NOTRIGGER
 .
 .
 .
```

*Figure 35. Extract from the MQSC command file, mymqscin*

You must limit lines to a maximum of 72 characters. The plus sign (+) indicates that the command is continued on the next line. Note that the plus sign must be preceded by a space.

## Using MQSC reports

The *runmqsc* command returns a *report*, which is sent to the current OUT stream The report contains:

- A header identifying MQSC as the source of the report:

```
Starting MQSeries Commands.
```

- An optional numbered listing of the MQSC commands issued. By default, the text of the input is echoed to the output. Within this output, each command is prefixed by a sequence number, as shown in Figure 36 on page 90. However, you can use the -e flag on the **runmqsc** command to suppress the output.
- A syntax error message for any commands found to be in error.
- An *operator message* indicating the outcome of running each command. For example, the operator message for the successful completion of a DEFINE QLOCAL command is:

```
AMQ8006: MQSeries queue created.
```

- Other messages resulting from general errors when running the script file.
- A brief statistical summary of the report indicating the number of commands read, the number of commands with syntax errors, and the number of commands that could not be processed.

  **Note:** The queue manager attempts to process only those commands that have no syntax errors.

```
Starting MQSeries Commands.
 .
 .
     12:     DEFINE QLOCAL('RED.LOCAL.QUEUE') REPLACE  +
       :            DESCR(' ') +
       :            PUT(ENABLED) +
       :            DEFPRTY(0) +
       :            DEFPSIST(NO) +
       :            GET(ENABLED) +
       :            MAXDEPTH(5000) +
       :            MAXMSGL(1024) +
       :            DEFSOPT(SHARED) +
       :            USAGE(NORMAL) +
       :            NOTRIGGER
AMQ8006: MQSeries queue created.
       :
 .
 .
15 MQSC commands read.
0 commands have a syntax error.
0 commands cannot be processed.
```

*Figure 36. Extract from the MQSC report file, mymqscou*

## Running the supplied MQSC command file

When you install MQSeries for Compaq NSK, an MQSC file called **AMQSCOS0** is supplied. This file contains the definitions of objects used by sample programs. The file is located in the samples subvolume, by default $SYSTEM.ZMQSSMPL.

## Using runmqsc to verify commands

You can use the **runmqsc** command to verify MQSC commands on a local default queue manager without actually running them. To perform this step, set the -v flag on the **runmqsc** command. For example:

```
runmqsc -i mymqscin -o mymqscou -v
```

When you invoke **runmqsc** against an MQSC command file, the queue manager verifies each command and returns a report without actually running the MQSC commands. This action lets you check the syntax of all the commands in your command file. This step is important if you are:

- Running a large number of commands from a command file
- Using an MQSC command file many times over.

This report is similar to that shown in Figure 36.

You cannot use this method to verify MQSC commands remotely. For example, if you attempt this command:

```
runmqsc -i mymqscin -o mymqscou -w 30 -v jupiter.queue.manager
```

the -w flag, which you use to indicate that the queue manager is remote, is ignored, and the command is run locally in verification mode .

## Resolving problems with MQSC

If the MQSC commands do not run properly, use the following checklist to see if any of these common problems apply to you.

When you use the **runmqsc** command:

- Check that $SYSTEM.ZMQSEXE is in PMSEARCH in TACLCSTM.
- Use the IN operator or the -i flag when redirecting input from a file. Otherwise, the queue manager interprets the file name as a queue manager name and issues the following error message:

```
AMQ8118: MQSeries queue manager does not exist.
```

- If you redirect output to a file, use the OUT operator or the -o flag. By default, the output file is created using the TACL defaults in effect at the time the command was issued. Specify a fully qualified file name to send your output to a specific file.
- Check that you created the queue manager that is going to run the commands. To do this, look in the configuration file MQSINI, which by default is located in the installation subvolume, $SYSTEM.ZMQSSYS. This file contains the names of the queue managers and the name of the default queue manager, if you have one.
- The queue manager should already be started; if it is not, start it, as described in “Starting a queue manager” on page 60. You get an error message if the queue manager is already started.
- Specify a queue manager name on the **runmqsc** command if you have not defined a default queue manager, otherwise you get this error:

```
AMQ8146: MQSeries queue manager not available.
```

  For information about correcting this type of problem, see “Making an existing queue manager the default” on page 62.
- You cannot specify an MQSC command as a **runmqsc** parameter. For example, the following is invalid:

```
runmqsc DEFINE QLOCAL(FRED)
```

- You cannot enter MQSC commands from TACL before you issue the **runmqsc** command. For example:

```
DEFINE QLOCAL(QUEUE1)


* Error Name of Variable, built-in, or file needed.
```

- You cannot run control commands from **runmqsc**. For example, you cannot start a queue manager once you are running MQSC interactively:

```
runmqsc
(C) Copyright IBM Corp. 1993, 2001. All Rights Reserved
Starting MQSeries Commands.

strmqm saturn.queue.manager
     1 : strmqm saturn.queue.manager
AMQ8405: Syntax error detected at or near end of command segment below:
```

See also “If you have problems using MQSC remotely” on page 118.

# Working with local queues

This section contains examples of some of the MQSC commands that you can use. Refer to the *MQSeries MQSC Command Reference* for a complete description of these commands.

## 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 local to that queue manager.

Use the MQSC 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, 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 performs this action:

```
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.
2. USAGE (NORMAL) indicates that this queue is not 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 94.

## Changing the physical file size for queues

By default, the queue manager creates queue data files that support up to 100 MB of data. If this limit is reached, applications receive the return code MQRC_Q_SPACE_NOT_AVAILABLE. To change the maximum storage allocated to a queue, first identify the physical files that hold the data for the queue using **dspmqfls**, then use the **altmqfls** command to resize the primary and secondary extent sizes and the maximum extents. If the file is already partitioned, **altmqfls** will resize all of the partitions. For more information, see "altmqfls (Alter queue file attributes)" on page 230. If more storage is required, you can partition the file across multiple volumes.

## Defining a dead-letter queue

Each queue manager should have a local queue to be used as a dead-letter queue so that messages that cannot be delivered to their correct destination can be stored for later retrieval.

You must tell the queue manager about the dead-letter queue. You can do this by specifying a dead-letter queue on the **crtmqm** command or you can use the ALTER QMGR command to specify one later. You must also define the dead-letter queue before it can be used.

A sample dead-letter queue called SYSTEM.DEAD.LETTER.QUEUE is supplied with the product. This queue is automatically created when the queue manager is created. You can modify this definition, if required. There is no need to rename it.

A dead-letter queue has no special requirements except that:

- It must be a local queue
- Its MAXMSGL (maximum message length) attribute must enable the queue to accommodate the largest messages that the queue manager has to handle **plus** the size of the dead-letter header (MQDLH).

MQSeries provides a dead-letter queue handler that lets you specify how messages found on a dead-letter queue are to be processed or removed. For further information, see "Chapter 9. MQSeries dead-letter queue handler" on page 145.

## 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. 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:

### Displaying default object attributes

```
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 a 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 92, 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.

## Changing the volume of a local queue

Use the **altmqfls** command to change the volume on which a local, predefined queue is stored. This might be necessary to spread disk I/O across volumes to balance the system for optimum performance. The queue manager must have been started before this command is issued, and the queue itself must not be open. Only one queue may be named on any **altmqfls** command. See "altmqfls (Alter queue file attributes)" on page 230 for the syntax of the **altmqfls** command.

## Changing the options for a local queue

Use the **altmqfls** command to change:

- Whether the queue is loaded from disk into cache when the queue manager starts. Use --qsoptions S.
- Whether the queue remains in memory while the queue server is running after it is first loaded. If set, the queue server retains the queue's structures and any data in memory once loaded. If not set, the storage associated with a queue may be removed from the queue server's address space when it is no longer being accessed. Use --qsoptions L.
- Whether the non-persistent messages are checkpointed to the backup queue server, providing fault tolerance at the expense of the CPU loading required to handle the extra checkpointing, extra IPC messages and extra memory required to store the messages. Use --qsoptions C.

  **Note:** *All* of the --qsoptions SLC are set each time the command is issued. For example, --qsoptions S will unset L and C. The --qsoptions option can be specified once and once only on a command line.
- The maximum number of bytes of data each persistent message can keep in the queue server's cache (as well as on disk). The number of bytes set are kept in memory and the browse operation returns this data to the application without having to access the disk. Using this will increase the memory resources in use by the queue server. Use the --browse parameter.

- The minimum message size used by a message overflow file to store message data. Persistent messages that are smaller than this are stored in the queue overflow file. persistent messages larger than this have their bulk data stored in a dedicated message overflow file. Use the --msgofthresh parameter.
- The subvolume on the volume where the queue resides where the queue server creates new message overflow files. Use the --msgofsubvol parameter.
- The name of a measure counter which, if part of an active measurement, is initialized to the current depth and then incremented and decremented by the queue server responsible for the queue when messages are added or removed from the queue. Use the --meascount parameter.
- The primary and secondary extent size and the maximum number of extents for the queue file. Use the --qsize (primaryextent,secondaryextent,maxextents) parameter. With the --qsize parameter, all of the values must be specified.
- The primary and secondary extent size and the maximum number of extents for the queue overflow file. Use the --osize (primaryextent,secondaryextent,maxextents) parameter. With the --osize parameter, all of the values must be specified.

## Reassigning objects to status servers and queue servers

Status servers handle all objects except local and model queues. Queue servers handle all local queue and model queue objects.

Initially, all objects are created to use the default queue server or status server, depending on the object. Using **altmqfls** *after* an object has been created, you can configure the object to use an appropriate server other than the default. You must have configured the new PATHWAY server class for the server and have started it before you can use the object.

You may specify either a process name or the word DEFAULT on the command line for **altmqfls**. No checking is performed that the new server is active or configured at the time the object is reconfigured.

You can use the **dspmqfls** command to display the current status server for an object.

Note that the queue server can be set for local and model queues. Dynamic queues inherit the queue server from the model queue that is used to create them.

## 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 only be deleted 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 for Compaq NSK provides a sample queue browser to enable you to look at the contents of the messages on a queue. The browser is supplied both as source and as a module that can be run. By default, the file names and paths are:

**Source** `$SYSTEM.ZMQSSMPL.AMQSBCG0`

**Executable** `$SYSTEM.ZMQSSMPL.AMQSBCG`

The sample takes two parameters:

**Queue name** For example, `SYSTEM.ADMIN.RESPQ.tpp01`.

**Queue manager name**
For example, `snooker`.

For example:

```
AMQSBCG SYSTEM.ADMIN.RESPQ.tpp01 snooker
```

There are no defaults; both parameters are required. Typical results from this command are:

```
AMQSBCG - starts here
**********************

 MQCONN to snooker
 MQOPEN - 'SYSTEM.ADMIN.RESPQ.tpp01'


 MQGET of message number 1
****Message descriptor****

  StrucId  : 'MD  '  Version : 1
  Report   : 0  MsgType : 8
  Expiry   : -1  Feedback : 0
  Encoding : 273  CodedCharSetId : 850
  Format : 'AMQMRESP'
  Priority : 5  Persistence : 1
  MsgId : X'414D5120736E6F6F6B657220202020202ED47690071A6D00'
  CorrelId : X'000000000000000000000000000000000000000000000000'
  BackoutCount : 0
  ReplyToQ       : '                                                '
  ReplyToQMgr    : 'snooker                                         '
  ** Identity Context
  UserIdentifier : 'tpp01       '
  AccountingToken :
   X'0437303730000000000000000000000000000000000000000000000000000000'
  ApplIdentityData : '                                '
  ** Origin Context
  PutApplType    : '6'
  PutApplName    : '                            '
  PutDate  : '19941124'    PutTime  : '11184015'
  ApplOriginData : '    '

****   Message      ****

 length - 268 bytes

00000000:  736E 6F6F 6B65 7220 2020 2020 2020 2020 'snooker         '
00000010:  2020 2020 2020 2020 2020 2020 2020 2020 '                '
00000020:  2020 2020 2020 2020 2020 2020 2020 2020 '                '
00000030:  534E 4F4F 4B45 522E 5749 4748 542E 5443 'SNOOKER.WIGHT.TC'
00000040:  5020 2020 2020 2020 2020 2020 2020 2020 'P               '
00000050:  2020 2020 2020 2020 2020 2020 2020 2020 '                '
00000060:  0000 0001 0000 0024 0000 0001 0000 0015 '.......$........'
00000070:  0000 0001 0000 0001 0000 0000 0000 0000 '................'
00000080:  0000 0003 0000 0004 0000 0028 0000 0DAD '...........(....'
00000090:  0000 0000 0000 0014 534E 4F4F 4B45 522E '........SNOOKER.'
000000A0:  5749 4748 542E 5443 5020 2020 0000 0003 'WIGHT.TCP   ....'
000000B0:  0000 0010 0000 05E7 0000 0001 0000 0004 '................'
000000C0:  0000 0050 0000 0DAE 0000 0000 0000 0039 '...P...........9'
000000D0:  2066 726F 6D20 736E 6F6F 6B65 7220 746F ' from snooker to'
000000E0:  2077 6967 6874 2076 6961 2074 6370 2F69 ' wight via tcp/i'
000000F0:  7020 2020 2020 2020 2020 2020 2020 2020 'p               '
00000100:  2020 2020 2020 2020 2000 0000           '         ...    '


 MQGET of message number 2
****Message descriptor****

  StrucId  : 'MD  '  Version : 1
  Report   : 0  MsgType : 2
  Expiry   : -1  Feedback : 0
  Encoding : 273  CodedCharSetId : 850
  Format : 'MQADMIN '
  Priority : 8  Persistence : 1
  MsgId : X'414D5120736E6F6F6B657220202020202ED476901524D200'
  CorrelId : X'414D5120736E6F6F6B657220202020202ED47690071A6D00'
  BackoutCount : 0
  ReplyToQ       : '                                                '
  ReplyToQMgr    : 'snooker                                         '
  ** Identity Context
  UserIdentifier : 'tpp01       '
  AccountingToken :
   X'0437303730000000000000000000000000000000000000000000000000000000'
  ApplIdentityData : '                                '
  ** Origin Context
  PutApplType    : '6'
  PutApplName    : '                            '
  PutDate  : '19941124'    PutTime  : '11184035'
  ApplOriginData : '    '

****   Message      ****

 length - 36 bytes

00000000:  0000 0002 0000 0024 0000 0001 0000 0015 '.......$........'
00000010:  0000 0001 0000 0001 0000 0000 0000 0000 '................'
00000020:  0000 0000                               '....            '


 MQGET of message number 3
****Message descriptor****
```

```
  StrucId  : 'MD  '  Version : 1
  Report   : 0  MsgType : 8
  Expiry   : -1  Feedback : 0
  Encoding : 273  CodedCharSetId : 850
  Format : 'AMQMRESP'
  Priority : 5  Persistence : 1
  MsgId : X'414D5120736E6F6F6B657220202020202ED477D62A9EA100'
  CorrelId : X'000000000000000000000000000000000000000000000000'
  BackoutCount : 0
  ReplyToQ       : '                                                '
  ReplyToQMgr    : 'snooker                                         '
  ** Identity Context
  UserIdentifier : 'trevor      '
  AccountingToken :
   X'0437303730000000000000000000000000000000000000000000000000000000'
  ApplIdentityData : '                                '
  ** Origin Context
  PutApplType    : '6'
  PutApplName    : '                            '
  PutDate  : '19941124'    PutTime  : '11240678'
  ApplOriginData : '    '

****   Message      ****

 length - 188 bytes

00000000:  736E 6F6F 6B65 7220 2020 2020 2020 2020 'snooker         '
00000010:  2020 2020 2020 2020 2020 2020 2020 2020 '                '
00000020:  2020 2020 2020 2020 2020 2020 2020 2020 '                '
00000030:  534E 4F4F 4B45 522E 5749 4748 542E 5443 'SNOOKER.WIGHT.TC'
00000040:  5020 2020 2020 2020 2020 2020 2020 2020 'P               '
00000050:  2020 2020 2020 2020 2020 2020 2020 2020 '                '
00000060:  0000 0001 0000 0024 0000 0001 0000 0015 '.......$........'
00000070:  0000 0001 0000 0001 0000 0000 0000 0000 '................'
00000080:  0000 0002 0000 0004 0000 0028 0000 0DAD '...........(....'
00000090:  0000 0000 0000 0014 534E 4F4F 4B45 522E '........SNOOKER.'
000000A0:  5749 4748 542E 5443 5020 2020 0000 0003 'WIGHT.TCP   ....'
000000B0:  0000 0010 0000 05E7 0000 0001           '............    '

 MQGET of message number 4
****Message descriptor****

  StrucId  : 'MD  '  Version : 1
  Report   : 0  MsgType : 2
  Expiry   : -1  Feedback : 0
  Encoding : 273  CodedCharSetId : 850
  Format : 'MQADMIN '
  Priority : 8  Persistence : 1
  MsgId : X'414D5120736E6F6F6B657220202020202ED477D63826C000'
  CorrelId : X'414D5120736E6F6F6B657220202020202ED477D62A9EA100'
  BackoutCount : 0
  ReplyToQ       : '                                                '
  ReplyToQMgr    : 'snooker                                         '
  ** Identity Context
  UserIdentifier : 'tiger       '
  AccountingToken :
   X'0437303730000000000000000000000000000000000000000000000000000000'
  ApplIdentityData : '                                '
  ** Origin Context
  PutApplType    : '6'
  PutApplName    : '                            '
  PutDate  : '19941124'    PutTime  : '11240694'
  ApplOriginData : '    '

****   Message      ****

 length - 36 bytes
```

```
00000000:  0000 0002 0000 0024 0000 0001 0000 0015 '.......$........'
00000010:  0000 0001 0000 0001 0000 0000 0000 0000 '................'
00000020:  0000 0000                               '....            '
```

```
 No more messages
 MQCLOSE
 MQDISC
```

# Working with alias queues

An alias queue (also known as a queue alias) provides a method of redirecting MQI calls. An alias queue is not a real queue but a definition that resolves to a real queue. The alias queue definition contains a target queue name which is specified by the TARGQ attribute (*BaseQName* in PCF). When an application specifies an alias queue in an MQI call, the queue manager resolves the real queue name at run time.

For example, an application has been developed to put messages on a queue called MY.ALIAS.QUEUE. It specifies the name of this queue when it makes an MQOPEN request and, indirectly, if it puts a message on this queue. The application is not aware that the queue is an alias queue. For each MQI call using this alias, the queue manager resolves the real queue name, which could be either a local queue or a remote queue defined at this queue manager.

By changing the value of the TARGQ attribute, you can redirect MQI calls to another queue, possibly on another queue manager. This is useful for maintenance, migration, and load-balancing.

## Defining an alias queue

The following command creates an alias queue:

```
DEFINE QALIAS (MY.ALIAS.QUEUE) TARGQ (YELLOW.QUEUE)
```

This command redirects MQI calls that specify MY.ALIAS.QUEUE, to the queue YELLOW.QUEUE. The command does not create the target queue; the MQI calls fail if the queue YELLOW.QUEUE does not exist at run time.

If you change the alias definition, you can redirect the MQI calls to another queue. For example:

```
DEFINE QALIAS (MY.ALIAS.QUEUE) TARGQ (MAGENTA.QUEUE) REPLACE
```

This command redirects MQI calls to another queue, MAGENTA.QUEUE.

You can also use alias queues to make a single queue (the target queue) appear to have different attributes for different applications. You do this by defining two aliases, one for each application. Suppose there are two applications:

- Application ALPHA can put messages on YELLOW.QUEUE, but is not allowed to get messages from it.

- Application BETA can get messages from YELLOW.QUEUE, but is not allowed to put messages on it.

You can perform this action using the following commands:

```
* This alias is put enabled and get disabled for application ALPHA

DEFINE QALIAS (ALPHAS.ALIAS.QUEUE) +
       TARGQ (YELLOW.QUEUE) +
       PUT (ENABLED) +
       GET (DISABLED)

* This alias is put disabled and get enabled for application BETA

DEFINE QALIAS (BETAS.ALIAS.QUEUE) +
       TARGQ (YELLOW.QUEUE) +
       PUT (DISABLED) +
       GET (ENABLED)
```

ALPHA uses the queue name ALPHAS.ALIAS.QUEUE in its MQI calls; BETA uses the queue name BETAS.ALIAS.QUEUE. They both access the same queue, but in different ways.

You can use the LIKE and REPLACE attributes when you define queue aliases, in the same way that you use them with local queues.

## Using other commands with queue aliases

You can use the appropriate MQSC commands to display or alter queue alias attributes, or delete the queue alias object. For example:

```
* Display the queue alias' attributes
* ALL = Display all attributes

DISPLAY QUEUE (ALPHAS.ALIAS.QUEUE) ALL


* ALTER the base queue name, to which the alias resolves.
* FORCE = Force the change even if the queue is open.

ALTER QALIAS (ALPHAS.ALIAS.QUEUE) TARGQ(ORANGE.LOCAL.QUEUE) FORCE


* Delete this queue alias, if you can.

DELETE QALIAS (ALPHAS.ALIAS.QUEUE)
```

You cannot delete a queue alias if, for example, an application currently has the queue open or has a queue open that resolves to this queue. See the *MQSeries MQSC Command Reference* for more information about this and other queue alias commands.

# Working with model queues

A queue manager creates a *dynamic queue* if it receives an MQI call from an application specifying a queue name that has been defined as a model queue. The name of the new dynamic queue is generated by the queue manager when the queue is created. A *model queue* is a template that specifies the attributes of any dynamic queues created from it.

Model queues provide a convenient method for applications to create queues as they are required.

## Defining a model queue

You define a model queue with a set of attributes in the same way that you define a local queue. Model queues and local queues have the same set of attributes except that on model queues you can specify whether the dynamic queues created are temporary or permanent. (Permanent queues are maintained across queue manager restarts, temporary ones are not). For example:

```
DEFINE QMODEL (GREEN.MODEL.QUEUE) +
       DESCR('Queue for messages from application X') +
       PUT (DISABLED) +
       GET (ENABLED) +
       NOTRIGGER +
       MSGDLVSQ (FIFO) +
       MAXDEPTH (1000) +
       MAXMSGL (2000) +
       USAGE (NORMAL) +
       DEFTYPE (PERMDYN)
```

This command creates a model queue definition. From the DEFTYPE attribute, the actual queues created from this template are permanent dynamic queues.

**Note:** The attributes not specified are automatically copied from the SYSYTEM.DEFAULT.MODEL.QUEUE default queue.

You can use the LIKE and REPLACE attributes when you define model queues, in the same way that you use them with local queues.

## Using other commands with model queues

You can use the appropriate MQSC commands to display or alter a model queue's attributes, or delete the model queue object. For example:

```
* Display the model queue's attributes
* ALL = Display all attributes

DISPLAY QUEUE (GREEN.MODEL.QUEUE) ALL


* ALTER the model to enable puts on any
* dynamic queue created from this model.

ALTER QMODEL (BLUE.MODEL.QUEUE) PUT(ENABLED)


* Delete this model queue:

DELETE QMODEL (RED.MODEL.QUEUE)
```

## Managing objects for triggering

MQSeries provides a facility for starting an application automatically when certain conditions on a queue are met. One example of the conditions is when the number of messages on a queue reaches a specified number. This facility is called *triggering* and is described in detail in the *MQSeries Application Programming Guide*. This section describes how to set up the required objects to support triggering on MQSeries for Compaq NSK.

## Defining an application queue for triggering

An application queue is a local queue that is used by applications for messaging, through the MQI. Triggering requires a number of queue attributes to be defined on the application queue. Triggering itself is enabled by the *Trigger* attribute (TRIGGER in MQSC).

In this example, a trigger event is to be generated when there are 100 messages of priority five or greater on the local queue MOTOR.INSURANCE.QUEUE, as follows:

```
DEFINE QLOCAL (MOTOR.INSURANCE.QUEUE) +
       PROCESS (MOTOR.INSURANCE.QUOTE.PROCESS) +
       MAXMSGL (2000) +
       DEFPSIST (YES) +
       INITQ (MOTOR.INS.INIT.QUEUE) +
       TRIGGER +
       TRIGTYPE (DEPTH) +
       TRIGDPTH (100)+
       TRIGMPRI (5)
```

Where:

**QLOCAL (MOTOR.INSURANCE.QUEUE)**
Specifies the name of the application queue being defined.

**PROCESS (MOTOR.INSURANCE.QUOTE.PROCESS)**
Specifies the name of the application to be started by a trigger monitor program.

**MAXMSGL (2000)**
Specifies the maximum length of messages on the queue.

### Application queue for triggering

**DEFPSIST (YES)**
Specifies that messages are persistent on this queue.

**INITQ (MOTOR.INS.INIT.QUEUE)**
Is the name of the initiation queue on which the queue manager is to put the trigger message.

**TRIGGER**
Is the trigger attribute value.

**TRIGTYPE (DEPTH)**
Specifies that a trigger event is generated when the number of messages of the required priority (TRIMPRI) reaches the number specified in TRIGDPTH.

**TRIGDPTH (100)**
Specifies the number of messages required to generate a trigger event.

**TRIGMPRI (5)**
Is the priority of messages that are to be counted by the queue manager in deciding whether to generate a trigger event. Only messages with priority 5 or higher are counted.

## Defining an initiation queue

When a trigger event occurs, the queue manager puts a trigger message on the initiation queue specified in the application queue definition. Initiation queues have no special settings, but you can use the following definition of the local queue MOTOR.INS.INIT.QUEUE for guidance:

```
DEFINE QLOCAL(MOTOR.INS.INIT.QUEUE) +
       GET (ENABLED) +
       NOSHARE +
       NOTRIGGER +
       MAXMSGL (2000) +
       MAXDEPTH (10)
```

## Creating a process definition

Use the DEFINE PROCESS command to create a process definition. A process definition associates an application queue with the application that is to process messages from the queue. This is done through the PROCESS attribute on the application queue MOTOR.INSURANCE.QUEUE. The following MQSC command defines the required process, MOTOR.INSURANCE.QUOTE.PROCESS, identified in this example:

```
DEFINE PROCESS (MOTOR.INSURANCE.QUOTE.PROCESS) +
               DESCR ('Insurance request message processing') +
               APPLTYPE (NSK) +
               APPLICID ('$DATA1.TEST.IRMP01') +
               USERDATA ('open, close, 235')
```

Where:

**MOTOR.INSURANCE.QUOTE.PROCESS**
Is the name of the process definition.

**DESCR ('Insurance request message processing')**
Is the descriptive text of the application program to which the definition

relates, following the keyword. This text is displayed when you use the DISPLAY PROCESS command. This can help you to identify what the process does. If you use spaces in the string, you must enclose the string in single quotes.

**APPLTYPE(NSK)**
Is the type of the application that runs on Compaq NSK.

**APPLICID ('$DATA1.TEST.IRMPO1')**
Is the name of the application executable program on the local system.

**USERDATA ('open, close, 235')**
Is user-defined data, which can be used by the application.

## Displaying your process definition

Use the DISPLAY PROCESS command, with the ALL keyword, to examine the results of your definition. For example:

```
DISPLAY PROCESS (MOTOR.INSURANCE.QUOTE.PROCESS) ALL


    24 : DISPLAY PROCESS (MOTOR.INSURANCE.QUOTE.PROCESS) ALL
AMQ8407: Display Process details.
    DESCR (Insurance request message processing)
    APPLICID  ($DATA1.TEST.IRMPO1)
    ENVRDATA ( )
    USERDATA (open, close, 235)
    PROCESS (MOTOR.INSURANCE.QUOTE.PROCESS)
    APPLTYPE (NSK)
```

USERDATA is a string representing the arguments passed to the triggered application. See the sample programs AMQSTRG0 and AMQINQA (in ZMQSSMPL subvolume) for examples of how to write trigger monitors and triggered applications.

You can also use the MQSC ALTER PROCESS to alter an existing process definition and DELETE PROCESS to delete a process definition.

# Chapter 6. Automating administration tasks

This chapter assumes that you have experience of administering MQSeries objects.

There may come a time when you decide that it would be beneficial to your installation to automate some administration and monitoring tasks. You can automate administration tasks for both local and remote queue managers using programmable command format (PCF) commands.

This chapter describes:

- How to use programmable command formats to automate administration tasks in Performing administration using PCF commands.
- How to use the command server in "Managing the command server for remote administration" on page 109.

## PCF commands

The purpose of MQSeries programmable command format (PCF) commands is to allow administration tasks to be programmed into an administration program. In this way you can create queues, process definitions, channels, and namelists, and change queue managers, from a program.

PCF commands cover the same range of functions provided by the MQSC facility.

Therefore, you can 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.

Each PCF command is a data structure that is embedded in the application data part of an MQSeries message. Each command is sent to the target queue manager using the MQI function MQPUT in the same way as any other message. The command server on the queue manager receiving the message interprets it as a command message and runs the command. To get the replies, the application issues an MQGET call and the reply data is returned in another data structure. The application can then process the reply and act accordingly.

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

Briefly, these are some of the things the application programmer must specify to create a PCF command message:

**Message descriptor**

This is a standard MQSeries message descriptor, in which:

Message type (*MsgType*) is MQMT_REQUEST.
Message format (*Format*) is MQFMT_ADMIN.

**Application data**

Contains the PCF message including the PCF header, in which:

The PCF message type (*Type*) specifies MQCFT_COMMAND.

The command identifier specifies the command, for example, *Change Queue* (MQCMD_CHANGE_Q).

© Copyright IBM Corp. 1993, 2001

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

## Attributes in MQSC and PCFs

Object attributes specified in MQSC are shown in this book in uppercase (for example, RQMNAME), although they are not case sensitive. MQSC attribute names are limited to eight characters.

Object attributes in PCF, which are not limited to eight characters, are shown in this book in italics. For example, the PCF equivalent of RQMNAME is `RemoteQMgrName`.

## Escape PCFs

Escape PCFs are PCF commands that contain MQSC commands within the message text. You can use PCFs to send commands to a remote queue manager. For more information about using escape PCFs, see the *MQSeries Programmable System Management* book.

## Using the MQAI to simplify the use of PCFs

The MQAI is an administration interface to MQSeries that is now available on the Compaq NSK platform.

It performs administration tasks on a queue manager through the use of *data bags*. Data bags allow you to handle properties (or parameters) of objects in a way that is easier than using PCFs.

The MQAI can be used:

- **To simplify the use of PCF messages**. The MQAI is an easy way to administer MQSeries; you do not have to write your own PCF messages and this avoids the problems associated with complex data structures.

  To pass parameters in programs that are written using MQI calls, the PCF message must contain the command and details of the string or integer data. To do this, several statements are needed in your program for every structure, and memory space must be allocated. This task is long and laborious.

  On the other hand, programs written using the MQAI pass parameters into the appropriate data bag and only one statement is required for each structure. The use of MQAI data bags removes the need for you to handle arrays and allocate storage, and provides some degree of isolation from the details of the PCF.
- **To handle error conditions more easily**. It is difficult to get return codes back from MQSC commands, but the MQAI makes it easier for the program to handle error conditions.

After you have created and populated your data bag, you can then send an administration command message to the command server of a queue manager, using the mqExecute call, which will wait for any response messages. The mqExecute call handles the exchange with the command server and returns responses in a response bag.

For more information about using the MQAI, see the *MQSeries Administration Interface Programming Guide and Reference*.

For more information about PCFs in general, see the *MQSeries Programmable System Management* book.

# Managing the command server for remote administration

Each queue manager can have a command server associated with it. A command server processes any incoming commands from remote queue managers, or PCF commands from applications. It presents the commands to the queue manager for processing and returns a completion code or operator message depending on the origin of the command.

A command server is mandatory for all administration involving PCFs, the MQAI, and also for remote administration.

**Note:** For remote administration, you must ensure that the target queue manager is running. Otherwise, the messages containing commands cannot leave the queue manager from which they are issued. Instead, these messages are queued in the local transmission queue that serves the remote queue manager. This situation should be avoided, if at all possible.

## Starting the command server

To start the command server use this command:

```
strmqcsv saturn.queue.manager
```

where `saturn.queue.manager` is the queue manager for which the command server is being started.

The command server can also be started in PATHCOM by thawing and starting its serverclass.

## Displaying the status of the command server

For remote administration, ensure that the command server on the target queue manager is running. If it is not running, remote commands cannot be processed. Any messages containing commands are queued in the target queue manager's command queue.

To display the status of the command server for a queue manager, called here `saturn.queue.manager`, the command is:

```
dspmqcsv saturn.queue.manager
```

You must issue this command on the target machine. If the command server is running, the following message is returned:

```
AMQ8027    MQSeries Command Server Status ..: Running
```

## Stopping a command server

To end a command server, the command, using the previous example is:

```
endmqcsv saturn.queue.manager
```

You can stop the command server in two different ways:

- For a controlled stop, use the **endmqcsv** command with the -c flag, which is the default.
- For an immediate stop, use the **endmqcsv** command with the -i flag.

**Note:** Stopping a queue manager also ends the command server associated with it (if one has been started).

# Chapter 7. Administering remote MQSeries objects

This chapter explains how to administer MQSeries objects on another queue manager. It also explains how you can use remote queue objects to control the destination of messages and reply messages.

It contains these sections:
- "Administering a remote queue manager" on page 113
- "Creating a local definition of a remote queue" on page 119
- "Using remote queue definitions as aliases" on page 122

For more information about channels, their attributes, and how to set them up, refer to the *MQSeries Intercommunication* book.

## Channels, clusters and remote queuing

A queue manager communicates with another queue manager by sending a message and, if required, receiving back a response. The receiving queue manager could be:
- On the same machine
- On another machine in the same location or on the other side of the world
- Running on the same platform as the local queue manager
- Running on another platform supported by MQSeries

These messages may originate from:
- User-written application programs that transfer data from one node to another.
- User-written administration applications that use PCFs or the MQAI
- Queue managers sending:
  - Instrumentation event messages to another queue manager.
  - MQSC commands issued from a **runmqsc** command in indirect mode (where the commands are run on another queue manager).

Before a message can be sent to a remote queue manager, the local queue manager needs a mechanism to detect the arrival of messages and transport them consisting:
- Of at least one channel
- A transmission queue
- A message channel agent (MCA)
- A channel listener
- A channel initiator

A channel is a one-way communication link between two queue managers and can carry messages destined for any number of queues at the remote queue manager.

Each end of the channel has a separate definition. For example, if one end is a sender or a server, the other end must be a receiver or a requester. A simple channel consists of a *sender channel definition* at the local queue manager end and a *receiver channel definition* at the remote queue manager end. The two definitions must have the same name and together constitute a single channel.

© Copyright IBM Corp. 1993, 2001

If the remote queue manager is expected to respond to messages sent by the local queue manager, a second channel needs to be set up to send responses back to the local queue manager.

Channels are defined using the MQSC DEFINE CHANNEL command. In this chapter, the examples relating to channels use the default channel attributes unless otherwise specified.

There is a message channel agent (MCA) at each end of a channel which controls the sending and receiving of messages. It is the job of the MCA to take messages from the transmission queue and put them on the communication link between the queue managers. Conversely, it is the job of the Receiving MCA to take messages from the communications link and put them on the target queues.

A transmission queue is a specialized local queue that temporarily holds messages before they are picked up by the MCA and sent to the remote queue manager. You specify the name of the transmission queue on a *remote queue definition*.

"Preparing channels and transmission queues for remote administration" on page 114 shows how to use these definitions to set up remote administration.

For more information about setting up distributed queuing in general, see the *MQSeries Intercommunication* book.

## Remote administration using clusters

In a traditional MQSeries network using distributed queuing, every queue manager is independent. If one queue manager needs to send messages to another queue manager it must have defined a transmission queue, a channel to the remote queue manager, and a remote queue definition for every queue to which it wants to send messages.

A *cluster* is a group of queue managers set up in such a way so that the queue managers can communicate directly with one another over a single network without the need for complex transmission queue, channels, and queue definitions. Clusters can be set up easily, and typically contain queue managers that are logically related in some way and need to share data or applications.

Once a cluster has been created the queue managers within it can communicate with each other *without the need for complicated channel or remote queue definitions*. Even the smallest cluster will reduce system administration overheads.

Establishing a network of queue managers in a cluster involves fewer definitions than establishing a traditional distributed queuing environment. With fewer definitions to make, you can set up or change your network more quickly and easily, and the risk in making an error in your definitions is reduced.

To set up a cluster, you usually need one cluster sender (CLUSSDR) definition and one cluster receiver (CLUSRCVR) definition per queue manager. You do not need any transmission queue definitions or remote queue definitions. The principles of remote administration are the same when used within a cluster, but the definitions themselves are greatly simplified.

For more information about clusters, their attributes, and how to set them up, refer to the *MQSeries Queue Manager Clusters* book.

# Administering a remote queue manager

This section explains how to administer a remote queue manager from a local queue manager. You can implement remote administration from a local node using:

- MQSC commands
- PCF commands

Preparing the queues and channels is essentially the same for both methods. In this book, the examples show MQSC commands, because they are easier to understand. However, you can convert the examples to PCFs if you wish. For more information about writing administration programs using PCFs, see the *MQSeries Programmable System Management* book.

In remote administration you send MQSC commands to a remote queue manager—either interactively or from a text file containing the commands. The remote queue manager may be on the same machine or, more typically, on a different machine. You can remotely administer queue managers in different MQSeries environments, including UNIX®, Compaq NSK, AS/400®, MVS/ESA, Windows® 2000, and OS/390.

To implement remote administration, you must create certain objects. Unless you have specialized requirements, you should find that the default values (for example, for message length) are sufficient.

## Preparing queue managers for remote administration

Figure 37 on page 114 shows the configuration of queue managers and channels that are required for remote administration. `source.queue.manager` is the *source* queue manager from which you can issue MQSC commands and to which the results of these commands (operator messages) are returned, if possible. `target.queue.manager` is the destination queue manager, which processes the commands and generates any operator messages.

**Note:** `source.queue.manager` must be the default queue manager on the machine you are using. For further information on creating a queue manager, see “crtmqm (Create queue manager)” on page 242.

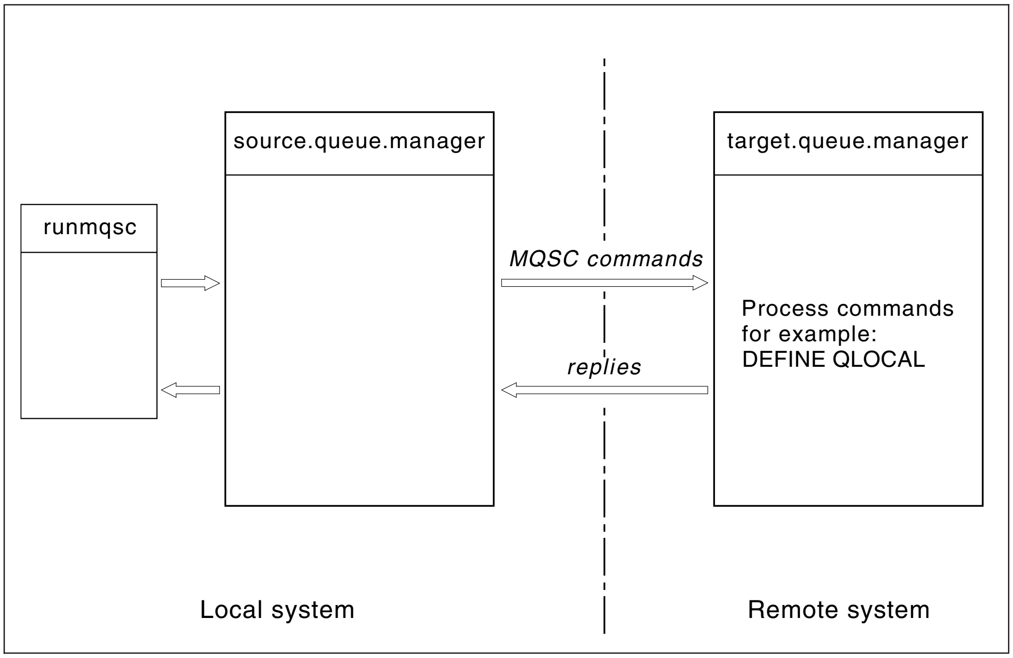


*Figure 37. Remote administration*

On both systems, if you have not already done so, you must:

- Create the queue manager, using the **crtmqm** command.
- Start the queue manager, using the **strmqm** command.

See “Restoring the default and system objects” on page 61 for more information about these steps. You have to run these commands locally or over a network facility, for example Telnet.

On the target queue manager:

- The command queue, SYSTEM.ADMIN.COMMAND.QUEUE, must be present. This queue is created by default when a queue manager is created.
- The command server must be started, using the **strmqcsv** command.

## Preparing channels and transmission queues for remote administration

To run MQSC commands remotely, you must set up two channels, one for each direction, and their associated transmission queues. This example assumes that TCP/IP is being used as the transport type and that you know the TCP/IP address.

The channel `source.to.target` is for sending MQSC commands from the source queue manager to the destination. Its sender is at `source.queue.manager` and its receiver is at queue manager `target.queue.manager`. The channel `target.to.source` is for returning the output from commands and any operator messages that are generated to the source queue manager. You must also define a transmission queue for each sender. This queue is a local queue that is given the name of the receiving queue manager. The XMITQ name must match the remote queue manager name for remote administration to work, unless you are using a queue manager alias. Figure 38 on page 115 summarizes this configuration. However, you should be aware that the SYSTEM.MQSC.REPLY.QUEUE is the name of the model queue that is used by MQSC to develop its own dynamic reply queue. This queue name varies and is internal to MQSC.

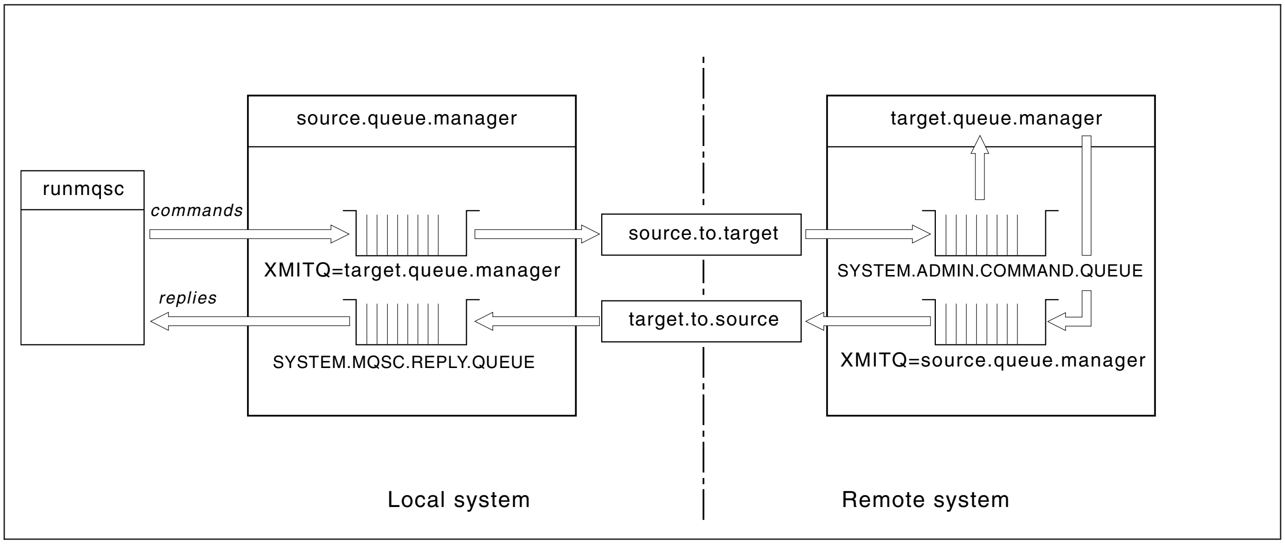


*Figure 38. Setting up channels and queues for remote administration*

See the *MQSeries Intercommunication* book for more information about setting up remote channels.

## Defining channels and transmission queues

On the source queue manager, issue these MQSC commands to define the channels and the transmission queue:

```
* Define the sender channel at the source queue manager
DEFINE CHANNEL ('source.to.target') +
       CHLTYPE(SDR) +
       CONNAME ('198.210.60.37(1414)') +
       XMITQ ('target.queue.manager') +
       TRPTYPE(TCP)

* Define the receiver channel at the source queue manager

DEFINE CHANNEL ('target.to.source') +
       CHLTYPE(RCVR) +
       TRPTYPE(TCP)

* Define the transmission queue on the source

DEFINE QLOCAL ('target.queue.manager') +
       USAGE (XMITQ)
```

Issue these commands on the destination queue manager (`target.queue.manager`), to create the channels and the transmission queue there:

```
* Define the sender channel on the destination queue manager

DEFINE CHANNEL ('target.to.source') +
       CHLTYPE(SDR) +
       CONNAME ('198.210.60.37(1414)') +
       XMITQ ('source.queue.manager') +
       TRPTYPE(TCP)

* Define the receiver channel on the destination queue manager

DEFINE CHANNEL ('source.to.target') +
       CHLTYPE(RCVR) +
       TRPTYPE(TCP)

* Define the transmission queue on the destination queue manager

DEFINE QLOCAL ('source.queue.manager') +
       USAGE (XMITQ)
```

**Note:** The TCP/IP connection names specified for the CONNAME attribute in the sender channel definitions are for illustration only. This is the IP address or network name of the machine at the other end of the connection. Use the values appropriate for your network.

## Starting the channels

The following description assumes that both ends of the channel are running on MQSeries for Compaq NSK. If this is not the case, refer to the relevant documentation for the non-Compaq NSK end of the channel.

To start the two channels, first ensure that the Compaq NSK TCP listener process has been configured for MQSeries on both nodes and are running at both ends of the connections. Then start the channels in **runmqsc**.

- On the source queue manager, enter:

```
start channel ('source.to.target')
```

- On the destination queue manager, enter:

```
start channel ('target.to.source')
```

### Automatic definition of channels

Automatic definition of channels applies only if the target queue manager is running on MQSeries Version 5.1, or later, products. If an inbound attach request is received and an appropriate receiver or server-connection definition cannot be found in the channel definition file (CDF), MQSeries creates a definition automatically and adds it to the CDF. Automatic definitions are based on two default definitions supplied with MQSeries: SYSTEM.AUTO.RECEIVER and SYSTEM.AUTO.SVRCONN.

You enable automatic definition of receiver and server-connection definitions by updating the queue manager object using the MQSC command, ALTER QMGR (or the PCF command Change Queue Manager).

For more information about the automatic creation of channel definitions, see the *MQSeries Intercommunication* book.

For information about the automatic definition of channels for clusters, see the *MQSeries Queue Manager Clusters* book.

## Issuing MQSC commands remotely

The command server *must* be running on the destination queue manager, if it is going to process MQSC commands remotely. (This is not necessary on the source queue manager.)

- On the destination queue manager, type:

```
strmqcsv target.queue.manager
```

- On the source queue manager, you can then run MQSC interactively in queued mode by entering:

```
runmqsc -w 30 target.queue.manager
```

This form of the **runmqsc** command—with the -w flag—runs the MQSC commands in queued mode, where commands are put (in a modified form) on the command-server input queue and executed in order.

When you type in an MQSC command, it is redirected to the remote queue manager, in this case, `target.queue.manager`. The timeout is set to 30 seconds; if a reply is not received within 30 seconds, the following message is generated on the local (source) queue manager:

```
AMQ8416: MQSC timed out waiting for a response from the command server.
```

At the end of the MQSC session, the local queue manager displays any timed-out responses that have arrived. When the MQSC session is finished, any further responses are discarded.

In queued mode, you can also run an MQSC command file on a remote queue manager. For example:

```
runmqsc /IN mycmds, OUT report/ -w 60 target.queue.manager
```

where mycmds is a file containing MQSC commands and report is the report file.

#### Working with queue managers on MVS/ESA

You can issue MQSC commands to an MVS/ESA queue manager from an MQSeries for Compaq NSK queue manager. However, to do this, you must modify the **runmqsc** command and the channel definitions at the sender.

In particular, you add the -x flag to the **runmqsc** command on a Compaq NSK node:

```
runmqsc -w 30 -x QMRI
```

The channel definition is as follows:

```
* Define the sender channel at the source
queue manager on Compaq NSK

  DEFINE CHANNEL ('source.to.target') +
         CHLTYPE(SDR) +
         CONNAME ('198.210.60.37(1414)') +
         XMITQ (QMRI) +
         TRPTYPE(TCP) +
```

You must also define the receiver channel and the transmission queue at the source queue manager as before. Again, this example assumes that TCP/IP is the transmission protocol being used.

### Recommendations for remote queuing

When you are implementing remote queuing:

1. Put the MQSC commands to be run on the remote system in a command file.
2. Verify your MQSC commands locally, by specifying the -v flag on the **runmqsc** command.

   You cannot use **runmqsc** to verify MQSC commands on another queue manager.
3. Check, as far as possible, that the command file runs locally without error.
4. Finally, run the command file against the remote system.

## If you have problems using MQSC remotely

If you have difficulty in running MQSC commands remotely, use the following checklist to see if you have:

- Started the command server on the destination queue manager.
- Defined a valid transmission queue.
- Defined the two ends of the message channels for both:
  - The channel along which the commands are being sent.
  - The channel along which the replies are to be returned.
- Specified the correct connection name (CONNAME) in the channel definition.
- Started the listeners before you started the message channels.

- Checked that the disconnect interval has not expired, for example, if a channel started but then shut down after some time. This is especially important if you start the channels manually.
- Ensured that you are not sending requests from a source queue manager that do not make sense to the target queue manager (for example, request include new parameters.)

See also “Resolving problems with MQSC” on page 91.

# Creating a local definition of a remote queue

You can use a remote queue definition as a local definition of a remote queue. You create a remote queue object on your local queue manager to identify a local queue on another queue manager.

## Understanding how local definitions of remote queues work

An application connects to a local queue manager and then issues an MQOPEN call. In the open call, the queue name specified is that of a remote queue definition on the local queue manager. The remote queue definition supplies the names of the destination queue, the destination queue manager, and optionally, a transmission queue. To put a message on the remote queue, the application issues an MQPUT call, specifying the handle returned from the MQOPEN call. The queue manager appends the remote queue name and the remote queue manager name to a transmission header in the message. This information is used to route the message to its correct destination in the network.

As administrator, you can control the destination of the message by altering the remote queue definition.

### Example

An application is required to put a message on a queue owned by a remote queue manager.

### How it works

The application connects to a queue manager, for example, `saturn.queue.manager`. The destination queue is owned by another queue manager.

On the MQOPEN call, the application specifies these fields in the MQOD:

| Field value | Description |
|---|---|
| *ObjectName*<br>CYAN.REMOTE.QUEUE | Specifies the local name of the remote queue object. This defines the destination queue and the destination queue manager. |
| *ObjectType*<br>(Queue) | Identifies this object as a queue. |
| *ObjectQmgrName*<br>Blank<br>or<br>`saturn.queue.manager` | This field is optional.<br>If blank, the name of the local queue manager is assumed. (This is the queue manager on which the remote queue definition was made and to which the application is connected).<br>If not blank, the name of the local queue manager must be specified. |

### Creating a local definition of remote queue

After this, the application issues an MQPUT call to put a message on to this queue.

On the local queue manager, you can create a local definition of a remote queue using the following MQSC commands:

```
DEFINE QREMOTE ('CYAN.REMOTE.QUEUE') +
       DESCR ('Queue for auto insurance requests from the branches') +
       RNAME ('AUTOMOBILE.INSURANCE.QUOTE.QUEUE') +
       RQMNAME ('jupiter.queue.manager') +
       XMITQ ('INQUOTE.XMIT.QUEUE')
```

Where:

**QREMOTE ('CYAN.REMOTE.QUEUE')**
Is the local name of the remote queue object. This is the name that applications connected to this queue manager must specify in the MQOPEN call to open the queue AUTOMOBILE.INSURANCE.QUOTE.QUEUE on the remote queue manager jupiter.queue.manager.

**DESCR ('Queue for auto insurance requests from the branches')**
Is additional text that describes the use of the queue.

**RNAME ('AUTOMOBILE.INSURANCE.QUOTE.QUEUE')**
Is the name of the destination queue on the remote queue manager. This is the real destination queue for messages that are sent by applications that specify the queue name ’CYAN.REMOTE.QUEUE’. The queue ’AUTOMOBILE.INSURANCE.QUOTE.QUEUE’ must be defined as a local queue on the remote queue manager.

**RQMNAME ('jupiter.queue.manager')**
Is the name of the remote queue manager that owns the destination queue ’AUTOMOBILE.INSURANCE.QUOTE.QUEUE’.

**XMITQ ('INQUOTE.XMIT.QUEUE')**
Is the name of the transmission queue. This is optional; if the name is not specified, a queue with the same name as the remote queue manager is used.

In either case, the appropriate transmission queue must be defined as a local queue with a *Usage* attribute specifying that it is a transmission queue (USAGE(XMITQ) in MQSC).

## An alternative way of putting messages on a remote queue

Using a local definition of a remote queue is not the only way of putting messages on a remote queue. Applications can specify the full queue name, which includes the remote queue manager name, as part of the MQOPEN call. In this case, a local definition of a remote queue is not required. However, this alternative means that applications must either know or have access to the name of the remote queue manager at run time.

## Using other commands with remote queues

You can use the appropriate MQSC commands to display or alter the attributes of a remote queue object, or you can delete the remote queue object. For example:

```
* Display the remote queue's attributes.
* ALL = Display all attributes

DISPLAY QUEUE (CYAN.REMOTE.QUEUE) ALL


* ALTER the remote queue to enable puts.
* This does not affect the destination queue,
* only applications that specify this remote queue.

ALTER QREMOTE (CYAN.REMOTE.QUEUE) PUT(ENABLED)


* Delete this remote queue
* This does not affect the destination queue
* only its local definition

DELETE QREMOTE (CYAN.REMOTE.QUEUE)
```

**Note:** If you delete a remote queue, you only delete the local representation of the remote queue. You do not delete the remote queue itself or any messages on it.

## Creating a transmission queue

A transmission queue is a local queue that is used when a queue manager forwards messages to a remote queue manager through a message channel. The channel provides a one-way link to the remote queue manager. Messages are queued at the transmission queue until the channel can accept them. When you define a channel, you must specify a transmission queue name at the sending end of the message channel.

The *Usage* attribute (USAGE in MQSC) defines whether a queue is a transmission queue or a normal queue.

### Default transmission queues

Optionally, you can specify a transmission queue in a remote queue object, using the *XmitQName* attribute (XMITQ in MQSC). If no transmission queue is defined, a default is used. When applications put messages on a remote queue, if a transmission queue with the same name as the destination queue manager exists, that queue is used. If this queue does not exist, the queue specified by the *DefaultXmitQ* attribute (DEFXMITQ in MQSC) on the local queue manager is used.

For example, the following MQSC command creates a default transmission queue on `source.queue.manager` for messages going to `target.queue.manager`:

```
DEFINE QLOCAL ('target.queue.manager') +
       DESCR ('Default transmission queue for target qm') +
       USAGE (XMITQ)
```

Applications can put messages directly on a transmission queue, or they can be put there indirectly, for example, through a remote queue definition. See also "Creating a local definition of a remote queue" on page 119.

# Using remote queue definitions as aliases

In addition to locating a queue on another queue manager, you can also use a local definition of a remote queue for both:
- Queue manager aliases
- Reply-to queue aliases

Both types of alias are resolved through the local definition of a remote queue.

As usual in remote queuing, the appropriate channels must be set up if the message is to arrive at its destination.

## Queue manager aliases

An alias is the process by which the name of the destination queue manager—as specified in a message—is modified by a queue manager on the message route. Queue manager aliases are important because you can use them to control the destination of messages within a network of queue managers.

You do this by altering the remote queue definition on the queue manager at the point of control. The sending application is not aware that the queue manager name specified is an alias.

For more information about queue manager aliases, see the *MQSeries Intercommunication* book.

## Reply-to queue aliases

Optionally, an application can specify the name of a reply-to queue when it puts a *request message* on a queue. If the application that processes the message extracts the name of the reply-to queue, it knows where to send the *reply message*, if required.

A reply-to queue alias is the process by which a reply-to queue—as specified in a request message—is altered by a queue manager on the message route. The sending application is not aware that the reply-to queue name specified is an alias.

A reply-to queue alias lets you alter the name of the reply-to queue and optionally its queue manager. This in turn lets you control which route is used for reply messages.

For more information about request messages, reply messages, and reply-to queues, see the *MQSeries Application Programming Reference*. For more information about reply-to queue aliases, see the *MQSeries Intercommunication* book.

# Data conversion

Message data in MQSeries-defined formats (also known as built-in formats) can be converted by the queue manager from one coded character set to another, provided that both character sets relate to a single language or a group of similar languages.

For example, conversion between coded character sets whose identifiers (CCSIDs) are 850 and 500 is supported, because both apply to Western European languages.

For EBCDIC new line (NL) character conversions to ASCII, see “What the MQSeries configuration file contains” on page 173.

Supported conversions are defined in Appendix F. ″Code page conversion tables″ in the *MQSeries Application Programming Reference*.

## When a queue manager cannot convert messages in built-in formats

The queue manager cannot automatically convert messages in built-in formats if their CCSIDs represent different national-language groups. For example, conversion between CCSID 850 and CCSID 1025 (which is an EBCDIC coded character set for languages using Cyrillic script) is not supported because many of the characters in one coded character set cannot be represented in the other. If you have a network of queue managers working in different national languages, and data conversion among some of the coded character sets is not supported, you can enable a default conversion. Default data conversion is described in “Default data conversion”.

## File CCSID

The file CCSID specifies any additional code sets and any default data conversion. You can update the information recorded in the CCSID file; you might want to do this if, for example, a future release of your operating system supports additional coded character sets. To specify additional code sets, you need to edit the CCSID file. Guidance on how to do this is provided in the file.

### Default data conversion

To implement default data conversion, you edit the CCSID file to specify a default EBCDIC CCSID and a default ASCII CCSID, and also to specify the defaulting CCSIDs. Instructions for doing this are included in the file.

If you update the CCSID file to implement default data conversion, the queue manager must be restarted before the change can take effect.

The default data-conversion process is as follows:

- If conversion between the source and target CCSIDs is not supported, but the CCSIDs of the source and target environments are either both EBCDIC or both ASCII, the character data is passed to the target application without conversion.
- If one CCSID represents an ASCII coded character set, and the other represents an EBCDIC coded character set, MQSeries converts the data using the default data-conversion CCSIDS defined in the CCSID file.

**Note:** You should try to restrict the characters being converted to those that have the same code values in the coded character set specified for the message and in the default coded character set. If you use only that set of characters that is valid for MQSeries object names you will, in general, satisfy this requirement. Exceptions occur with EBCDIC CCSIDs 290, 930, 1279, and 5026 used in Japan, where the lowercase characters have different codes from those used in other EBCDIC CCSIDs.

## Conversion of messages in user-defined formats

Messages in user-defined formats cannot be converted from one coded character set to another by the queue manager. If data in a user-defined format requires conversion, you must supply a data-conversion exit for each such format. The use of default CCSIDs for converting character data in user-defined formats is not recommended, although it is possible. For more information about converting data in user-defined formats and about writing data conversion exits, see the *MQSeries Application Programming Guide*.

## Changing the queue manager CCSID

You are recommended to stop and restart the queue manager when you change the CCSID of the queue manager, by using the CCSID attribute of the ALTER QMGR command.

This ensures that all running applications, including the command server and channel programs, are stopped and restarted.

This is necessary, because any applications that are running when the queue manager CCSID is changed, continue to use the existing CCSID.

# Chapter 8. Protecting MQSeries objects

This chapter explains the features of security control in MQSeries for Compaq NSK, and describes how you can implement security control.

This chapter contains these sections:

## Why you need to protect MQSeries resources

Because MQSeries queue managers handle the transfer of information that is potentially valuable, you need the safeguard of an authority system. This step ensures that the resources that a queue manager owns and manages are protected from unauthorized access, which could lead to the loss or disclosure of the information. In a secure system, it is essential that none of the following are accessed or changed by any unauthorized user or application:

- Connections to a queue manager.
- Access to MQSeries objects such as queues, clusters, channels, and processes.
- Commands for queue manager administration, including MQSCs and PCF commands.
- Access to MQSeries messages.
- Context information associated with messages.

You should develop your own policy with respect to which users have access to which resources.

## Understanding user IDs in the MQM user group

All queue manager resources run with the group ID MQM.

To be able to access MQSeries for Compaq NSK, your user ID must correspond to an MQSeries principal. Initially, only the user ID that created the queue manager has the MQSeries principal mqm. You must use the **altmqusr** command to create a principal for each user that will access MQSeries. The principal and user must both be unique. To display the principals and their properties for a queue manager, use the **dspmqusr** command.

If your user ID belongs to the MQSeries for Compaq NSK group MQM, and an MQSeries principal has been created for your user ID, you have all authorities to all MQSeries resources. Your user ID *must* belong to the MQM group to be able to use all the MQSeries for Compaq NSK control commands (except **crtmqcvx**). In particular, you need this authority to:

- Use the **runmqsc** utility to run MQSC commands.

© Copyright IBM Corp. 1993, 2001

- Administer authorities on MQSeries for Compaq NSK using the **setmqaut** command.

If you are sending channel commands to queue managers on a remote Compaq NSK system, you must ensure that your user ID is a member of Compaq NSK group MQM on the target system. For a list of PCF and MQSC channel commands, see "Channel command security" on page 135.

It is not essential for your user ID to belong to group MQM for issuing:
- PCF commands—including Escape PCFs—from an administration program
- MQI calls from an application program

**Note:** Authorisations for the mqm principal (and MQM groups) are important because the mqm principal is used by the internal queue manager components themselves to access protected resources. If you remove authorizations for the mqm principal or MQM group from objects within the queue manager, or if you remove the mqm principal itself, you could end up with a queue manager that cannot be administered, or in the worst case, cannot be used at all.

### Getting additional information

For more information about:
- MQSeries for Compaq NSK command sets, see "Chapter 3. Using the MQSeries command sets" on page 27
- MQSeries for Compaq NSK control commands, see "Chapter 17. The MQSeries control commands" on page 227
- PCF commands and Escape PCFs, see the *MQSeries Programmable System Management* book
- MQI calls, see the *MQSeries Application Programming Guide* and *MQSeries Application Programming Reference*

## Understanding the Object Authority Manager (OAM)

By default, access to queue manager resources is controlled through an authorization service installable component. The authorization service component supplied with MQSeries for Compaq NSK is called the OAM and is automatically installed and enabled for each queue manager you create, unless you specify otherwise. In this chapter, the term OAM is used to denote the Object Authority Manager supplied with this product.

The OAM is an *installable component* of the authorization service. Providing the OAM as an installable component gives you the flexibility to:
- Replace the supplied OAM with your own authorization service component using the interface provided.
- Augment the facilities supplied by the OAM with those of your own authorization service component, again using the interface provided.
- Remove or disable the OAM and run with no authorization service at all.

For more information on installable services, see the *MQSeries Programmable System Management* book.

The OAM manages users' authorizations to manipulate MQSeries objects, including queues, process definitions, and channels. It also provides a command interface through which you can grant or revoke access authority to an object for a specific group of users. The decision to allow access to a resource is made by the OAM, and the queue manager follows that decision. If the OAM cannot make a decision, the queue manager prevents access to that resource.

## How the OAM works

The OAM uses the user and group IDs and security features of the Compaq NSK operating system. Users can access queue manager objects only if they have the required authority.

## Managing access through user groups

Managing access permissions to MQSeries resources is based on Compaq NSK *groups*. The OAM maintains authorizations at the group level.

In the command interfaces, MQSeries principals are used rather than user IDs. The reason for this is that authorities granted to a user ID can also be granted to other entities. For example, authorities can be granted to an application program that issues MQI calls, or to an administration program that issues PCF commands. In these cases the principal associated with the program is not necessarily the user ID that was used when the program was started.

Compaq NSK user IDs may have the form `<group>.<name>` where both group and name may be up to 8 characters each, whereas MQSeries principal names can be up to 12 characters. In addition, the period character (.) is illegal in user IDs on some other platforms. In MQSeries for Compaq NSK, the principal database contains mappings of Compaq NSK user IDs to MQSeries principal names of 12 characters or fewer.

### When a user belongs to more than one user group

The authorization that a user has is the union of the authorizations of all the groups to which the user belongs and the default authorization for all users. You can use the control command **setmqaut** to set the authorizations for a specific group.

**Note:** Any changes made using the **setmqaut** command take immediate effect, unless the object is in use. In this case, the change occurs when the object is next opened.

### Group sets and the primary group

Management of access permissions to MQSeries resources is based on Compaq NSK user groups. When SAFEGUARD is running, a Compaq NSK user ID can be associated with more than one group, and therefore the corresponding MQSeries principal is also associated with these groups. The primary group is always the Compaq Administrative Group. Secondary groups are configured by creating SAFEGUARD File Sharing Groups, and associating a Compaq NSK user ID with that File Sharing Group.

The OAM maintains authorizations at the level of groups rather than individual principals. The mapping of principals to group names is carried out within in the OAM using the principal database and the Compaq NSK and SAFEGUARD facilities; OAM operations are carried out at the group level. You can, however, display the authorizations of an individual principal.

## Protecting resources with the OAM

Through OAM you can control:

- Access to MQSeries objects through the MQI. When an application program attempts to access an object, the OAM checks if the user ID making the request has the authorization (through its user group) for the operation requested.

  In particular, this means that queues, and the messages on queues, can be protected from unauthorized access.
- Permission to use MQSC commands; only members of user group mqm, or those authorized via **setmqaut**, can execute queue manager administration commands, for example, to create a queue.
- Permission to use control commands; only members of user group mqm can execute control commands, for example, creating a queue manager or starting a command server.
- Permission to use PCF commands.

Different groups of users can be granted different kinds of access authority to the same object. For example, for a specific queue, one group might be allowed to perform both put and get operations; another group can only be allowed to browse the queue (MQGET with browse option). Similarly, some groups might have get and put authority to a queue, but are not allowed to alter or delete the queue.

## Using groups for authorizations

Using groups rather than individual principals for authorization reduces the amount of administration required. Typically, a particular kind of access is required by more than one principal. For example, you might define a group consisting of end users who want to run a particular application. New users can be given access by adding the appropriate group to their Compaq NSK user ID. Unless MQSeries is installed on a system using SAFEGUARD to create data sharing groups, each user ID can be associated with a single, primary group only.

Without SAFEGUARD there is a limit of 255 principals per group. With the use of SAFEGUARD file sharing, the limit is 65535 principals per group. Using SAFEGUARD file sharing also allows a principal to have its own authorizations. For example if the SAFEGUARD administrative group is not given any authorization then each file sharing member belonging to the group can have its own authority (Principal A, Group MQSEC member TESTSEC1 has PUT only, Principal B Group MQSEC member TESTSEC2 has GET only).

Without SAFEGUARD two groups (two Compaq NSK Userids in different groups, for example MQ.TEST (connect/put) and MQ1.TEST (connect/get)) would need to be created to accomplish this because authorizations for the group are combined with the principal (Compaq NSK userid MQM.MANAGER group MQM (connect) principal mqm and Compaq NSK user ID MQM.APPS group MQM principal apps (get/put)). The authority set for the group (connect for mqm) and that of the principal (get/put for apps) are added together (connect/get/put). Validation of the users authority is then done on the combined data. If an application attempts an MQGET while logged on using the user ID MQ.TEST, the operation is rejected with a MQRC_NOT_AUTHORIZED (2035). Logging on as MQ1.TEST would pass. Any user logged on a Compaq user ID in the MQM group would pass. Once a principal is given authority, all principals in the group are granted the same authority.

```
Principal  Compaq Userid   Group          GroupType              Security

mqm        MQM.MANAGER      MQM                                   connect
apps       MQM.APPS         MQM                                   get/put
inq        MQM.INQ          MQM                                   inq
mq         MQ.TEST          MQ                                    connect/put
mq1        MQ1.TEST         MQ1                                   connect/get
A          MQSEC.FRED       MQSEC       safeguard admin           none
                            TESTSEC1    safeguard file sharing    connect/put
B          MQSEC.JOE        MQSEC       safeguard admin           none
                            TESTSEC2    safeguard file sharing    connect/get
C          MQSEC.FRANK      MQSEC       safeguard admin           none
                            TESTSEC3    safeguard file sharing    none
```

*Figure 39. Using groups with SAFEGUARD to provide authorization*

In Figure 39 any logged on user for group MQM would have connect, get, put and inq authority. User MQ.TEST connect and put while MQ1.TEST connect and get. Users in the MQSEC group would have the authority of the safeguard file sharing member that has been granted.

You should keep the number of groups as small as possible. For example, you can divide users into one group for application users and one for administrators.

## Disabling the Object Authority Manager (OAM)

By default, the OAM is enabled. You can disable the OAM by setting the Compaq NSK environment variable MQSNOAUT before the queue manager is created, as follows:

```
PARAM MQSNOAUT 1
```

However, if you disable the OAM for a queue manager, you cannot restart the OAM later. You might want to have the OAM enabled and ensure that all users and applications have access through an appropriate user ID. You can also disable the OAM for testing purposes only either by removing the authorization service stanza in the queue manager configuration file (QMINI) or by setting `MQAUTH` off in the `Authority` stanza of QMINI, as described in “Queue manager configuration file (QMINI)” on page 174.

**Note:** Specifying `PARAM MQSNOAUT 0` does not enable the OAM. The environment variable must not exist in the environment if the OAM is to be re-enabled.

# Using the Object Authority Manager (OAM) commands

The OAM provides a command interface for granting and revoking authority. Before you can use these commands, you must be authorized—your user ID must belong to the Compaq NSK MQM group. (This group should have been set up before you installed MQSeries for Compaq NSK.)

If your user ID is a member of group MQM, you have a ‘super user’ authority to the queue manager. You are now authorized to issue any MQI request or control command from your user ID.

The OAM provides four commands that you can invoke from TACL to manage the authorizations of users. These are:

- **altmqusr** (Create, remove, or alter an MQSeries principal)
- **dspmqusr** (Display principal)
- **setmqaut** (Set or reset authority)
- **dspmqaut** (Display authority)

Authority checking occurs in the following calls: MQCONN, MQOPEN, MQPUT1, and MQCLOSE. Authority checking is only performed at the first instance of any of these calls, and authority is not amended until you reset (that is, close and reopen) the object. Therefore, any changes made to the authority of an object using **setmqaut** do not take effect until you reset the object.

## What to specify when using the OAM commands

The OAM commands apply to the specified queue manager; if you do not specify a queue manager, the default queue manager is used. On these commands, you must specify the object uniquely, that is, you must specify the object name and its type. You also have to specify the user or group name to which the authority applies.

### Authorization lists

You specify a list of authorizations with setmqaut command. This is a quick way of specifying whether authorization is to be granted or revoked, and which resources in which the authorization applies. Each authorization in the list is specified as a lowercase keyword, prefixed with a plus sign (+) or a minus sign (-). You can use a plus sign to add the specified authorization or a minus sign to remove the authorization. You can specify any number of authorizations in a single command. For example:

```
+browse -get +put
```

## Using the altmqusr command

Provided you have the required authorization, you can use the **altmqusr** command to create an MQSeries principal and associate it with a Compaq NSK user ID (or SAFEGUARD alias). The following example shows how the **altmqusr** command is used:

```
altmqusr -m saturn.queue.manager -p MQPRINCIPAL -u MQM.MQUSER
```

In this example:

| This term... | Specifies the... |
|---|---|
| saturn.queue.manager | Queue manager name |
| MQPRINCIPAL | Principal name to be created |
| MQM.MQUSER | Compaq NSK user ID |

See “altmqusr (Alter MQSeries user information)” on page 234 for a description of this command.

The **altmqusr** command can also be used to remove a principal (and thereby revoke all access rights to MQSeries). For example:

```
altmqusr -m saturn.queue.manager -p MQPRINCIPAL -remove
```

In this example:

| This term... | Specifies the... |
|---|---|
| saturn.queue.manager | Queue manager name |
| MQPRINCIPAL | Principal name to be removed |
| -remove | Instruction to delete the principal |

## Using the dspmqusr command

You can display the contents of the principal database, in addition to the Compaq NSK administrative and file-sharing groups that the user ID corresponding to each MQSeries principal belongs to, using the **dspmqusr** command. The -p parameter restricts the information displayed to the specified principal. For example:

```
dspmqusr -m saturn.queue.manager -p MQPRINCIPAL
```

In this example:

| This term... | Specifies the... |
|---|---|
| saturn.queue.manager | Queue manager name |
| MQPRINCIPAL | Principal name to be displayed |

See “dspmqusr (Display MQSeries user information)” on page 258 for a description of this command.

## Using the setmqaut command

Provided you have the required authorization, you can use the **setmqaut** command to grant or revoke authorization of a principal or user group to access a particular object. The following example shows how the **setmqaut** command is used:

```
setmqaut -m saturn.queue.manager -t queue -n RED.LOCAL.QUEUE -g GroupA +browse -get +put
```

In this example:

| This term.... | Specifies the.... |
|---|---|
| saturn.queue.manager | Queue manager name |
| queue | Object type |
| RED.LOCAL.QUEUE | Object name |
| GroupA | ID of the group to be given the authorizations |
| +browse -get +put | Authorization list for the specified queue. There must be no spaces between the ″+″ or ″-″ signs and the keyword. |

The authorization list specifies the authorizations to be given, where:

| This term... | Specifies... |
|---|---|
| +browse | Add authorization to browse (MQGET with browse option) |
| -get | Remove authorization to get (MQGET) messages from the queue. |
| +put | Add authorization to put (MQPUT) messages on the queue. |

Applications started with user IDs that belong to Compaq NSK user group GroupA have these authorizations.

The following command revokes put authority on the queue MyQueue to groups GroupA and GroupB.

```
setmqaut -m saturn.queue.manager -t queue -n MyQueue -g GroupA -g GroupB -put
```

For a formal definition of the command and its syntax, see “setmqaut (Set/reset authority)” on page 277.

### Authority commands and installable services

The **setmqaut** command takes an additional parameter that specifies the name of the authorization service component to which the update applies. You must specify this parameter if you have multiple authorization components running at the same time. By default, this is not the case. If the parameter is omitted, the update is made to the first authorization component it finds, if one exists. By default, this is the supplied OAM.

## Access authorizations

Authorizations defined by the authorization list associated with the **setmqaut** command can be categorized as follows:

- Authorizations related to MQI calls
- Authorization related administration commands
- Context authorizations
- General authorizations, that is, for MQI calls, for commands, or both

Each authorization is specified by a keyword used with the setmqaut and dspmqaut commands. These are described in “setmqaut (Set/reset authority)” on page 277.

## Display authority command

You can use the command **dspmqaut** to view the authorizations that a specific principal or group has for a particular object. The flags have the same meaning as those in the **setmqaut** command. Authorization can be displayed for only one group or principal at a time. See “dspmqaut (Display authority)” on page 248 for a formal specification of this command.

For example, the following command displays the authorizations that the group GpAdmin has to a process definition named Annuities on queue manager QueueMan1.

```
dspmqaut -m QueueMan1 -t process -n Annuities -g GpAdmin
```

The keywords displayed as a result of this command identify the authorizations that are active.

## Object Authority Manager (OAM) guidelines

Some operations are particularly sensitive and should be limited to privileged users. For example:

- Creating, deleting, starting, and stopping queue managers
- Accessing certain special queues, such as transmission queues or the command queue SYSTEM.ADMIN.COMMAND.QUEUE
- Programs that use full MQI context options
- Creating and copying application queues

### User IDs

The special group called MQM that you create is intended for use by product administrators only. It should never be available to nonprivileged users.

### Queue manager volumes

The volume containing queues and other queue manager data is private to the product. Objects in this directory have Compaq NSK user authorizations that relate to their OAM authorizations. Standard Compaq NSK commands cannot be used to grant or revoke authorizations to MQI resources because:

- MQSeries objects are not necessarily the same as the corresponding system object name. See “Volume structure” on page 56 for more information about this.
- MQSeries objects do not necessarily map to the object’s NSK security settings.

### Queues

The authority to access a dynamic queue is based on—but not necessarily the same as—that of the model queue from which it is derived.

For alias queues and remote queues, the authorization is that of the object itself, not the queue to which the alias or remote queue resolves. It is, therefore, possible to authorize a principal to access an alias queue that resolves to a local queue to which the principal has no access permissions.

You should limit the authority to create queues to privileged users. If you do not limit this authority, users can bypass the normal access control by creating an alias.

### Alternate user authority

Alternate user authority controls whether one user ID can use the authority of another user ID when accessing an MQSeries object. This method is essential when a server receives requests from a program and the server needs to ensure that the program has the required authority for the request. The server can have the required authority, but it needs to know whether the program has the authority for the actions it has requested.

For example:

- A server program running under user ID PAYSERV retrieves a request message from a queue that was put on the queue by user ID USER1.

- When the server program gets the request message, it processes the request and puts the reply back into the reply-to queue specified with the request message.
- Instead of using its own user ID (PAYSERV) to authorize opening the reply-to queue, the server can specify some other user ID, in this case, USER1. In this example, you can use alternate user authority to control whether PAYSERV is allowed to specify USER1 as an alternate user ID when it opens the reply-to queue.

The alternate user ID is specified on the *AlternateUserId* field of the object descriptor.

Both the user ID and the alternate user IDs must be specified as principals corresponding to entries in the principal database associated with a Compaq NSK user ID for authorization to be granted.

**Note:** You can use alternate user IDs on any MQSeries object. Use of an alternate user ID does not affect the user ID used by any other resource managers.

## Context authority

Context is information that applies to a particular message and is contained in the message descriptor, MQMD, which is part of the message. The context information comes in two sections:

**Identity section**
This part specifies who the message came from. It consists of the following fields:

- *UserIdentifier*
- *AccountingToken*
- *ApplIdentityData*

**Origin section**
This section specifies where the message came from, and when it was put onto the queue. It consists of the following fields:

- *PutApplType*
- *PutApplName*
- *PutDate*
- *PutTime*
- *ApplOriginData*

Applications can specify the context data when either an MQOPEN or an MQPUT call is made. This data can be generated by the application, it can be passed on from another message, or it can be generated by the queue manager by default. For example, context data can be used by server programs to check the identity of the requester, testing whether the message came from an application, running under an authorized user ID.

A server program can use the *UserIdentifier* to determine the user ID of an alternate user. The *UserIdentifier* must be specified as a principal corresponding to an entry in the principal database.

You use context authorization to control whether the user can specify any of the context options on any MQOPEN or MQPUT1 call. For information about the context options, see the *MQSeries Application Programming Guide*. For descriptions of the message descriptor fields relating to context, see the *MQSeries Application Programming Reference* book.

## Remote security considerations

For remote security, you should consider:

**Put authority**
For security across queue managers you can specify the put authority that is used when a channel receives a message sent from another queue manager.

Specify the channel attribute PUTAUT as follows:

**DEF** Default user ID. The user ID that the message channel agent is running under.

**CTX** The user ID in the message context.

In both cases, the user ID must be specified as a principal corresponding to an entry in the principal database.

**Transmission queues**
Queue managers automatically put remote messages on a transmission queue; no special authority is required. However, putting a message directly on a transmission queue requires special authorization; see Table 3 on page 137.

**Channel exits**
Channel exits can be used for added security.

For more information, see the *MQSeries Intercommunication* book.

## Channel command security

Channel commands can be issued as PCF commands, through the MQAI, MQSC commands, and control commands.

### PCF commands

You can issue PCF channel commands by sending a PCF message to the SYSTEM.ADMIN.COMMAND.QUEUE on a remote Compaq NSK system. The user ID as specified in the message descriptor of the PCF message must be specified as a principal corresponding to an entry in the principal database associated with a Compaq NSK user ID belonging to the mqm group on the target system. These commands are:

- ChangeChannel
- CopyChannel
- CreateChannel
- DeleteChannel
- PingChannel
- ResetChannel
- StartChannel
- StopChannel
- ResolveChannel

See the *MQSeries Programmable System Management* book for the PCF security requirements.

### MQSC channel commands

You can issue MQSC channel commands to a remote Compaq NSK system either by sending the command directly in a PCF escape message or by issuing the command using **runmqsc** in indirect mode. The user ID as specified in the message descriptor of the PCF message must be specified as a principal corresponding to an entry in the principal database associated with a Compaq NSK user ID belonging to the mqm group on the target system. (PCF commands are implicit in MQSC commands issued from **runmqsc** in indirect mode.) These commands are:

- ALTER CHANNEL
- DEFINE CHANNEL
- DELETE CHANNEL
- PING CHANNEL
- RESET CHANNEL
- START CHANNEL
- START CHINIT
- STOP CHANNEL
- RESOLVE CHANNEL

For MQSC commands issued from the **runmqsc** command, the user ID in the PCF message is normally that of the current user.

# Understanding the authorization specification tables

The authorization specification tables starting on page 137 define precisely how the authorizations work and the restrictions that apply. The tables apply to these situations:

- Applications that issue MQI calls.
- Administration programs that issue MQSC commands as escape PCFs.
- Administration programs that issue PCF commands.

In this section, the information is presented as a set of tables that specify the following:

| | |
|---|---|
| **Action to be performed** | MQI option, MQSC command, or PCF command. |
| **Access control object** | Queue, process, or queue manager. |
| **Authorization required** | Expressed as an 'MQZAO_' constant. |

In the tables, the constants prefixed by MQZAO_ correspond to the keywords in the authorization list for the **setmqaut** command for the particular entity. For example, MQZAO_BROWSE corresponds to the keyword `+browse`; similarly, the keyword MQZAO_SET_ALL_CONTEXT corresponds to the keyword `+setall` and so on. These constants are defined in the header file CMQZCH in subvolume ZMQSLIB, which is supplied with the product.

## MQI authorizations

An application is allowed to issue certain MQI calls and options only if the user identifier under which it is running (or whose authorizations it is able to assume) has been granted the relevant authorization.

Four MQI calls may require authorization checks: MQCONN, MQOPEN, MQPUT1, and MQCLOSE.

For MQOPEN and MQPUT1, the authority check is made on the name of the object being opened, and not on the name, or names, resulting after a name has been resolved. For example, an application may be granted authority to open an alias queue without having authority to open the base queue to which the alias resolves. The rule is that the check is carried out on the first definition encountered during the process of name resolution that is not a queue-manager alias, unless the queue-manager alias definition is opened directly; that is, its name appears in the *ObjectName* field of the object descriptor. Authority is always needed for the particular object being opened; in some cases additional queue-independent authority—which is obtained through an authorization for the queue-manager object—is required.

Table 3 summarizes the authorizations needed for each call.

*Table 3. Security authorization needed for MQI calls*

| Authorization required for: | Queue object | Process object | Queue manager | Namelists |
|---|---|---|---|---|
| **MQCONN option** | Not applicable | Not applicable | MQZAO_ CONNECT | Not applicable |
| **MQOPEN Option** | | | | |
| MQOO_INQUIRE | MQZAO_INQUIRE (2) | MQZAO_INQUIRE (2) | MQZAO_INQUIRE (2) | MQZAO_INQUIRE (2) |
| MQOO_BROWSE | MQOO_BROWSE | Not applicable | No check | Not applicable |
| MQOO_INPUT_* | MQZAO_INPUT | Not applicable | No check | Not applicable |
| MQOO_SAVE_ ALL_CONTEXT (3) | MQZAO_INPUT | Not applicable | No check | Not applicable |
| MQOO_OUTPUT (Normal queue) (4) | MQOO_OUTPUT | Not applicable | No check | Not applicable |
| MQOO_PASS_ IDENTITY_CONTEXT (5) | MQZAO_PASS_ IDENTITY_ CONTEXT | Not applicable | No check | Not applicable |
| MQOO_PASS_ ALL_CONTEXT (5, 6) | MQZAO_PASS _ALL_CONTEXT | Not applicable | No check | Not applicable |
| MQOO_SET_ IDENTITY_CONTEXT (5, 6) | MQZAO_SET_ IDENTITY_ CONTEXT | Not applicable | MQZAO_SET_ IDENTITY_ CONTEXT | Not applicable |
| MQOO_SET_ ALL_CONTEXT (5, 8) | MQZAO_SET_ ALL_CONTEXT | Not applicable | MQZAO_SET_ ALL_CONTEXT (7) | Not applicable |
| MQOO_OUTPUT (Transmission queue) (9) | MQZAO_SET_ ALL_CONTEXT | Not applicable | MQZAO_SET_ ALL_CONTEXT (7) | Not applicable |
| MQOO_SET | MQZAO_SET | Not applicable | No check | Not applicable |
| MQOO_ALTERNATE _USER _AUTHORITY | (10) | (10) | MQZAO_ ALTERNATE_ USER_ AUTHORITY (10, 11) | Not applicable |
| **MQPUT1 Option** | | | | |
| MQPMO_PASS_ IDENTITY_CONTEXT | MQZAO_PASS_ IDENTITY_ CONTEXT (12) | Not applicable | No check | Not applicable |
| MQPMO_PASS_ ALL_CONTEXT | MQZAO_PASS_ ALL_CONTEXT (12) | Not applicable | No check | Not applicable |
| MQPMO_SET_ IDENTITY_CONTEXT | MQZAO_SET_ IDENTITY_ CONTEXT (12) | Not applicable | MQZAO_SET_ IDENTITY_ CONTEXT (7) | Not applicable |
| MQPMO_SET_ ALL_CONTEXT | MQZAO_SET_ ALL_CONTEXT (12) | Not applicable | MQZAO_SET_ ALL_CONTEXT (7) | Not applicable |
| (Transmission queue) (9) | MQZAO_SET_ ALL_CONTEXT | Not applicable | MQZAO_SET_ ALL_CONTEXT (7) | Not applicable |

Table 3. Security authorization needed for MQI calls (continued)

| Authorization required for: | Queue object | Process object | Queue manager | Namelists |
|---|---|---|---|---|
| MQPMO_ ALTERNATE _ USER_ AUTHORITY | (13) | Not applicable | MQZAO_ ALTERNATE_ USER_ AUTHORITY (11) | Not applicable |
| **MQCLOSE Option** | | | | |
| MQCO_DELETE | MQZAO_DELETE (14) | Not applicable | Not applicable | Not applicable |
| MQCO_DELETE_ PURGE | MQZAO_DELETE (14) | Not applicable | Not applicable | Not applicable |

**Specific notes:**

1. If a model queue is being opened:
   - MQZAO_DISPLAY authority is needed for the model queue, in addition to whatever other authorities (also for the model queue) are required for the open options specified.
   - MQZAO_CREATE authority is not needed to create the dynamic queue.
   - The user identifier used to open the model queue is automatically granted all of the queue-specific authorities (equivalent to MQZAO_ALL) for the dynamic queue created.
2. Either the queue, process, namelist or queue manager object is checked, depending on the type of object being opened.
3. MQOO_INPUT_* must also be specified. This is valid for a local, model, or alias queue.
4. This check is performed for all output cases, except the case specified in note 9.
5. MQOO_OUTPUT must also be specified.
6. MQOO_PASS_IDENTITY_CONTEXT is also implied by this option.
7. This authority is required for both the queue manager object and the particular queue.
8. MQOO_PASS_IDENTITY_CONTEXT, MQOO_PASS_ALL_CONTEXT, and MQOO_SET_IDENTITY_CONTEXT are also implied by this option.
9. This check is performed for a local or model queue that has a *Usage* queue attribute of MQUS_TRANSMISSION, and is being opened directly for output. It does not apply if a remote queue is being opened (either by specifying the names of the remote queue manager and remote queue, or by specifying the name of a local definition of the remote queue).
10. At least one of MQOO_INQUIRE (for any object type), or (for queues) MQOO_BROWSE, MQOO_INPUT_*, MQOO_OUTPUT, or MQOO_SET must also be specified. The check carried out is as for the other options specified, using the supplied alternate user identifier for the specific-named object authority, and the current application authority for the MQZAO_ALTERNATE_USER_IDENTIFIER check.
11. This authorization allows any *AlternateUserId* to be specified.
12. An MQZAO_OUTPUT check is also carried out, if the queue does not have a *Usage* queue attribute of MQUS_TRANSMISSION.
13. The check carried out is as for the other options specified, using the supplied alternate user identifier for the specific-named queue authority, and the

current application authority for the MQZAO_ALTERNATE_USER_IDENTIFIER check.

14. The check is carried out only if both of the following are true:
    - A permanent dynamic queue is being closed and deleted.
    - The queue was not created by the MQOPEN which returned the object handle being used.

    Otherwise, there is no check.

**General notes:**

1. The special authorization MQZAO_ALL_MQI includes all of the following that are relevant to the object type:
   - MQZAO_CONNECT
   - MQZAO_INQUIRE
   - MQZAO_SET
   - MQZAO_BROWSE
   - MQZAO_INPUT
   - MQZAO_OUTPUT
   - MQZAO_PASS_IDENTITY_CONTEXT
   - MQZAO_PASS_ALL_CONTEXT
   - MQZAO_SET_IDENTITY_CONTEXT
   - MQZAO_SET_ALL_CONTEXT
   - MQZAO_ALTERNATE_USER_AUTHORITY
2. MQZAO_DELETE (see note 14 on page 139) and MQZAO_DISPLAY are classed as administration authorizations. They are not therefore included in MQZAO_ALL_MQI.
3. ‘No check’ means that no authorization checking is carried out.
4. ‘Not applicable’ means that authorization checking is not relevant to this operation. For example, you cannot issue an MQPUT call to a process object.

## Administration authorizations

These authorizations allow a user to issue administration commands. This can be an MQSC command as an escape PCF message or as a PCF command itself. These methods allow a program to send an administration command as a message to a queue manager, for execution on behalf of that user.

## Authorizations for MQSC commands in escape PCFs

Table 4 summarizes the authorizations needed for each MQSC command that is contained in Escape PCF.

*Table 4. MQSC commands and security authorization needed*

| (2) Authorization required for: | Queue object | Process object | Queue manager object | Namelists |
|---|---|---|---|---|
| ALTER object | MQZAO_CHANGE | MQZAO_CHANGE | MQZAO_CHANGE | MQZAO_CHANGE |
| CLEAR QLOCAL | MQZAO_CLEAR | Not applicable | Not applicable | Not applicable |
| DEFINE object NOREPLACE (3) | MQZAO_CREATE (4) | MQZAO_CREATE (4) | Not applicable | MQZAO_CREATE (4) |
| DEFINE object REPLACE (3, 5) | MQZAO_CHANGE | MQZAO_CHANGE | Not applicable | MQZAO_CHANGE |
| DELETE object | MQZAO_DELETE | MQZAO_DELETE | Not applicable | MQZAO_DELETE |
| DISPLAY object | MQZAO_DISPLAY | MQZAO_DISPLAY | MQZAO_DISPLAY | MQZAO_DISPLAY |

**Specific notes:**

1. The user identifier, under which the program (for example, **runmqsc**) which submits the command is running, must also have MQZAO_CONNECT authority to the queue manager.
2. Either the queue, process, namelist or queue manager object is checked, depending on the type of object.
3. For DEFINE commands, MQZAO_DISPLAY authority is also needed for the LIKE object if one is specified, or on the appropriate SYSTEM.DEFAULT.xxx object if LIKE is omitted.
4. The MQZAO_CREATE authority is not specific to a particular object or object type. Create authority is granted for all objects, for a specified queue manager, by specifying an object type of QMGR on the SETMQAUT command.
5. This applies if the object to be replaced does in fact already exist. If it does not, the check is as for DEFINE object NOREPLACE.

**General notes:**

1. To perform any PCF command, you must have DISPLAY authority on the queue manager.
2. The authority to execute an escape PCF depends on the MQSC command within the text of the escape PCF message.
3. 'Not applicable' means that authorization checking is not relevant to this operation. For example, you cannot issue a CLEAR QLOCAL on a queue manager object.

## Authorizations for PCF commands

Table 5 summarizes the authorizations needed for each PCF command.

*Table 5. PCF commands and security authorization needed*

| (2) Authorization required for: | Queue object | Process object | Queue manager object | Namelists |
|---|---|---|---|---|
| Change object | MQZAO_CHANGE | MQZAO_CHANGE | MQZAO_CHANGE | MQZAO_CHANGE |
| Clear Queue | MQZAO_CLEAR | Not applicable | Not applicable | Not applicable |
| Copy object (without replace) (3) | MQZAO_CREATE (4) | MQZAO_CREATE (4) | Not applicable | MQZAO_CREATE (4) |
| Copy object (with replace) (3, 6) | MQZAO_CHANGE | MQZAO_CHANGE | Not applicable | MQZAO_CHANGE |
| Create object (without replace) (5) | MQZAO_CREATE (4) | MQZAO_CREATE (4) | Not applicable | MQZAO_CREATE (4) |
| Create object (with replace) (5, 6) | MQZAO_CHANGE | MQZAO_CHANGE | Not applicable | MQZAO_CHANGE |
| Delete object | MQZAO_DELETE | MQZAO_DELETE | Not applicable | MQZAO_DELETE |
| Inquire object | MQZAO_DISPLAY | MQZAO_DISPLAY | MQZAO_DISPLAY | MQZAO_DISPLAY |
| Inquire object names | No check | No check | No check | No check |
| Reset queue statistics | MQZAO_DISPLAY and MQZAO_CHANGE | Not applicable | Not applicable | Not applicable |

**Specific notes:**

1. The user identifier under which the program submitting the command is running must also have authority to connect to its local queue manager, and to open the administration command queue for output.

2. Either the queue, process, or queue-manager object is checked, depending on the type of object.
3. For Copy commands, MQZAO_DISPLAY authority is also needed for the From object.
4. The MQZAO_CREATE authority is not specific to a particular object or object type. Create authority is granted for all objects, for a specified queue manager, by specifying an object type of QMGR on the SETMQAUT command.
5. For Create commands, MQZAO_DISPLAY authority is also needed for the appropriate SYSTEM.DEFAULT.* object.
6. This applies if the object to be replaced already exists. If it does not, the check is as for Copy or Create without replace.

**General notes:**

1. To perform any PCF command, you must have DISPLAY authority on the queue manager.
2. The special authorization MQZAO_ALL_ADMIN includes all of the following that are relevant to the object type:
   - MQZAO_CHANGE
   - MQZAO_CLEAR
   - MQZAO_DELETE
   - MQZAO_DISPLAY

   MQZAO_CREATE is not included, because it is not specific to a particular object or object type.
3. 'No check' means that no authorization checking is carried out.
4. 'Not applicable' means that authorization checking is not relevant to this operation. For example, you cannot use a Clear Queue command on a process object.

## Understanding authorization files

For MQSeries for Compaq NSK, V5.1, all authorization information is stored in the following TM/MP audited files in location $VOL.<QMgrSubVol>D:

| | |
|---|---|
| **OAMDB** | The OAM Database |
| **PRIDB** | The principal database |
| **PRIDBA** | The principal database alternate key file |

### The principal database

Each record in the principal database maps a Compaq NSK user ID to a principal name. The principal database is an ENSCRIBE key-sequenced file that provides a mapping between the OAM principals and Compaq NSK logon IDs.

| | |
|---|---|
| **OAM Principal** | MQPRINCIPAL |
| **Compaq logon ID** | `0x2CFF` |

The primary key is the OAM principal (12 characters). The alternate key is the Compaq logon ID (a 2-byte short integer). The OAM principal is always case sensitive. The bytes of the logon ID field are `<group>.<user>`; the example above is for Compaq NSK ID (44,255).

### The OAM Database

Each record in the new OAM authorizations database refers to a specific queue manager object, or class of object. The primary key is the object name plus object type. The records are of variable length, and the record layout is as follows:

## Authorization files

| Object Name | Type | #Auth Entries | Auth Entries |
|---|---|---|---|
| QUEUE.AUTH | 1 | 2 | PAYROLL 0x00000004, ADMIN 0xFFFFFFFF .... |

The Object Name field is the full 48-character, blank-filled object name. The type field (4 bytes) differentiates between the types of MQSeries object, and the classes of object required by the OAM.

The Type field may take the following values:

| | |
|---|---|
| **1** | Queue name |
| **2** | Process name |
| **4** | Queue manager name |
| **128** | Class |

The #Auth Entries field (4 bytes) specifies the number of individual authorizations in the Auth Entries field in this record. Each of the Auth Entries specifies a group name and the authorization for that group for this object:

| | |
|---|---|
| **Group Name** | PAYROLL |
| **Auth** | 0x00000004 |

The Group Name field is 12 bytes in length and contains a blank-filled Compaq NSK Administrative or SAFEGUARD File-sharing Group Name (first 12 characters only). The Auth field is a 4-byte (ULONG) bit mask with the authority for the group. There may be up to 250 individual Group and Auth pairs in each record.

Multiple records for the same object are used to hold authorization information for more than 250 groups if necessary.

The authority specification is the union of the individual bit patterns based on the following assignments:

| Authorization keyword | Formal name | Hexadecimal Value |
|---|---|---|
| connect | MQZAO_CONNECT | 0x00000001 |
| browse | MQZAO_BROWSE | 0x00000002 |
| get | MQZAO_INPUT | 0x00000004 |
| put | MQZAO_OUTPUT | 0x00000008 |
| inq | MQZAO_INQUIRE | 0x00000010 |
| set | MQZAO_SET | 0x00000020 |
| passid | MQZAO_PASS_IDENTITY_CONTEXT | 0x00000040 |
| passall | MQZAO_PASS_ALL_CONTEXT | 0x00000080 |
| setid | MQZAO_SET_IDENTITY_CONTEXT | 0x00000100 |
| setall | MQZAO_SET_ALL_CONTEXT | 0x00000200 |
| altusr | MQZAO_ALTERNATE_USER_AUTHORITY | 0x00000400 |
| allmqi | MQZAO_ALL_MQI | 0x000007FF |
| crt | MQZAO_CREATE | 0x00010000 |
| dlt | MQZAO_DELETE | 0x00020000 |
| dsp | MQZAO_DISPLAY | 0x00040000 |
| chg | MQZAO_CHANGE | 0x00080000 |
| clr | MQZAO_CLEAR | 0x00100000 |
| chgaut | MQZAO_AUTHORIZE | 0x00800000 |
| alladm | MQZAO_ALL_ADMIN | 0x009E0000 |
| none | MQZAO_NONE | 0x00000000 |
| all | MQZAO_ALL | 0x009E07FF |

These definitions are made in the header file cmqzc h. In the following example, `groupB` has been granted authorizations based on the hexadecimal number `0x40007`. This corresponds to:

```
     MQZAO_CONNECT                      0x00000001
     MQZAO_BROWSE                       0x00000002
     MQZAO_INPUT                        0x00000004
     MQZAO_DISPLAY                      0x00040000
                                        ----------
    Authority is:                       0x00040007
```

These access rights mean that anyone in `groupB` can issue the MQI calls:

- MQCONN
- MQGET (with browse)

They also have DISPLAY authority for the object associated with this authorization file.

## Class authorization records

The class authorization records hold authorizations that relate to the entire class. The object name and type fields correspond as follows:

| Object Name | Type |
|---|---|
| **@QMGRCLASS** | 0x80 |
| **@PROCESSCLASS** | 0x80 |
| **@QUEUECLASS** | 0x80 |

The entry MQZA0_CRT in the authorization field gives authorization to create an object in the class. This is the only class authority.

## All-class authorization record

The all-class authorization record holds authorizations that apply to an entire queue manager. The object name and type fields correspond as follows:

| Object Name | Type |
|---|---|
| **@ALLCLASSES** | 0x80 |

The following authorizations apply to the entire queue manager and are held in the all class authorization file.

| Entry... | Gives authorization to... |
|---|---|
| MQZAO_ALTUSER | Assume the identity of another user when interacting with MQSeries objects. |
| MQZAO_SET_ALL_CONTEXT | Set the context of a message when issuing MQPUT. |
| MQZAO_SET_IDENTITY_CONTEXT | Set the identity context of a message when issuing MQPUT. |

## Authorization files

# Chapter 9. MQSeries dead-letter queue handler

MQSeries for Compaq NSK provides a dead-letter queue (DLQ), also known as an undelivered-message queue, which is a holding queue for messages that cannot be delivered to their destination queues. Every queue manager in a network should have a DLQ.

Messages are put on the DLQ by queue managers, message channel agents (MCAs), and applications. All messages on the DLQ should be prefixed with the *dead-letter header* structure MQDLH. Messages put on the DLQ by a queue manager or by a message channel agent always have this header structure. Applications putting messages on the DLQ should also supply an MQDLH. The *Reason* field of the MQDLH structure contains a reason code that identifies why the message is on the DLQ.

You should have a routine that runs regularly to process messages on the DLQ. MQSeries supplies a default routine called the *dead-letter queue handler* (the DLQ handler), which you invoke using the **runmqdlq** command.

Instructions for processing messages on the DLQ are supplied to the DLQ handler by means of a user-written *rules table*. That is, the DLQ handler matches messages on the DLQ against entries in the rules table. When a DLQ message matches an entry in the rules table, the DLQ handler performs the action associated with that entry.

This chapter contains the following sections:

- "Invoking the DLQ handler"
- "DLQ handler rules table" on page 146
- "How the rules table is processed" on page 152
- "Example DLQ handler rules table" on page 153

## Invoking the DLQ handler

You invoke the DLQ handler using the **runmqdlq** command. You can name the DLQ that you want to process and the queue manager that you want to use as follows:

- From the command prompt using parameters. For example:

```
runmqdlq /IN qrule/ ABC1.DEAD.LETTER.QUEUE ABC1.QUEUE.MANAGER
```

- In the rules table. For example:

```
INPUTQ(ABC1.DEAD.LETTER.QUEUE) INPUTQM(ABC1.QUEUE.MANAGER)
```

The above examples apply to the DLQ called ABC1.DEAD.LETTER.QUEUE, owned by the queue manager ABC1.QUEUE.MANAGER.

© Copyright IBM Corp. 1993, 2001

If you do not specify the DLQ or the queue manager as shown above, the default queue manager for the installation is used along with the DLQ belonging to that queue manager.

The **runmqdlq** command reads input from the rules table, supplied to the standard IN file. You associate the rules table with **runmqdlq** by redirecting `IN` to the `rules file`.

To run the DLQ handler, you must be authorized to access both the DLQ itself and any message queues to which messages on the DLQ are forwarded. Furthermore, if the DLQ handler is to be able to put messages on queues with the authority of the user ID in the message context, you must be authorized to assume the identity of other users.

For more information about the **runmqdlq** command, see "runmqdlq (Run dead-letter queue handler)" on page 270

# DLQ handler rules table

The DLQ handler rules table defines how the DLQ handler is to process messages that arrive on the DLQ. There are two types of entry in a rules table:

- The first entry in the table, which is optional, contains *control data*.
- All other entries in the table are *rules* for the DLQ handler to follow. Each rule consists of a *pattern* (a set of message characteristics) that a message is matched against, and an *action* to be taken when a message on the DLQ matches the specified pattern. There must be at least one rule in a rules table.

Each entry in the rules table comprises one or more keywords.

For a description of the syntax rules applicable to the rules tables, see "Rules table conventions" on page 151.

## Control data

This section explains the keywords that you can include in a control-data entry in a DLQ handler rules table. Please note the following:

- The default value for a keyword, if any, is underlined.
- The vertical line (|) separates alternatives, only one of which can be specified.
- All keywords are optional.

**INPUTQ (***QueueName***|' ')**

This keyword is the name of the DLQ to which the rules table applies. It lets you name the DLQ you want to process:

1. If you specify a *QName* parameter on the **runmqdlq** command, it overrides any INPUTQ value in the rules table.
2. If you do not specify a *QName* parameter on the **runmqdlq** command, but you specify a value in the rules table, the INPUTQ value in the rules table is used.
3. If you do not specify a DLQ or you specify INPUTQ(' ') in the rules table, the DLQ belonging to the queue manager whose name is supplied on the *QMgrName* parameter on the **runmqdlq** command or on the INPUTQM keyword in the rules table is processed.

**INPUTQM (***QueueManagerName***|'_')**

This keyword is the name of the queue manager that owns the DLQ. It lets you name the queue manager that owns the DLQ named on the INPUTQ keyword:

1. If you specify a *QMgrName* parameter on the **runmqdlq** command, it overrides any INPUTQM value in the rules table.
2. If you do not specify a *QMgrName* parameter on the **runmqdlq** command, the INPUTQM value in the rules table is used.
3. If no queue manager is specified or you specify INPUTQM(' ') in the rules table, the default queue manager for the installation is used.

**RETRYINT (***Interval***|60)**

This keyword is the interval (in seconds) at which the DLQ handler should attempt to reprocess messages on the DLQ that could not be processed at the first attempt, and for which repeated attempts are requested. By default, the retry interval is 60 seconds.

**WAIT (YES|NO|***nnn***)**

This keyword indicates whether the DLQ handler should wait for further messages to arrive on the DLQ when it detects that there are no further messages that it can process.

**YES** This keyword causes the DLQ handler to wait indefinitely.

**NO** This keyword causes the DLQ handler to terminate when it detects that the DLQ is either empty or contains no messages that it can process.

*nnn* This keyword causes the DLQ handler to wait for *nnn* seconds for new work to arrive before terminating, after it detects that the queue is either empty or contains no messages that it can process.

You should specify WAIT (YES) for busy DLQs, and WAIT (NO) or WAIT (*nnn*) for DLQs that have a low level of activity. If the DLQ handler is allowed to terminate, you should reinvoke it by using triggering.

The control data shown in Figure 40 shows that the rules table applies to the DLQ belonging to queue manager QM1. The plus sign (+) at the end of line 1 indicates that the control data continues from the first nonblank character on line 2.

```
INPUTQ' ' +
  INPUTQM'QM1'
```

*Figure 40. Example control data*

As an alternative to including control data in the rules table, you can supply the names of the DLQ and its queue manager as input parameters of the **runmqdlq** command. If any value is specified both in the rules table and on input to the **runmqdlq** command, the value specified on the **runmqdlq** command takes precedence.

**Note:** If a control-data entry is included in the rules table, it must be the first entry in the table.

## Rules (patterns and actions)

Figure 41 on page 148 shows an example rule from a DLQ handler rules table. This rule instructs the DLQ handler to make three attempts to deliver to its destination queue any persistent message that was put on the DLQ because MQPUT and MQPUT1 were inhibited.

```
PERSIST(MQPER_PERSISTENT) REASON (MQRC_PUT_INHIBITED) +
  ACTION (RETRY) RETRY (3)
```

*Figure 41. Example rule.* The plus sign (+) at the end of line 1 indicates that the rule continues from the first nonblank character on line 2.

All keywords that you can use on a rule are explained in the remainder of this section. Please note the following:

- The default value for a keyword, if any, is underlined. For most keywords, the default value is * (asterisk), which matches any value.
- The vertical line (|) separates alternatives, only one of which can be specified.
- All keywords except ACTION are optional.

This section begins with a description of the pattern-matching keywords (those against which messages on the DLQ are matched), and then explains the action keywords (those that determine how the DLQ handler is to process a matching message).

## Pattern-matching keywords

The pattern-matching keywords that you use to specify values against matched messages on the DLQ are explained below. All pattern-matching keywords are optional.

**APPLIDAT (**_ApplIdentityData_**| *)**
This keyword is the _ApplIdentityData_ value specified in the message descriptor (MQMD) of the message on the DLQ.

**APPLNAME (**_PutApplName_**|*)**
This keyword is the name of the application that issued the MQPUT or MQPUT1 call, as specified in the _PutApplName_ field of the message descriptor (MQMD) of the message on the DLQ.

**APPLTYPE (**_PutApplType_**|*)**
This keyword is the _PutApplType_ value specified in the message descriptor (MQMD) of the message on the DLQ.

**DESTQ (**_QueueName_**|*)**
This keyword is the name of the message queue for which the message is destined.

**DESTQM (**_QueueManagerName_**|*)**
This keyword is the name of the queue manager of the message queue for which the message is destined.

**FEEDBACK (**_Feedback_**|*)**
If the _MsgType_ value is MQFB_REPORT, the keyword _Feedback_ describes the nature of the report.

Symbolic names can be used. For example, you can use the symbolic name MQFB_COA to identify those messages on the DLQ that require confirmation of their arrival on their destination queues.

**FORMAT (**_Format_**|*)**
This keyword is the name that the sender of the message uses to describe the format of the message data.

**MSGTYPE (**_MsgType_**|*)**
This keyword is the message type of the message on the DLQ.

Symbolic names can be used. For example, you can use the symbolic name MQMT_REQUEST to identify those messages on the DLQ that require replies.

**PERSIST (**_Persistence_**|*)**
This keyword is the persistence value of the message. (The persistence of a message determines whether it survives restarts of the queue manager.)

Symbolic names can be used. For example, you can use the symbolic name MQPER_PERSISTENT to identify those messages on the DLQ that are persistent.

**REASON (**_ReasonCode_**|*)**
This keyword is the reason code that describes why the message was put to the DLQ.

Symbolic names can be used. For example, you can use the symbolic name MQRC_Q_FULL to identify those messages placed on the DLQ because their destination queues were full.

**REPLYQ (**_QueueName_**|*)**
This keyword is the name of the reply-to queue specified in the message descriptor (MQMD) of the message on the DLQ.

**REPLYQM (**_QueueManagerName_**|*)**
This keyword is the name of the queue manager of the reply-to queue, as specified in the message descriptor (MQMD) of the message on the DLQ.

**USERID (**_UserIdentifier_**|*)**
This keyword is the user ID of the user who originated the message on the DLQ, as specified in the message descriptor (MQMD).

## Action keywords

The action keywords that you use to describe how a matching message is to be processed are detailed as follows:

**ACTION (DISCARD|IGNORE|RETRY|FWD)**
This keyword is the action to be taken for any message on the DLQ that matches the pattern defined in this rule.

**DISCARD** This keyword causes the message to be deleted from the DLQ.

**IGNORE** This keyword causes the message to be left on the DLQ.

**RETRY** This keyword causes the DLQ handler to try again to put the message on its destination queue.

**FWD** This keyword causes the DLQ handler to forward the message to the queue named on the FWDQ keyword.

You must specify the ACTION keyword. The number of attempts made to implement an action is governed by the RETRY keyword. The interval between attempts is controlled by the RETRYINT keyword of the control data.

**FWDQ (**_QueueName_**|&DESTQ|&REPLYQ)**
This keyword is the name of the message queue to which the message should be forwarded when ACTION (FWD) is requested.

_QueueName_
This keyword is the name of a message queue. FWDQ(' ') is not valid.

**&DESTQ**
This keyword causes the queue name to be taken from the *DestQName* field in the MQDLH structure.

**&REPLYQ**
This keyword causes the name to be taken from the *ReplyToQ* field in the message descriptor, MQMD.

To avoid error messages when a rule specifying FWDQ (&REPLYQ) matches a message with a blank *ReplyToQ* field, you can specify REPLYQ (?*) in the message pattern.

**FWDQM (***QueueManagerName***|&DESTQM|&REPLYQM|'_')**
This keyword identifies the queue manager of the queue to which a message is to be forwarded.

*QueueManagerName*
This keyword is the name of the queue manager of the queue to which a message is to be forwarded when ACTION (FWD) is requested.

**&DESTQM**
This keyword causes the queue manager name to be taken from the *DestQMgrName* field in the MQDLH structure.

**&REPLYQM**
This keyword causes the name to be taken from the *ReplyToQMgr* field in the message descriptor (MQMD).

**' '** FWDQM(' ') is the default value and identifies the local queue manager.

**HEADER (YES|NO)**
This keyword specifies whether the MQDLH should remain on a message for which ACTION (FWD) is requested. By default, the MQDLH remains on the message. The HEADER keyword is not valid for actions other than FWD.

**PUTAUT (DEF|CTX)**
This keyword defines the authority with which messages should be put by the DLQ handler:

**DEF** This keyword causes messages to be put with the authority of the DLQ handler itself.

**CTX** This keyword causes the messages to be put with the authority of the user ID in the message context. If you specify PUTAUT (CTX), you must be authorized to assume the identity of other users.

**RETRY (***RetryCount***|1)**
RETRY is the number of times, in the range 1–999, that an action should be attempted (at the interval specified on the RETRYINT keyword of the control data).

**Note:** The count of attempts made by the DLQ handler to implement any particular rule is specific to the current instance of the DLQ handler; the count does not persist across restarts. If the DLQ handler is restarted, the count of attempts made to apply a rule is reset to zero.

## Rules table conventions

The rules table must adhere to the following conventions regarding its syntax, structure, and contents:

- A rules table must contain at least one rule.
- Keywords can occur in any order.
- A keyword can be included once only in any rule.
- Keywords are not case sensitive.
- A keyword and its parameter value must be separated from other keywords by at least one blank or comma.
- Any number of blanks can occur at the beginning or end of a rule, and between keywords, punctuation, and values.
- Each rule must begin on a new line.
- For reasons of portability, the significant length of a line should not be greater than 72 characters.
- Use the plus sign (+) as the last nonblank character on a line to indicate that the rule continues from the first nonblank character in the next line. Use the minus sign (–) as the last nonblank character on a line to indicate that the rule continues from the start of the next line. Continuation characters can occur within keywords and parameters.
- Comment lines, which begin with an asterisk (*), can occur anywhere in the rules table.
- Blank lines are ignored.
- Each entry in the DLQ handler rules table comprises one or more keywords and their associated parameters. The parameters must follow these syntax rules:
  - Each parameter value must include at least one significant character. The delimiting quotation marks in quoted values are not considered significant. For example, these parameters are valid:

    | | |
    |---|---|
    | `FORMAT('ABC')` | 3 significant characters |
    | `FORMAT(ABC)` | 3 significant characters |
    | `FORMAT('A')` | 1 significant character |
    | `FORMAT(A)` | 1 significant character |
    | `FORMAT(' ')` | 1 significant character |

    These parameters are invalid because they contain no significant characters:

    ```
    FORMAT('')
    FORMAT( )
    FORMAT()
    FORMAT
    ```
  - Wildcard characters are supported: you can use the question mark (?) in place of any single character, except a trailing blank; you can use the asterisk (*) in place of zero or more adjacent characters. The asterisk (*) and the question mark (?) are *always* interpreted as wildcard characters in parameter values.
  - Wildcard characters cannot be included in the parameters of these keywords: ACTION, HEADER, RETRY, FWDQ, FWDQM, and PUTAUT.
  - Trailing blanks in parameter values, and in the corresponding fields in the message on the DLQ, are not significant when performing wildcard matches. However, leading and embedded blanks within strings in quotation marks are significant to wildcard matches.

- Numeric parameters cannot include the question mark (?) wildcard character. The asterisk (*) can be used in place of an entire numeric parameter, but cannot be included as part of a numeric parameter. For example, these are valid numeric parameters:

| | |
|---|---|
| **MSGTYPE(2)** | Only reply messages are eligible |
| **MSGTYPE(*)** | Any message type is eligible |
| **MSGTYPE('*')** | Any message type is eligible |

  However, `MSGTYPE('2*')` is not valid, because it includes an asterisk (*) as part of a numeric parameter.
- Numeric parameters must be in the range 0–999. If the parameter value is in this range, it is accepted, even if it is not currently valid in the field to which the keyword relates. Symbolic names can be used for numeric parameters.
- If a string value is shorter than the field in the MQDLH or MQMD to which the keyword relates, the value is padded with blanks to the length of the field. If the value, excluding asterisks, is longer than the field, an error is diagnosed. For example, these are all valid string values for an 8-character field:

| | |
|---|---|
| **'ABCDEFGH'** | 8 characters |
| **'A*C*E*G*I'** | 5 characters excluding asterisks |
| **'*A*C*E*G*I*K*M*O*'** | 8 characters excluding asterisks |

- Strings that contain blanks, lowercase characters, or special characters other than period (.), forward slash (/), underscore (_), and percent sign (%) must be enclosed in single quotation marks. Lowercase characters not enclosed in quotation marks are folded to uppercase. If the string includes a quotation, two single quotation marks must be used to denote both the beginning and the end of the quotation. When the length of the string is calculated, each occurrence of double quotation marks is counted as a single character.

## How the rules table is processed

The DLQ handler searches the rules table for a rule whose pattern matches a message on the DLQ. The search begins with the first rule in the table and continues sequentially through the table. When a rule with a matching pattern is found, the action from that rule is attempted. The DLQ handler increments the retry count for a rule by one whenever it attempts to apply that rule. If the first attempt fails, the attempt is repeated until the count of attempts made matches the number specified on the RETRY keyword. If all attempts fail, the DLQ handler searches for the next matching rule in the table.

This process is repeated for subsequent matching rules until an action is successful. When each matching rule has been attempted the number of times specified on its RETRY keyword, and all attempts have failed, ACTION (IGNORE) is assumed. ACTION (IGNORE) is also assumed if no matching rule is found.

**Notes:**

1. Matching rule patterns are sought only for messages on the DLQ that begin with an MQDLH. Messages that do not begin with an MQDLH are reported periodically as being in error, and remain on the DLQ indefinitely.
2. All pattern keywords can be allowed to default, such that a rule can consist of an action only. However, action-only rules are applied to all messages on the queue that have MQDLHs and that have not already been processed in accordance with other rules in the table.

3. The rules table is validated when the DLQ handler is started, and errors are flagged at that time. (Error messages issued by the DLQ handler are described in the *MQSeries Messages* book.) You can make changes to the rules table at any time, but those changes do not take effect until the DLQ handler is restarted.
4. The DLQ handler does not alter the content of messages, of the MQDLH, or of the message descriptor. The DLQ handler always puts messages to other queues with the message option MQPMO_PASS_ALL_CONTEXT.
5. The DLQ handler opens the DLQ with the MQOO_INPUT_AS_Q_DEF option.
6. Multiple instances of the DLQ handler could run concurrently against the same queue, using the same rules table. However, it is more usual for there to be a one-to-one relationship between a DLQ and a DLQ handler.

## Ensuring that all DLQ messages are processed

The DLQ handler keeps a record of all messages on the DLQ that have been viewed but not removed. If you use the DLQ handler as a filter to extract a small subset of the messages from the DLQ, the DLQ handler still has to keep a record of those messages on the DLQ that it did not process. Also, the DLQ handler cannot guarantee that new messages arriving on the DLQ are viewed, even if the DLQ is defined as first-in-first-out (FIFO). Therefore, if the queue is not empty, a periodic rescan of the DLQ is performed to check all messages. For these reasons, you should ensure that the DLQ contains as few messages as possible. If messages that cannot be discarded or forwarded to other queues (for whatever reason) are allowed to accumulate on the queue, the workload of the DLQ handler increases and the DLQ itself is in danger of filling up.

You can take specific measures to enable the DLQ handler to empty the DLQ. For example, do not use ACTION (IGNORE), which leaves messages on the DLQ. ACTION (IGNORE) is assumed for messages that are not explicitly addressed by other rules in the table. Instead, for those messages that you would otherwise ignore, use an action that moves the messages to another queue. For example:

```
ACTION (FWD) FWDQ (IGNORED.DEAD.QUEUE) HEADER (YES)
```

Similarly, the final rule in the table should process messages that have not been addressed by earlier rules in the table. For example, the final rule in the table could be:

```
ACTION (FWD) FWDQ (REALLY.DEAD.QUEUE) HEADER (YES)
```

This action causes messages that fall through to the final rule in the table to be forwarded to the queue `REALLY.DEAD.QUEUE`, where they can be processed manually. If you do not have such a rule, messages are likely to remain on the DLQ indefinitely.

# Example DLQ handler rules table

The following is an example rules table that contains a single control-data entry and several rules:

```
*************************************************************************
*       An example rules table for the runmqdlq command                 *
*************************************************************************
* Control data entry
```

## Example rules table

```
* ------------------
* If no queue manager name is supplied as an explicit parameter to
* runmqdlq, use the default queue manager for the machine.
* If no queue name is supplied as an explicit parameter to runmqdlq,
* use the DLQ defined for the local queue manager.
*
inputqm(' ')  inputq(' ')

* Rules
* -----
* We include rules with ACTION (RETRY) first to try to
* deliver the message to the intended destination.

* If a message is placed on the DLQ because its destination
* queue is full, attempt to forward the message to its
* destination queue. Make 5 attempts at approximately
* 60-second intervals (the default value for RETRYINT).

REASON(MQRC_Q_FULL) ACTION(RETRY) RETRY(5)

* If a message is placed on the DLQ because of a put inhibited
* condition, attempt to forward the message to its
* destination queue. Make 5 attempts at approximately
* 60-second intervals (the default value for RETRYINT).

REASON(MQRC_PUT_INHIBITED) ACTION(RETRY) RETRY(5)

* The AAAA corporation are always sending messages with incorrect
* addresses. When we find a request from the AAAA corporation,
* we return it to the DLQ (DEADQ) of the reply-to queue manager
* (&REPLYQM).
* The AAAA DLQ handler attempts to redirect the message.

MSGTYPE(MQMT_REQUEST) REPLYQM(AAAA.*) +
  ACTION(FWD) FWDQ(DEADQ) FWDQM(&REPLYQM)

* The BBBB corporation never do things by half measures. If
* the queue manager BBBB.1 is unavailable, try to
* send the message to BBBB.2

DESTQM(bbbb.1) +
  action(fwd) fwdq(&DESTQ) fwdqm(bbbb.2) header(no)

* The CCCC corporation considers itself very security
* conscious, and believes that none of its messages
* will ever end up on one of our DLQs.
* Whenever we see a message from a CCCC queue manager on our
* DLQ, we send it to a special destination in the CCCC organization
* where the problem is investigated.

REPLYQM(CCCC.*) +
  ACTION(FWD) FWDQ(ALARM) FWDQM(CCCC.SYSTEM)

* Messages that are not persistent run the risk of being
* lost when a queue manager terminates. If an application
* is sending nonpersistent messages, it should be able
* to cope with the message being lost, so we can afford to
* discard the message.

PERSIST(MQPER_NOT_PERSISTENT) ACTION(DISCARD)

* For performance and efficiency reasons, we like to keep
* the number of messages on the DLQ small.
* If we receive a message that has not been processed by
* an earlier rule in the table, we assume that it
* requires manual intervention to resolve the problem.
* Some problems are best solved at the node where the
* problem was detected, and others are best solved where
* the message originated. We don't have the message origin,
* but we can use the REPLYQM to identify a node that has
* some interest in this message.
* Attempt to put the message onto a manual intervention
* queue at the appropriate node. If this fails,
* put the message on the manual intervention queue at
* this node.

REPLYQM('?*') +
  ACTION(FWD) FWDQ(DEADQ.MANUAL.INTERVENTION) FWDQM(&REPLYQM)

ACTION(FWD) FWDQ(DEADQ.MANUAL.INTERVENTION)
```

## Example rules table

# Chapter 10. Instrumentation and EMS events

This chapter:

- Provides a brief introduction to MQSeries instrumentation events, which you can use to monitor the operation of queue managers. See "MQSeries instrumentation events". For detailed information about instrumentation events, see the *MQSeries Programmable System Management* book.
- Describes the use of Event Management Service (EMS) events by MQSeries for Compaq NSK. See "Event Management Service (EMS) events" on page 159.

## MQSeries instrumentation events

Instrumentation events cause *event messages* to be generated when a queue manager detects a predefined set of conditions. For example, a *Queue Full* event results from the following conditions:

- Queue Full events are enabled for a specified queue.
- 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 cause instrumentation events include:

- A limit on 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 specifying a user ID that is not authorized

With the exception of channel events, all instrumentation events must be enabled before they can be generated.

The event message contains information about the conditions resulting in the event. It is put onto an *event queue*. An application can retrieve the event message from this queue for analysis.

If you define 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

There are four types of instrumentation event:

**Queue manager events**

Queue manager events are related to the definitions of resources within queue managers. For example, a queue manager event could be generated when an application attempts to put a message to a queue that does not exist.

**Performance events**

Performance events are notifications that a threshold has been reached by a resource. For example, a performance event could be generated when a

© Copyright IBM Corp. 1993, 2001

queue-depth limit has been reached or, following an MQGET call, if a queue has not been serviced within a predefined time.

**Channel events**

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

**Trigger events**

A trigger event can occur when a queue manager detects that the conditions for the trigger event have been met. For example, a queue can be configured to generate a trigger event each time a message arrives. (The conditions for trigger events and instrumentation events are quite different.)

A trigger event causes a trigger message to be put on an initiation queue and, optionally, an application program is started.

## Event notification through event queues

When an event occurs, the queue manager puts an event message on the appropriate event queue, if defined. The event message contains information about the event that you can retrieve by writing a suitable MQI application program that:

- Gets the message from the queue.
- Processes the message to extract the event data. For a description of event message formats, see the *MQSeries Programmable System Management* book.

Each category of event has its own event queue. All events in that category result in an event message being put onto the same queue.

| **This event queue...** | **Contains messages from...** |
|---|---|
| SYSTEM.ADMIN.QMGR.EVENT | Queue manager events |
| SYSTEM.ADMIN.PERFM.EVENT | Performance events |
| SYSTEM.ADMIN.CHANNEL.EVENT | Channel events |

## Using triggered event queues

You can set up the event queues with triggers so that, when an event is generated, the event message put onto the event queue starts a user-written monitoring application. This application can process the event messages and take appropriate action. For example, some events can require that an operator be informed, and others can start an application that performs various administration tasks automatically.

## Enabling instrumentation events

How you enable an instrumentation event depends on the event type:

- Queue manager events are enabled by setting attributes on the queue manager.
- Performance events as a whole must be enabled on the queue manager. You must also enable specific performance events by setting the appropriate queue attribute, and identify the conditions, such as a queue-depth-high limit, that will result in the event.
- Channel events occur automatically; they do not need to be enabled. If you do not want to monitor channel events, you can put-inhibit the channel event queue.

You enable and disable the generation of instrumentation events using either of the following:

- MQSC commands. For more information, see the *MQSeries MQSC Command Reference*.
- PCF commands for queue managers. For more information, see the *MQSeries Programmable System Management* book.
- MQAI commands. For more information, see the *MQSeries Administration Interface Programming Guide and Reference*.

## Event messages

Event messages contain information relating to the origin of an event, including the type of event, the name of the application that caused the event, and, for performance events, a short statistics summary for the queue.

The format of event messages is similar to that of PCF response messages. The message data can be retrieved from event messages by user-written administration programs using the data structures described in the *MQSeries Programmable System Management* book.

# Event Management Service (EMS) events

MQSeries for Compaq NSK generates Event Management Service (EMS) event messages that correspond to the MQSeries queue-manager events, channel events, and performance events. EMS messages may also be generated that correspond to the message entries in the MQSeries logs and to FFSTs. These event messages can alert system operators and administrators to software conditions that could have an adverse effect on the MQSeries operating environment.

## EMS template files supplied with MQSeries for Compaq NSK

The following files are supplied in subvolume ZMQSSYS:

**ZMQSTMPL (file code 839)**
An EMS template object file containing the formatting templates for the EMS events generated by MQSeries.

**ZMQSDDL (file code 101)**
The Data Definition Language schema for the EMS events generated by MQSeries.

**ZMQSC (file code 101)**
Compiled output (C) from the DDL compiler of definitions of the EMS events generated by the product.

**ZMQSCOB (file code 101)**
Compiled output (COBOL) from the DDL compiler of definitions of the EMS events generated by the product.

**ZMQSPAS (file code 101)**
Compiled output (PASCAL) from the DDL compiler of definitions of the EMS events generated by the product.

**ZMQSTACL (file code 101)**
Compiled output (TACL) from the DDL compiler of definitions of the EMS events generated by the product.

**ZMQSTAL (file code 101)**
Compiled output (TAL) from the DDL compiler of definitions of the EMS events generated by the product.

## EMS events

The subvolume ZMQSSYS contains the EMS template file SMQSTMPL, from which the template file ZMQSTMPL is generated. The file ZMQSTMPL is ready for integration with your system's event templates using COUP and SYSGEN. The source of the event templates is supplied, so that you can modify the formatting of the events when they are used in your environment.

For example, you might not be interested in displaying all of the information that is contained in an event, or you might want to add or change text that is displayed along with the information in the event. See the Compaq documentation for a description of the EMS event template source language, and for the procedures used to compile the definitions to produce an alternative ZMQSTMPL file.

# Integrating the MQSeries EMS event templates

The template object file must be integrated into your system's resident and nonresident EMS template files, so that programs such as VIEWPOINT and EMSDIST can format and display MQSeries EMS events.

A procedure for integrating the MQSeries EMS templates into the system templates is described in the remainder of this section. Note that different procedures might be preferred in your installation.

1. Determine the names of the current system templates using the COUP command `INFO ALLPROCESSORS`: note the values displayed for the `EMS^TEMPLATES` parameter. For example:

```
     $DEV2 ZMQSSYS 425> coup
     CONFIGURATION UTILITY PROGRAM - T9023D30 - (26MAY95)     SYSTEM
\RAPTOR
     COPYRIGHT Compaq COMPUTERS INCORPORATED 1987-1994
     CONFIG $SYSTEM.SYS06.OSCONFIG
     1) info allprocessors
           EMS^TEMPLATES ( RESIDENT $SYSTEM.SYS01.RTMPLATE,
                           NONRESIDENT $SYSTEM.SYS01.RTMPLATE )
           SYSTEM^ID     ( NAME \RAPTOR, NUMBER 001 )
           SYSTEM^TIME   ( GMT^OFFSET -05:00, DST USA66 )
           DP2_UPSOPTION ( OFF )
     2) exit
```

2. Determine the name of the current system template for a G Series operating system using the SCF command ASSUME SUBSYS $ZZKRN; INFO. Note the values displayed for NONRESIDENT_TEMPLATES and RESIDENT_TEMPLATES. For example:

```
SCF;ASSUME SUBSYS $ZZKRN;INFO
NONSTOP KERNEL - Info SUBSYS  \HAWK.$ZZKRN
Current Settings
*DAYLIGHT_SAVING_TIME ................ USA66
*NONRESIDENT_TEMPLATES................ $SYSTEM.SYS01.TEMPLATE
*POWERFAIL_DELAY_TIME................. 30
*RESIDENT_TEMPLATES................... $SYSTEM.SYS01.RTMPLATE
SUPER_SUPER_IS_UNDENIABLE............ OFF
*SYSTEM_NAME.......................... \HAWK
*SYSTEM_NUMBER........................ 2
SYSTEM_PROCESSOR_TYPE ............... NSR-W
*TIME_ZONE_OFFSET..................... -05:00

Pending Changes (will take effect at next system load)
None  Total Errors = 0    Total Warnings = 0
```

3. Run the TEMPLI compiler to create new system template files combining the current system templates with the new MQSeries templates. This is a two-step process:
   a. Create a text file containing the following commands:

```
FILE <current NONRESIDENT system template file>
FILE <MQSeries install volume>.ZMQSSYS.ZMQSTMPL
EXIT
```

   For example:

```
   FILE $SYSTEM.SYS06.TEMPLATE
   FILE $DEV2.ZMQSSYS.ZMQSTMPL
   EXIT
```

   b. Run the TEMPLI compiler, specifying the new text file as input:

```
 TEMPLI /IN <command file>/<new resident template file>, <new
  nonresident template file>
```

   For example, if the command file you created is called TEMGUIDE and you are creating new template files in $SYSTEM.EMS:

```
 TEMPLI /IN TEMGUIDE/$SYSTEM.EMS.NEWRES, $SYSTEM.EMS.NEWNRES
```

   The compilation of the new template files can take several minutes, as all the EMS event templates required on your system are processed.
4. Using the COUP command, configure your system to use the new EMS event templates in place of the current templates:

## EMS events

```
ASSUME ALLPROCESSORS
ALTER EMS^TEMPLATES(RESIDENT <new resident template file>,
                         NONRESIDENT <new nonresident template file>)
EXIT
```

For G Series use the SCF commands to configure your system to use the new EMS event templates:

```
ALTER $ZZKRN, RESIDENT_TEMPLATES $SYSTEM.SYS01.NEWRES
ALTER $ZZKRN, NONRESIDENT_TEMPLATES $SYSTEM.SYS01.NEWNRES
        EXIT
```

**Note:** To make this change permanent, you must update the system using SYSGEN.

For further information about EMS templates, see the Compaq *DSM Template Services Manual*. This book also describes how to use SYSGEN to perform this task.

# Defining the PARAM MQEMSEVENTS

To complete the enablement of MQSeries EMS events, you must ensure that the PARAM MQEMSEVENTS is correctly defined. The value is a four-character string interpreted as a bit map, as follows:

| EMS message | Bit-map entry | MQEMSEVENT value |
|---|---|---|
| FFST | 0x00000001 | 1 |
| START / STOP | 0x00000002 | 2 |
| PERFORMANCE | 0x00000004 | 4 |
| CHANNEL | 0x00000008 | 8 |
| QUEUE MANAGER | 0x00000010 | 16 |
| MESSAGE | 0x00000020 | 32 |
| ERROR | 0x00000040 | 64 |
| ALL | 0x0000007F | 127 |

Thus, to switch on all EMS events for MQSeries, you must define the following PARAM in the TACL environment from which any administration commands are issued:

```
PARAM MQEMSEVENTS 127
```

This definition is also required in server class definitions of all server classes for MQSeries. Each server class may be configured with different options. See “Changing the parameters of PATHWAY server classes” on page 39 for more information.

By default, no EMS events are generated (that is, the PARAMs are not defined).

## Using an alternative collector

On a Compaq NSK system, the default EMS event collector is called $0, and is always present. All EMS events generated by an MQSeries queue manager are sent to the default collector. If you want a different collector to collect EMS events for a queue manager, modify the `EMSCollector` entry in the `Configuration` stanza in the QMINI file, and restart the queue manager. You may specify a different EMS event collector for each queue manager.

## Writing programs to process MQSeries EMS events

You can write an application to monitor an MQSeries queue manager by processing EMS event messages. Such an application could also affect the operation of the queue manager by issuing PCF commands in response to the EMS event messages generated.

The files ZMQSC, ZMQSTAL, ZMQSCOB, ZMQSPAS, and ZMQSTACL supplied with MQSeries for Compaq NSK in the ZMQSSYS subvolume define the tokens contained in the MQSeries EMS event messages in C, TAL, COBOL, PASCAL, and TACL. These definitions could be used by an administration program to understand the format of the messages.

For further information about the EMS events generated by MQSeries, see "Appendix P. EMS event template used by MQSeries for Compaq NSK" on page 379.

# Chapter 11. Understanding transactional support and messaging

Applications that use the Message Queue Interface (MQI) let you execute put and get operations under syncpoint control. In MQSeries for Compaq NSK, there are two transactional operations as follows:

- Commit — the act of completing a transaction so that changes to the database are recorded and stable. Protected resources are released after the transaction is committed.
- Back out — an operation that reverses all the changes made during the current unit of recovery or unit of work. After the operation is complete, a new unit of recovery or unit of work may be started.

Commit and back out are provided as part of the TM/MP (or TMF) Transaction environment on Compaq NSK. On MQSeries for Compaq NSK, MQPUT, MQGET, and MQPUT1 are syncpointed operations by default. That is, unless no syncpointing is requested explicitly by an application, a TMF transaction *must* be in progress or the MQI call fails.

An application initiates a TM/MP transaction using the TM/MP BEGINTRANSACTION procedure, commits the transaction using ENDTRANSACTION and can backout the transaction using ABORTTRANSACTION.

MQSeries for Compaq NSK also supports products that provide syncpoint operation via TM/MP, for example, NonStop Tuxedo.

## Using the NonStop TM/MP (Transaction Manager)

MQSeries for Compaq NSK V5.1 relies on the transaction management facilities of Compaq's NonStop TM/MP to maintain transaction integrity.

The NonStop TM/MP transaction system provides transactional protection and concurrency, and object-catalog and message integrity.

TM/MP transactions can coordinate MQSeries put and get operations with updates to ENSCRIBE or NonStop SQL database files made by the application.

MQSeries handles TM/MP transactions transparently. If you have a TM/MP transaction in progress when an MQI function is called, any put and get operations with the syncpoint option become part of the same transaction. That is, the updates to the queues occur when the transaction is committed. In the event of any failure, TM/MP ensures that all committed transactions are applied to the database files, and uncommitted transactions are backed out. A transaction backout reapplies before-images to database records to undo the effects of a cancelled transaction. Changes do not occur until a commit operation is complete.

If the user application has a transaction in progress and attempts an MQI call out of syncpoint, MQSeries suspends the current, inherited transaction, starts one of its own, commits that transaction, and resumes the original transaction prior to returning control to the user application. Updates to queues resulting from put and get operations occur immediately.

© Copyright IBM Corp. 1993, 2001

TM/MP transactions are used to coordinate put and get operations on non-persistent messages as well as persistent ones. MQSeries for Compaq NSK uses a special interface to TM/MP called *OpenTMF* to allow this coordination to occur.

## Syncpointing limits

The file system can limit the amount of persistent message data that can be put or got within a single transaction by limiting the number of record locks on the physical files that hold persistent message data.

The file system default lock limit per transaction is 5000 locks per disk volume. You can change this using SCF. For example, the following command changes the limit to 10000 locks per transaction per disk volume:

```
ALTER DISK $DISK01, MAXLOCKSPERTCP 10000
```

For messages that are stored in queue overflow files (because their size is below the threshold size for the use of message overflow files) the number and size of messages is limited.

We advise you to set the message overflow threshold size to no more than 200 KB. At this message size, the default record lock limit can accommodate about 100 messages within a single transaction — more than adequate for most applications. In addition, the performance benefits of using message overflow files become significant at this message size.

Also note that the use of TM/MP audit trail is greatly reduced when message overflow files are used instead of queue overflow files.

For more information about the differences and benefits of message and queue overflow files, see “Message overflow files” on page 206.

## No-syncpoint operations on persistent messages

Persistent messages require TM/MP transactions started internally by the queue server in order to update the ENSCRIBE files that hold message data. There is a limit imposed by the NSK File System of 100 concurrent transactions started by any one process. Therefore, a single queue server can support no more than 100 concurrent no-syncpoint persistent message PUT or GET operations. The only way this can happen is if multiple large messages (larger than 32 KB) are being enqueued or dequeued at the same time through the same queue server.

If this situation occurs, the MQPUT or MQGET will be terminated by the reason code MQRC_SYNCPOINT_LIMIT_REACHED. Re-assign queues to alternate queue servers in order to spread the load across multiple processes, or change applications to use different queues hosted by different queue servers.

## Syncpoint operations on non-persistent messages

Since non-persistent messages are stored in memory and not in audited disk files, they require no audit trail space themselves. MQSeries uses an internal interface of TM/MP to control the availability of non-persistent messages that are enqueued or dequeued in syncpoint. Any mixture of persistent and non-persistent messages may be included within a syncpoint operation—MQSeries will ensure that at the time the TM/MP transaction completes, the operations on all non-persistent messages will be logically committed or backed out at the same time as those for persistent messages, depending on the actual outcome of the transaction.

## Configuration requirements for TM/MP and MQSeries for Compaq NSK

Your NSK system needs to be configured with TMF (TM/MP) auditing enabled for all volumes that are to contain queue managers or queues. Use the TMFCOM command `status datavols` to determine the status of auditing on any volume on your system. (Note that you have to be SUPER.SUPER to use TMFCOM.) In addition, the TMF audit trails configured for the data volumes that support queue managers must be large enough to allow for the peak rate and size of message traffic expected on all queue managers that use these volumes.

Since misbehaved applications can cause long-running transactions, the TMF system should be configured automatically to cancel long-running transactions. The size of the audit trail, and the time limit on long-running transactions, are application-dependent tuning parameters. The audit trail configured for MQSeries does not need to be configured for dumping to tape.

### Monitoring

Use the TMFCOM interface to monitor the status of TMF, with MQSeries running. Use the `status tmf` and `status datavols` commands to investigate the general status of TMF, and the status of individual data volumes.

The System event log (EMS) should also be monitored for critical TM/MP events that indicate potential future problems within TM/MP that could affect MQSeries or the applications that use it. TM/MP is a critical resource for MQSeries and must operate continuously for MQSeries to function properly.

### Audit-trail size

Approximate TM/MP audit-trail sizings can be calculated using the following guidelines:

- Audit trail space is required for persistent message operations (put and destructive get) only.
- The audit trail space should be approximately the total message data size plus 1500 bytes.
- Persistent message operations involving messages that are above the message overflow threshold require only 4 KB of audit trail per put or get, irrespective of their size.

### Resource manager configuration

The internal interface of TM/MP needs to be configured appropriately for the volume of transactions that are expected to be processed using MQSeries. The MQSeries queue servers take the role of resource manager as far as the TM/MP subsystem is concerned, and there are various thresholds and limits in the TM/MP subsystem that apply to resource managers. The required configuration depends on the number of queue servers you use, the distribution of queue servers across the CPUs and how many concurrent syncpoint operations are in progress at any one time. The ALTER BEGINTRANS command of TMFCOM is used to change the values, as described below:

**RMOPENPERCPU** - should be at least twice the maximum number of queue servers that will run in any CPU. The default value of 128 is usually sufficient.

### TM/MP configuration requirements

**BRANCHESPERRM** - should be at least the maximum number of concurrent syncpoint operations that can be handled by any single queue server. The default value of 128 is usually sufficient, but if not then this parameter may be increased to the maximum value of 1024, or Queues may be assigned to other queue servers to reduce the maximum number of concurrent syncpoint operations handled by a queue server.

For new values of these parameters to take effect, the TM/MP subsystem must be stopped and restarted.

## Troubleshooting

EMS events or FFST reports indicating that BEGINTRANSACTION commands have been disabled by TMF usually mean that the audit trail is filled. This can occur because the audit trail is too small, or because a badly behaved application has held a long-running transaction and TMF has not terminated it in time.

In this instance:

- Increase the size of the audit trail, or
- Identify the cause of the long-running transaction and correct it, or
- Reconfigure TMF to terminate long-running transactions after a shorter period.

EMS events and FFST reports indicating that TMF is not running indicate a configuration problem with TMF that must be corrected before running the queue manager again. In general, the MQSeries queue manager requires TMF to be running correctly to operate in any capacity. Although messages are not lost or corrupted, the queue manager is not able to operate without TMF.

# Chapter 12. Recovery and restart

A messaging system ensures that messages entered into the system are delivered to their destination. A messaging system must also provide a method of tracking the messages in the system, and of recovering messages if the system fails for any reason.

MQSeries for Compaq NSK ensures that persistent messages are not lost by using the Compaq NonStop Transaction Manager (TM/MP). TM/MP provides transaction protection, queue-file consistency, and queue-file recovery.

MQSeries for Compaq NSK also uses NonStop process pair technology to ensure that even non-persistent messages are resilient to failures. The queue servers that are responsible for the storage of messages checkpoint non-persistent messages to their backup process running in a different CPU.

Non-persistent message checkpointing is a per-queue option that you can configure using **altmqfls**. Checkpointing of non-persistent messages is enabled by default.

The TM/MP subsystem manages the complex operations for current transactions and database consistency, both user operations and MQSeries operations, making these operations transparent to both users and application programs.

A recovery restores the queue manager to the state it was in when the queue manager stopped. Any transactions that are in process are rolled back, removing from the queues any messages that were not committed at the time the queue manager stopped. Recovery restores all persistent messages; non-persistent messages are lost during the process.

The remainder of this chapter introduces the concepts of recovery and restart in more detail and explains how to recover if you experience any problems. It covers the following topics:
- “Fault tolerance and recovery”
- “Backing up and restoring MQSeries” on page 170

## Fault tolerance and recovery

If you properly configure the MQSeries Version 5.1 product and the Compaq NSK system software and hardware (for example, all components are configured as redundant or mirrored devices or process pairs as prescribed by Compaq), the failure of any single hardware or software component does not result in loss, duplication or corruption of data or the permanent loss (that is, requiring outside intervention to restore) of any function of the system. MQSeries for Compaq NSK V5.1 can recover from a single point of failure while maintaining data integrity as specified above.

Repeated consecutive failure (for example, fail-recovery looping) of the same software component is trapped once a configured maximum number of failures is exceeded. In such instances, or in the case of multiple-point failure, the MQSeries product cannot preserve queue integrity.

For more on setting up a queue manager for data integrity and availability, see “Chapter 16. Data integrity and availability” on page 211.

© Copyright IBM Corp. 1993, 2001

# Backing up and restoring MQSeries

Periodically, you might want to make a backup of your queue manager data to provide protection against possible corruption due to hardware failures.

## Backing up MQSeries

To back up a queue manager's data, you must:

1. Ensure that the queue manager is not running.

   If your queue manager is running, stop it with the **endmqm** command.

   **Note:** If you try to make a backup of a running queue manager, the backup might not be consistent due to updates in progress when the files were copied.
2. Locate the volumes and subvolumes under which the queue manager stores its data

   You can use the information in the configuration files to determine these directories. For more information, see "Chapter 13. Configuration files" on page 173.

   **Note:** If you have difficulty understanding the names that appear in the directory it is because the names are transformed to ensure that they are compatible with the platform on which you are using MQSeries. For more information about name transformations, see "Volume structure" on page 56.
3. Make copies of all the queue manager's data and log file subvolumes.

   Ensure that you do not overlook any of the files.

## Restoring MQSeries

To restore a backup of a queue manager's data, you must:

1. Ensure that the queue manager is not running.
2. Locate the subvolumes under which the queue manager stores its data. This information is located in the configuration file.
3. Empty the subvolumes into which you are going to place the backed up data.
4. Copy the backed up queue manager data into the correct places.

Check the resulting directory structure to ensure that you have all of the required directories.

Check that the MQSeries and queue manager configuration files are consistent so that MQSeries can look in the correct places for the restored data.

If the data was backed up and restored correctly, the queue manager starts.

# Recovery and restart of status servers and queue servers

The status server and queue server processes are Compaq NSK process pairs. This means that they are designed to continue to provide their services in the event of a failure of a single CPU, or of the process itself. In the case of a single failure (for example, the CPU that contains the primary status server or queue server process fails, or the primary status server or queue server process itself fails) the backup status server or queue server process takes over as the new primary without interruption of queue manager processing.

In normal single-point-of-failure situations, therefore, no recovery actions specific to the status server or queue server are required. A message is logged to the home terminal and the message log file by an status server or queue server whenever the backup has to be restarted by the primary, or the backup takes over as primary.

In the case of a more serious failure (for example, an environmental failure that prevents initialization of the primary or backup status server or queue servers) the TS/MP PATHMON process attempts to restart the status server or queue server up to 10 times.

The status server or queue server accesses only databases that are protected by TM/MP, so that in the event of failures affecting access to the disks, the protection provided by TM/MP and the DP2 disk subsystem can be relied upon.

The status server or queue server can be individually stopped and restarted by use of TS/MP PATHCOM commands if necessary, though this is not normally required. The **strmqm** command automatically starts all status server or queue server classes that have names that begin with the character string MQS-STATUS, or MQS-QUEUE. On queue manager shutdown, all active status server or queue server classes coordinate their shutdown and, when all active queue manager connections are closed, any status server or queue server involved with those connections shuts down.

Compaq NSK aborts transactions under certain circumstances on the failure of a Primary process of a NonStop process pair. This can cause the failure of MQPUT or MQGET operations in progress at the time of a failure of the Primary process. See “Chapter 16. Data integrity and availability” on page 211 for more information.

## Disaster recovery using RDF

The following procedures should be used to bring into operation a queue manager on the backup site, if a disaster makes the primary site unusable:

1. Ensure that RDF has completed updating the databases.
2. Use FUP to set the audit flag on for the following files:
   a. All files in the <qmgr>M subvolume.
   b. All files in the <qmgr>D subvolume, except QMINI, AMQRFNxx, CCSIDMEM, QMINIMEM, STATABLE, UMQSINI, PATHCTL, TRACEOPT, and SHUTDOWN.
   c. All files in the <qmgr>S subvolume.
3. Set your default volume to the <qmgr>D subvolume. Run up PATHMON manually, run PATHCOM against it, and load the PATHWAY configuration for the queue manager.
4. Perform an INFO command on all objects.
   a. Verify that all instances of the node name appear either as ″*″ **or** have the correct node name for the backup system.
   b. Verify that the CPU numbers assigned to the server classes are still valid for this backup site.
   c. Verify that the Home Terminal and Out file names are valid for the backup site. If they are not, change them.
   d. Verify that any alternate TCP/IP process name specified is valid for this system.
   e. If the backup site is itself not configured for RDF operation, remove any PARAM MQRDF settings from the EC server class definitions.
   f. After verifying the PATHWAY configuration, save it back to disk using the shutdown2 command and exit from PATHCOM. If the home terminal name

has been changed, modify the QMINI file for the queue manager to match the PATHWAY configuration. If necessary, change the TCP/IP listener ports configuration in the QMINI file.

g. Use **strmqm** to start the queue manager.
h. Using **runmqsc** verify the channel configuration, and adjust if necessary.
i. If you attempt to bring up the same channels as were running before, the channel configuration on the remote queue managers might also have to be changed unless the backup system can be reconfigured to use the same IP address of hostname, for TCP/IP channels, or the same SNAX/APC and ICE resource names (for example, process name, LU names, and so on) for SNA channels.
j. Be prepared for channel synchronization or sequence errors, particularly if the primary site channels were running at the time of the disaster. RDF does not ensure that the databases on the backup site are up to date (in lockstep with the primary) so data can be lost as a result of a complete disaster. To minimize the chances of this, ensure that your RDF configuration can handle the volume of database updates associated with your message flow.

# Chapter 13. Configuration files

MQSeries for Compaq NSK uses *configuration files* to hold basic product configuration information. This chapter describes what configuration files are and how you can use them to change the way that queue managers operate. It contains the following sections:

- "What are configuration files?"
- "MQSeries configuration file (MQSINI)"
- "Queue manager configuration file (QMINI)" on page 174
- "Editing configuration files" on page 182

## What are configuration files?

Configuration files define optional values for individual queue managers and for MQSeries on the node as a whole. These files are referred to as *ini* files or *stanza* files. A configuration file contains one or more stanzas, where a stanza is a group of lines in the file that together have a common function or define part of a system. For example, there are stanzas associated with logs, channels, and installable services.

Configuration files can be modified automatically by commands that change the configuration of queue managers on the node and also by editing them manually. In general, however, configuration files should not be modified manually while queue managers are running.

There are two types of configuration file:

- The *MQSeries configuration file*, MQSINI, which specifies values for MQSeries on the node as a whole. There is normally one MQSeries configuration file per node.
- *Queue manager configuration files*, QMINI, which specify values for specific queue managers. There is one queue manager configuration file for each queue manager on the node.

## MQSeries configuration file (MQSINI)

The MQSeries configuration file, MQSINI, contains information relevant to all the queue managers on an MQSeries installation node. It is created automatically during installation. In particular, the MQSeries configuration file is used to locate the data associated with each queue manager. The MQSeries configuration file is located in the ZMQSSYS subvolume, by default $SYSTEM.ZMQSSYS.MQSINI. An environment variable, MQMACHINIFILE, is provided for use on systems where the MQSeries configuration file does not have the default name or location.

### What the MQSeries configuration file contains

The MQSINI file contains installation-wide defaults, the names of the queue managers, the name of the default queue manager, and the location of the files associated with each of them. The following stanzas can appear in MQSINI:

**AllQueueManagers**
Specifies values for installation-wide file locations and volumes.

**DefaultQueueManager**
Specifies the default queue manager for the installation. This queue

© Copyright IBM Corp. 1993, 2001

manager processes MQSC commands when a queue manager name is not explicitly specified. The stanza is automatically updated if you create a new default queue manager. If you inadvertently create a default queue manager and then want to revert to the original, you must alter this stanza manually.

**QueueManager**
There is one such stanza for each queue manager. The QueueManager stanza specifies the queue manager name and the location of the files associated with that queue manager. The names of these files are based on the queue manager name but are transformed if the queue manager name is not a valid file name.

Figure 42 shows an example MQSINI file.

```
#*******************************************************************#
#* Module Name: MQSINI                                             *#
#* Type:        MQSeries machine-wide ini file                     *#
#* Function:    Define configuration data for all queue managers   *#
#*                                                                 *#
#*******************************************************************#
#* Notes :                                                         *#
#* 1) This file defines configuration data for all queue managers  *#
#*                                                                 *#
#*******************************************************************#
AllQueueManagers:
    MQSVolume=$DATA00             /Volume for the installation
    MQSExePath=$DATA00.ZMQSEXE    /Location of product executables
    QMDefaultVolume=$DATA00       /Default volume for queue manager creation
    ConvEBCDICNewline=NL_TO_LF    /Data Conversion EBCDIC Newline
    NSKSegidRange=10-20           /Segment Id Range
QueueManager:
    Name=MT01                     /A queue manager called MT01
    QMVolume=$DATA00              /Volume of the queue manager
    QMSubvolume=MT01              /Subvolume prefix for the queue manager
DefaultQueueManager:
    Name=MT01                     /Name of the default queue manager (optional)
```

*Figure 42. Example MQSeries configuration file (MQSINI).* The MQSINI file is initialized during installation with the volume and subvolume information you provide.

**Note:** Because the MQSeries configuration file is used to locate the data associated with queue managers, a nonexistent or incorrect configuration file can cause some or all MQSeries commands to fail. Also, applications cannot connect to a queue manager that is not defined in the MQSeries configuration file.

## Queue manager configuration file (QMINI)

A queue manager configuration file, QMINI, contains information relevant to a specific queue manager. There is one queue manager configuration file for each queue manager. It is created automatically when the queue manager with which it is associated is created.

The file is held in the subvolume of the queue manager. For example, the path and name for a configuration file for a queue manager called QMNAME could be `$VOLUME.QMNAMED.QMINI`.

**Note:** The queue manager name can be up to 48 characters in length. A subvolume name is generated based on the queue manager name. This process is known as *name transformation*, and ensures the name is both valid and unique.

## What the queue manager configuration file contains

The stanzas that can appear in a queue manager configuration file, QMINI, are as follows:

**Configuration**

This stanza defines the global configurations for the queue manager.

The following entries can be modified:

```
DefaultQueueServerName
DefaultStatusServerName
HomeTerminalName
PathmonProcName
EMSCollectorName
MinIdleMCALU62Responders
MinIdleMCATCPResponders
MinIdleMCACallers
MinIdleLQMAgents
```

For more information about these entries, see “Modifying queue manager properties” on page 44. Other entries in this stanza must not be changed.

**DefaultProcess**

This stanza defines the default values used for MQSeries processes. Entries in this stanza must not be changed.

**ECBoss**

This stanza defines the configuration of the MQSeries EC Boss process. The `ExpectedNumECs` entry defines the number of EC processes for this queue manager. This value must correspond with the PATHWAY configuration for the queue manager. For more information, see “Modifying queue manager properties” on page 44. Other entries in this stanza must not be changed.

**EC** The `MCAAgentPriority` and `LQMAgentPriority` entries of the EC stanza, which control the process priorities of agent processes, can be modified. For more information, see “Modifying queue manager properties” on page 44. Other entries in this stanza must not be modified.

The following stanzas define the specific operating parameters for each MQSeries process type. Typically, you do not need to change the values of these parameters. However, see “Example queue manager configuration file” on page 177.

**MCACaller**

**MCATCPResponder**

**MCALU62Responder**

**MQIServer**

**LQMAgent**

**ChannelInitiator**

**TCPListener**

## Queue manager configuration file

**Authority**

Provides the recommended mechanism for enabling and disabling the OAM for a queue manager. Set the MQAUTH flag to On or Off to enable or disable the OAM without having to add and remove the Service and Service Component stanzas.

**Service**

Specifies the name of one of the installable services, and the number of entry points to that service. There is one stanza for each service. These services are available:

- Authorization service
- Name service

The Object Authority Manager (OAM) is enabled by default: the authorization service stanza and its associated `ServiceComponent` stanza are present in QMINI by default.

You can disable the OAM simply by setting the MQAUTH flag in the Authority stanza to Off and restarting the queue manager. Alternatively, you can:

1. Delete the queue manager (using the **dltmqm** command)
2. Create the queue manager again (using the **crtmqm** command) with the MQSNOAUT environment variable set.
3. Delete the authorization service stanzas from QMINI.

The name service stanza must be added manually to QMINI if you want to enable the supplied name service.

**ServiceComponent**

These stanzas define the service component associated with a particular service. There can be more than one service component stanza for each service, but each service component stanza must match the corresponding service stanza. See the *MQSeries Programmable System Management* book for more information.

**TuningParameters**

This stanza defines internal tuning parameters used by the local queue manager agents. You should not change these values.

**Channels**

This stanza contains information about the channels. As well as defining the maximum number of channels (`MaxChannels`) that can be defined for the queue manager, a second entry (`MaxActiveChannels`) limits the number of channels that can be active simultaneously. MaxActiveChannels must not be greater than MaxChannels. The channels stanza also contains an entry (`ChanInitDiscInterval`) that can be used to tune the performance of the channel initiator. For more information about these entries, see "Modifying queue manager properties" on page 44. Other entries in this stanza must not be modified.

See the *MQSeries Intercommunication* book for more information about channels.

**TCPConfig**

Specifies network-protocol configuration parameters. These stanzas override the default parameters for channels. Only stanzas representing changed default values are actually present.

The TCPListenerPort values are overridden by the listener program if the parameter MQLISTENPORTNUM is present in the environment of the listener process.

See the *MQSeries Intercommunication* book for more information.

For information about modifying the `TCPPort`, `TCPNumListenerPorts`, and `TCPListenerPort` entries, see "Modifying queue manager properties" on page 44.

## Example queue manager configuration file

Figure 43 shows a sample queue manager configuration file (QMINI).

```
#*******************************************************************#
#* Module Name: QMINI                                              *#
#* Type       : MQSeries queue manager configuration file          *#
#  Function   : Define the configuration of a single queue manager *#
#*                                                                 *#
#*******************************************************************#
#* Notes      :                                                    *#
#* 1) This file defines the configuration of the queue manager     *#
#*                                                                 *#
#*******************************************************************#
Configuration:
   PathmonProcName=$p01p
   DefaultStatusServerName=$p01s
   ServerClassName=MQS-ECBOSS
   EMSCollectorName=$0
   HomeTerminalName=$ZTN0.#PTY001C
   ShutdownFileName=SHUTDOWN
   TraceOptionsFileName=TRACEOPT
   RuntimeFileName=RUNTIME
   StatableFileName=STATABLE
   ChannelDefFileName=CHDEFS
   DefaultCCSID=819
   DefaultTraceOptions=0
   MaxIdleAgents=10
   MinIdleMCALU62Responders=0
   MinIdleMCATCPResponders=0
   MinIdleMCACallers=0
   MinIdleLQMAgents=1
   MaxIdleAgentReuse=10
DefaultProcess:
   ExeFileName=DEFAULT
   TraceVolSubvol=$DATA1.p101L
   TracePrefix=TR
   ErrorVolSubvol=$DATA1.p101L
   ErrorPrefix=ER
   DebugMode=0
   IPCCTimeOut=10000
   IPCCMemSetSize=32000
   MemSetSize=16000
   ExtPoolSize=256000
   IniPoolSize=256000
   Priority=175
```

*Figure 43. Example queue manager configuration file (QMINI) (Part 1 of 6)*

```
ECBoss:
   ExeFileName=MQECBOSS
   TraceVolSubvol=$DATA1.p101L
   TracePrefix=TR
   ErrorVolSubvol=$DATA1.p101L
   ErrorPrefix=ER
   DebugMode=0
   IPCCTimeOut=10000
   IPCCMemSetSize=32000
   MemSetSize=16000
   ExtPoolSize=300000
   IniPoolSize=256000
   Priority=175
   ExpectedNumECs=1
EC:
   ExeFileName=MQEC
   TraceVolSubvol=$DATA1.p101L
   TracePrefix=TR
   ErrorVolSubvol=$DATA1.p101L
   ErrorPrefix=ER
   DebugMode=0
   IPCCTimeOut=10000
   IPCCMemSetSize=32000
   MemSetSize=16000
   ExtPoolSize=256000
   IniPoolSize=256000
   Priority=175
   LQMAgentExe=MQLQMAG
   MCACallerExe=MQMCACAL
   MCATCPResponderExe=MQTCPRES
   MCALU62ResponderExe=MQLU6RES
   MCAAgentPriority=165
   LQMAgentPriority=165
   StopProcessTimer=3000
   IdleProcessTimer=3000
```

*Figure 43. Example queue manager configuration file (QMINI) (Part 2 of 6)*

```
MCACaller:
   ExeFileName=MQMCACAL
   TraceVolSubvol=$DATA1.p101L
   TracePrefix=TR
   ErrorVolSubvol=$DATA1.p101L
   ErrorPrefix=ER
   DebugMode=0
   IPCCTimeOut=10000
   IPCCMemSetSize=32000
   MemSetSize=16000
   ExtPoolSize=300000
   IniPoolSize=256000
   Priority=175
MCATCPResponder:
   ExeFileName=MQTCPRES
   TraceVolSubvol=$DATA1.p101L
   TracePrefix=TR
   ErrorVolSubvol=$DATA1.p101L
   ErrorPrefix=ER
   DebugMode=0
   IPCCTimeOut=10000
   IPCCMemSetSize=32000
   MemSetSize=16000
   ExtPoolSize=300000
   IniPoolSize=256000
   Priority=175
MCALU62Responder:
   ExeFileName=MQLU6RES
   TraceVolSubvol=$DATA1.p101L
   TracePrefix=TR
   ErrorVolSubvol=$DATA1.p101L
   ErrorPrefix=ER
   DebugMode=0
   IPCCTimeOut=10000
   IPCCMemSetSize=32000
   MemSetSize=16000
   ExtPoolSize=300000
   IniPoolSize=256000
   Priority=175
```

*Figure 43. Example queue manager configuration file (QMINI) (Part 3 of 6)*

## Queue manager configuration file

```
MQIServer:
   ExeFileName=MQMQISER
   TraceVolSubvol=$DATA1.p101L
   TracePrefix=TR
   ErrorVolSubvol=$DATA1.p101L
   ErrorPrefix=ER
   DebugMode=0
   IPCCTimeOut=10000
   IPCCMemSetSize=32000
   MemSetSize=16000
   ExtPoolSize=300000
   IniPoolSize=256000
   Priority=175
LQMAgent:
   ExeFileName=MQLQMAG
   TraceVolSubvol=$DATA1.p101L
   TracePrefix=TR
   ErrorVolSubvol=$DATA1.p101L
   ErrorPrefix=ER
   DebugMode=0
   IPCCTimeOut=50
   IPCCMemSetSize=32000
   MemSetSize=16000
   ExtPoolSize=120000
   IniPoolSize=200000
   Priority=175
ChannelInitiator:
   ExeFileName=RUNMQCHI
   TraceVolSubvol=$DATA1.p101L
   TracePrefix=TR
   ErrorVolSubvol=$DATA1.p101L
   ErrorPrefix=ER
   DebugMode=0
   IPCCTimeOut=10000
   IPCCMemSetSize=32000
   MemSetSize=16000
   ExtPoolSize=256000
   IniPoolSize=256000
   Priority=175
TCPListener:
   ExeFileName=RUNMQLSR
   TraceVolSubvol=$DATA1.p101L
   TracePrefix=TR
   ErrorVolSubvol=$DATA1.p101L
   ErrorPrefix=ER
   DebugMode=0
   IPCCTimeOut=10000
   IPCCMemSetSize=32000
   MemSetSize=16000
   ExtPoolSize=256000
   IniPoolSize=256000
   Priority=175
```

*Figure 43. Example queue manager configuration file (QMINI) (Part 4 of 6)*

```
 Queue Manager Server:
   ExeFileName=MQMGRSVR
   TraceVolSubvol=$DATA01.MV1L
   TracePrefix=TR
   ErrorVolSubvol=$DATA01.MV1L
   ErrorPrefix=ER
   DebugMode=0
   IPCCTimeOut=10000
   IPCCMemSetSize=32000
   MemSetSize=16000
   ExtPoolSize=300000
   IniPoolSize=256000
   Priority=175
Repository Server:
   ExeFileName=MQREPSVR
   TraceVolSubvol=$DATA01.MV1L
   TracePrefix=TR
   ErrorVolSubvol=$DATA01.MV1L
   ErrorPrefix=ER
   DebugMode=0
   IPCCTimeOut=10000
   IPCCMemSetSize=32000
   MemSetSize=16000
   ExtPoolSize=256000
   IniPoolSize=256000
   Priority=175
```

*Figure 43. Example queue manager configuration file (QMINI) (Part 5 of 6)*

```
Authority:
   MQAUTH=On
Service:
   Service=AuthorizationService
   EntryPoints=9
ServiceComponent:
   Service=AuthorizationService
   Name=MQSeries.Compaq.auth.service
   Module=MQOAM
   ComponentDataSize=0
   ComponentID=0
TuningParameters:
   KernelMemSetSize=32000
   ObjCatMemSetSize=32000
   QueueMemSetSize=16000
   MQGETActiveQPoll=50
   MQGETInactiveQPoll=1000
Channels:
   RetryAll=1
   MaxChannels=10
   MaxActiveChannels=10
   MaxTries=3
   MaxTriesInterval=10
   ChanInitDiscInterval=10
   AdoptNewMCA=NO
   AdoptNewMCATimeout=60
   AdoptNewMCACheck=NAME,ADDRESS,QM
TCPConfig:
   TCPPort=1414
   TCPNumListenerPorts=1
   TCPListenerPort=1414
   TCPKeepAlive=1
```

*Figure 43. Example queue manager configuration file (QMINI) (Part 6 of 6)*

# Editing configuration files

You can edit the default configuration files to alter the system defaults. However, before editing any configuration file, ensure that you have a backup that you can restore if necessary, and that any affected queue managers are stopped.

You might have to edit your configuration files if, for example:

- You lose a configuration file (recover from backup, if possible).
- You need to change the distribution of your queue manager across CPUs.
- You need to change your default queue manager (for example, if you accidentally delete the existing queue manager).
- You are advised to do so by your IBM Support Center.

For more information, see "Modifying queue manager properties" on page 44.

## Implementing changes to configuration files

If you edit a configuration file, the changes are not implemented immediately by the queue manager. Changes made to the MQSeries configuration file (MQSINI) take effect only when MQSeries queue managers are created or started. Changes made to a queue manager configuration file (QMINI) take effect when the queue manager is started. If the queue manager is running when you make the changes, you must stop and then restart the queue manager for any changes to be recognized by the system.

## Recommendations for configuration files

When you create a new queue manager, you should:

- Back up the MQSeries configuration file (MQSINI)
- Back up the new queue manager configuration file (QMINI)

# Chapter 14. Problem determination

This chapter provides troubleshooting information for MQSeries for Compaq NSK. To determine a problem, you should list the symptoms and then trace them back to the cause.

Performance problems caused by the limitations of your hardware cannot be solved immediately. If you believe that the cause of the problem is in the MQSeries code, contact your IBM Support Center. This chapter contains these sections:

- "Making a preliminary check"
- "Common programming errors" on page 187
- "What to do next" on page 187
- "Application design considerations" on page 190
- "Effect of message length" on page 191
- "Error logs" on page 195
- "Dead-letter queues" on page 198
- "Configuration files and problem determination" on page 198
- "Using MQSeries trace" on page 198
- "First Failure Support Technology™ (FFST)" on page 199

## Making a preliminary check

The cause of a problem can be in:

- MQSeries
- Your network
- An application
- The Compaq system software

The sections that follow provide questions that you might want to consider. Answer the questions and make a note of any issues that might be relevant to the problem.

### Has MQSeries run successfully previously?

If MQSeries has not successfully run previously, you might not have set it up correctly. See the *MQSeries for Compaq NSK Quick Beginnings* book to check that you have carried out all the steps correctly.

### Are there any error messages?

MQSeries uses error logs to capture messages concerning the operation of MQSeries itself, any queue managers that you start, and error data coming from the channels that are in use. Check the error logs for any messages have been recorded that are associated with your problem.

See "Error logs" on page 195 for information about the contents of the error logs and their locations.

### Are there any return codes explaining the problem?

If your application gets a return code indicating that a Message Queue Interface (MQI) call has failed, refer to the *MQSeries Application Programming Reference* for a description of that return code.

© Copyright IBM Corp. 1993, 2001

## Can you reproduce the problem?

If you can reproduce the problem, consider the following questions:

- Is the problem caused by a command or an equivalent administration request?

  Does the operation work if it is entered by another method? If the command works if it is entered on the command line, but not otherwise, check that the command server has not stopped, and that the queue definition of the SYSTEM.ADMIN.COMMAND.QUEUE has not been changed.
- Is the problem caused by a program?
- Can you identify any application that always seems to be running in the system when the problem occurs? If so, examine the application for errors.
- Is the volume that the queue manager database is to reside on TM/MP protected? Issue a TMFCOM; STATUS DATAVOLS to verify the volume is TM/MP protected.
- Have the required TACL environment parameters as described in "Appendix C. Setting TACL environment variables for MQSeries for Compaq NSK" on page 299 been added? Issue a PARAM at the TACL prompt to display the parameters that are currently set.
- Is the TM/MP audit trail sized to handle the load. (See "Chapter 11. Understanding transactional support and messaging" on page 165.) Issue a TMFCOM; INFO AUDITTRAIL to display the audit trail configuration.
- Does Compaq file security allow access to the qmD database files? A minimum of read access is required to the files to enable access by users outside the MQM group. A return code 2035 (MQRC_NOT_AUTHORIZED) will be returned for a MQCONN request when Compaq file security attributes preclude access.

## Have any changes been made since the last successful run?

When you are considering changes that might recently have been made, think about the MQSeries system, and also about the other programs it interfaces with, the hardware, and any new applications. Consider also the possibility that a new application that you are not aware of might have been run on the system.

- Have you changed, added, or deleted any queue definitions?
- Have you changed or added any channel definitions? Changes may have been made to either MQSeries channel definitions or any underlying communications definitions required by your application.
- Do your applications deal with return codes that they might get as a result of any changes you have made?
- Have any modification to the MQSeries PATHWAY been made?
- Have any modifications to the MQSeries Installation files been made, for example changing file security?
- Have any modifications to the MQSeries Compaq NSK database files been made, for example changing file security or altering TM/MP audit?
- Has the queue manager QMINI file had changes applied?

## Has the application run successfully before?

If the problem appears to involve one particular application, consider whether the application has run successfully before.

Consider the following questions:

- Have any changes been made to the application since it last ran successfully?

If so, can the error exist in the new or modified part of the application. Check the changes and see if you can find an obvious reason for the problem. Is it possible to retry using a back level of the application?

- Have all the functions of the application been fully exercised previously?

  Does the problem occur when part of the application that has never been invoked before is used for the first time? If so, the error might exist in that part of the application. Analyze what the application was doing when it failed, and check the source code in that part of the program for errors.

  If a program has run successfully on previous occasions, check the current queue status, and the files that were being processed when the error occurred. It is possible that they contain some unusual data value that causes a rarely used path in the program to be invoked?
- Does the application check all return codes?

  Has your MQSeries system been changed, such that your application does not check the return codes it receives as a result of the change. For example, does your application assume that the queues it accesses can be shared? If a queue has been redefined as exclusive, can your application deal with return codes indicating that it can no longer access that queue?
- Does the application run on other MQSeries systems?

  Is there a difference in the way that this MQSeries system is set up which is causing the problem? For example, have the queues been defined with the same message length or priority?
- Have you set PARAM SAVE-ENVIRONMENT ON?

  If not, you will receive MQRC 2058 on MQCONN calls. Set the PARAM in your application environment, as described in “Appendix C. Setting TACL environment variables for MQSeries for Compaq NSK” on page 299).

## If the application has not run successfully previously

If your application has not yet run successfully, you should examine it carefully for any errors.

Before you look at the code, and depending upon which programming language the code is written in, examine the output from the translator, or the compiler and linkage editor, if applicable, to see if any errors are reported.

If your application fails to translate, compile, or link-edit into the load library, it cannot run. See the *MQSeries Application Programming Reference* for information about building your application.

If the documentation shows that each of these steps was accomplished without error, you should consider the coding logic of the application. Do the symptoms of the problem indicate the function that is failing and, therefore, the piece of code in error? See “Common programming errors” on page 187 for some examples of common errors that cause problems with MQSeries applications.

## Does the problem affect specific parts of the network?

You might be able to identify specific parts of the network that are affected by the problem (remote queues, for example). If the link to a remote message queue manager is not working, the messages cannot flow to a remote queue.

Check that the connection between the two systems is available, and that the intercommunication component of MQSeries has been started.

### Preliminary checks

Check that messages are reaching the transmission queue, and check the local queue definition of the transmission queue and any remote queues.

Have you made any network-related changes, or changed any MQSeries definitions, that might account for the problem?

Check that the MQSeries PATHWAY TCP Listener server MQS-TCPLISxx is started. If the server is failing to start, check that there are no ending connections (for example a FIN-WAIT status) for the port using SCF (for example SCF; STATUS PROCESS $ZTC0). The remote partner needs to be stopped and restarted to release the port. For SNA, check that the MQSeries SNA Listener is configured for the SNAX/APC or ICE PATHWAY, see “LU 6.2 responder processes” on page 352.

Check that the correct TCPIP process name, hostname or ipaddress, and port is used for the connection name for channels and corresponds with the remote channel definition.

## Does the problem occur at specific times of the day?

If the problem occurs at specific times of day, it could be that it is dependent on system loading. Typically, peak system loading is at mid-morning and mid-afternoon, so these are the times when load-dependent problems are most likely to occur. (If your MQSeries network extends across more than one time zone, peak system loading might seem to occur at some other time of day.)

## Is the problem intermittent?

An intermittent problem could be caused by failing to take into account the fact that processes can run independently of each other. For example, a program may issue an MQGET call, without specifying a wait option, before an earlier process has completed. An intermittent problem may also be seen if your application tries to get a message from a queue while the call that put the message is in-doubt (that is, before it has been committed or backed out).

Occassionly, PATHWAY errors may be logged while ENDMQM is executing. The error: *1018* SERVER FILE (6006) or *1018* SERVER FILE (7006) may be displayed while the queue manager is shutting down. These are expected while the MQS-QMGRSVR00, MQS-STATUS00 or MQS-QUEUE00 servers are ending.

## Have you applied any service updates?

If a service update has been applied to MQSeries, check that the update action completed successfully and that no error message was produced.

- Did the update have any special instructions?
- Was any test run to verify that the update had been applied correctly and completely?
- Does the problem still exist if MQSeries is restored to the previous service level?
- If the installation was successful, check with the IBM Support Center for any patch error.
- If a patch has been applied to any other program, consider the effect it might have on the way MQSeries interfaces with it.
- Verify the service level. Edit ZMQSSYS.MEMOPTF read only. Note the entry A) for CSD HISTORY. Enter at the TACL prompt VPROC $vol.ZMQSLIB.MQSRLLIB where $vol is your MQ installation volume. It should

match the VPROC information of the MEMOPTF or may be newer if an efix has been applied. Refer to the MEMOEFIX file provided with the efix for the updated VPROC information.

## Common programming errors

The errors in the following list illustrate the most common causes of problems encountered while running MQSeries programs. You should consider the possibility that the problem with your MQSeries system could be caused by one or more of these errors:

- Assuming that queues can be shared, when they are in fact exclusive.
- Passing incorrect parameters in an MQI call.
- Passing insufficient parameters in an MQI call. This may mean that MQI cannot set up completion and reason codes for your application to process.
- Failing to check return codes from MQI requests.
- Passing variables with incorrect lengths specified.
- Passing parameters in the wrong order.
- Failing to initialize *MsgId* and *CorrelId* correctly.
- Failing to issue BEGINTRANSACTION when MQPMO_SYNCPOINT is specified on the MQPUT command.

### Problems with commands

You should be careful when including special characters, such as back slash (\) and double quotation marks (”), in descriptive text for some commands. If you use either of these characters in descriptive text, precede them with a \. That is, enter \\ or \” if you want \ or ” in your text.

## What to do next

When you have established that no changes have been made to your system, and that there are no problems with your application programs, choose the option that best describes the symptoms of your problem.

- “Have you obtained incorrect output?”
- “Have you failed to receive a response from a PCF command?” on page 188
- “Does the problem affect only remote queues?” on page 189
- “Is your application or MQSeries for Compaq NSK running slowly?” on page 190

### Have you obtained incorrect output?

In this book, “incorrect output” refers to your application:

- Not receiving a message that it was expecting.
- Receiving a message containing unexpected or corrupted information.
- Receiving a message that it was not expecting, for example, one that was destined for a different application.
- Is the structure you are using to display the information correct? For example is the MQDLH structure that is used to display the dead letter queue header data added to the beginning of the message text for a message on the dead letter queue?
- Is it reproducible on a MQSeries installation on another machine at a different operating system (OS) level or same level? If not reproducible there may be an OS error that is corrected with a Interim Program Maintenance (IPM) from Compaq that needs to be applied. Check with Compaq support.

### What next

In all cases, check that any queue or queue manager aliases that your applications are using are correctly specified and accommodate any changes that have been made to your network.

If an MQSeries error message is generated, all of which are prefixed with the letters "AMQ", you should look in the error log. See "Error logs" on page 195 for further information.

## Have you failed to receive a response from a PCF command?

If you have issued a command but you have not received a response, consider the following questions:

- Is the command server running?

  Work with the **dspmqcsv** command to check the status of the command server. If the response to this command indicates that the command server is not running, use the **strmqcsv** command to start it. If the response to the command indicates that the SYSTEM.ADMIN.COMMAND.QUEUE is not enabled for MQGET requests, enable the queue for MQGET requests.
- Has a reply been sent to the dead-letter queue?

  The dead-letter queue header structure contains a reason or feedback code describing the problem. See the *MQSeries Application Programming Reference* for information about the dead-letter queue header structure (MQDLH).

  If the dead-letter queue contains messages, you can use the supplied browse sample application (AMQSBCG) to browse the messages using the MQGET call. The sample application steps through all the messages on a named queue for a named queue manager, displaying both the message descriptor and the message context fields for all the messages on the named queue.
- Has a message been sent to the error log?

  See "Error logs" on page 195 for further information.
- Are the queues enabled for put and get operations?
- Is the *WaitInterval* long enough?

  If your MQGET call has timed out, a completion code of MQCC_FAILED and a reason code of MQRC_NO_MSG_AVAILABLE are returned. (See the *MQSeries Application Programming Reference* for information about the *WaitInterval* field, and completion and reason codes from MQGET.)
- If you are using your own application program to put commands onto the SYSTEM.ADMIN.COMMAND.QUEUE, do you need to commit a transaction?

  Unless you have specifically excluded your request message from syncpoint, you need to commit a transaction before attempting to receive reply messages.
- Are the MAXDEPTH and MAXMSGL attributes of your queues set sufficiently high?
- Are you using the *CorrelId* and *MsgId* fields correctly?

  Set the values of *MsgId* and *CorrelId* in your application to ensure that you receive all messages from the queue.

Try stopping the command server and then restarting it, responding to any error messages that are produced.

If the system still does not respond, the problem could be with the queue manager. Try stopping the queue manager and then restarting. If the problem still occurs after restart, contact your IBM Support Center for help.

## Are some of your queues failing?

If you suspect that the problem occurs with only a subset of queues, check the local queues that you think are having problems:

1. Display the information about each queue. You can use the MQSC command DISPLAY QUEUE to display the information.
2. Use the data displayed to do the following checks:
   - If CURDEPTH is at MAXDEPTH, this indicates that the queue is not being processed. Check that all applications are running normally.
   - If CURDEPTH is not at MAXDEPTH, check the following queue attributes to ensure that they are correct:
     - If triggering is being used:
       - Is the trigger monitor running?
       - Is the trigger depth too great? That is, does it generate a trigger event often enough?
       - Is the process name correct?
       - Is the process available and operational?
     - Can the queue be shared? If not, another application could already have it open for input.
     - Is the queue enabled appropriately for GET and PUT?
   - If there are no application processes getting messages from the queue, determine why this is so. It could be because the applications need to be started, a connection has been disrupted, or the MQOPEN call has failed for some reason.

     Check the queue attributes IPPROCS and OPPROCS. These attributes indicate whether the queue has been opened for input and output. If a value is zero, it indicates that no operations of that type can occur. Note that the values may have changed and that the queue was open but is now closed.

     You need to check the status at the time you expect to put or get a message.

If you are unable to solve the problem, contact your IBM Support Center for help.

## Does the problem affect only remote queues?

If the problem affects only remote queues, check the following:

- Check that required channels have been started and are triggerable, and that any required initiators are running.
- Check that the programs that should be putting messages to the remote queues have not reported problems.
- If you use triggering to start the distributed queuing process, check that the transmission queue has triggering set on. Also, check that the trigger monitor is running.
- Check the error logs for messages indicating channel errors or problems.
- If necessary, start the channel manually. See the *MQSeries Intercommunication* book for information about how to do this.

For information about how to define channels, see “Appendix M. Setting up communications” on page 351 and the *MQSeries Intercommunication* book.

## Is your application or MQSeries for Compaq NSK running slowly?

If your application is running slowly, this could indicate that it is in a loop, or waiting for a resource that is not available.

Has MQSeries tracing been enabled using either **strmqtrc** or the MQMC queue manager panel. This will cause a performance degradation. Check if any TR files are open in the qmL subvolume. Disable tracing using either **endmqtrc** or MQMC queue manager panel.

This could also be caused by a performance problem. Perhaps it is because your system is operating near the limits of its capacity. This type of problem is probably worst at peak system load times, typically at mid-morning and mid-afternoon. (If your network extends across more than one time zone, peak system load might seem to occur at some other time.)

Examine the priority of application and queue manager processes using the STATUS command. A loop causes the priority of the process to be reduced gradually to zero by NSK.

Check that each of the CPUs in the NSK system is being utilized fully. If some processors are only lightly loaded, your NSK system needs balancing. Consider adding ECs to other processors to distribute MQSeries workload.

A performance problem may be caused by a limitation of your hardware.

**Note:** After a fresh install of MQSeries or a cold load of the Compaq NSK system, MQSeries executables might take longer to run than expected when they are first invoked. This is because the Compaq NSK operating system goes through a “fixup” phase, during which it ensures that all external declarations are resolved.

If you find that performance degradation is not dependent on system loading, but happens sometimes when the system is lightly loaded, a poorly designed application program is probably to blame. This could manifest itself as a problem that only occurs when certain queues are accessed.

The following symptoms might indicate that MQSeries is running slowly:

- Your system is slow to respond to MQSeries commands.
- Repeated displays of the queue depth indicate that the queue is being processed slowly for an application with which you would expect a large amount of queue activity.

If the performance of your system is still degraded after reviewing the above possible causes, the problem may lie with MQSeries for Compaq NSK itself. If you suspect this, you need to contact your IBM Support Center for assistance.

# Application design considerations

There are a number of ways in which poor program design can affect performance. These can be difficult to detect because the program can appear to perform well, while impacting the performance of other tasks. Several problems specific to programs making MQSeries calls are discussed in the following sections.

For more information about application design, see the *MQSeries Application Programming Guide*.

## Effect of message length

Although MQSeries allows messages to hold up to 100 MB of data, the amount of data in a message affects the performance of the application that processes the message. To achieve the best performance from your application, you should send only the essential data in a message; for example, in a request to debit a bank account, the only information that may need to be passed from the client to the server application is the account number and the amount of the debit.

## Searching for a particular message

The MQGET call usually retrieves the first message from a queue. If you use the message and correlation identifiers (MsgId and CorrelId) in the message descriptor to specify a particular message, the queue manager has to search the queue until it finds that message. Using the MQGET call in this way affects the performance of your application.

## Queues that contain messages of different lengths

If the messages on a queue are of different lengths, to determine the size of a message, your application could use the MQGET call with the *BufferLength* field set to zero so that, even though the call fails, it returns the size of the message data. The application could then repeat the call, specifying the identifier of the message it measured in its first call and a buffer of the correct size. However, if there are other applications serving the same queue, you might find that the performance of your application is reduced because its second MQGET call spends time searching for a message that another application has retrieved in the time between your two calls.

If your application cannot use messages of a fixed length, another solution to this problem is to use the MQINQ call to find the maximum size of messages that the queue can accept, then use this value in your MQGET call. The maximum size of messages for a queue is stored in the *MaxMsgLength* attribute of the queue. This method could use large amounts of storage, however, because the value of this queue attribute could be as high as 100 MB, the maximum allowed by MQSeries for Compaq NSK.

## Frequency of syncpoints

Programs that issue numerous MQPUT calls within syncpoint, without committing them, can cause performance problems. Affected queues can fill up with messages that are currently inaccessible, while other tasks might be waiting to get these messages. This has implications in terms of: storage; TMF audit trail usage; and processes tied up with tasks that are attempting to get messages.

## Use of the MQPUT1 call

Use the MQPUT1 call if you have only a single message to put on a queue. If you want to put more than one message, use the MQOPEN call, followed by a series of MQPUT calls and a single MQCLOSE call.

# Incorrect output

The term “incorrect output” can be interpreted in many different ways. For the purpose of problem determination within this book, the meaning is explained in “Have you obtained incorrect output?” on page 187.

Two types of incorrect output are discussed in this section:

- Messages that do not appear when you are expecting them
- Messages that contain the wrong information, or information that has been corrupted

Additional problems that you might find if your application includes the use of distributed queues are also discussed.

## Messages that do not appear on the queue

If messages do not appear when you are expecting them, check for the following:

- Has the message been put on the queue successfully?
- Has the queue been defined correctly. For example, is MAXMSGL sufficiently large?
- Is the queue enabled for putting?
- Is the queue already full? This could mean that an application was unable to put the required message on the queue.
- Are you able to get any messages from the queue?
- Do you need to take a syncpoint?

  If messages are being put or retrieved within syncpoint, they are not available to other tasks until the unit of recovery has been committed.
- Is your wait interval long enough?

  You can set the wait interval as an option for the MQGET call. You should ensure that you are waiting long enough for a response.
- Are you waiting for a specific message that is identified by a message or correlation identifier (*MsgId* or *CorrelId*)?

  Check that you are waiting for a message with the correct *MsgId* or *CorrelId*. A successful MQGET call sets both these values to that of the message retrieved, so you may need to reset these values in order to get another message successfully.

  Also, check whether you can get other messages from the queue.
- Can other applications get messages from the queue?
- Was the message you are expecting defined as persistent?

  If not, and MQSeries has been restarted, the message has been lost.
- Has another application got exclusive access to the queue?

If you are unable to find anything wrong with the queue, and MQSeries is running, make the following checks on the process that you expected to put the message on to the queue:

- Did the application get started?

  If it should have been triggered, check that the correct trigger options were specified.
- Did the application stop?
- Is a trigger monitor running?
- Was the trigger process defined correctly?
- Did the application complete correctly?

Look for evidence of an abnormal end in the job log.

- Did the application commit its changes, or were they backed out?

If multiple transactions are serving the queue, they can conflict with one another. For example, suppose one transaction issues an MQGET call with a buffer length of zero to find out the length of the message, and then issues a specific MQGET call specifying the *MsgId* of that message. However, in the meantime, another transaction issues a successful MQGET call for that message, so the first application receives a reason code of MQRC_NO_MSG_AVAILABLE. Applications that are expected to run in a multi-server environment must be designed to cope with this situation.

Consider that the message could have been received, but that your application failed to process it in some way. For example, did an error in the expected format of the message cause your program to reject it? If this is the case, refer to "Messages that contain unexpected or corrupted information".

## Messages that contain unexpected or corrupted information

If the information contained in the message is not what your application was expecting, or has been corrupted in some way, consider the following points:

- Has your application, or the application that put the message onto the queue, changed?

  Ensure that all changes are simultaneously reflected on all systems that need to be aware of the change.

  For example, the format of the message data may have been changed, in which case, both applications must be recompiled to pick up the changes. If one application has not been recompiled, the data will appear corrupt to the other.
- Is an application sending messages to the wrong queue?

  Check that the messages your application is receiving are not really intended for an application servicing a different queue. If necessary, change your security definitions to prevent unauthorized applications from putting messages on to the wrong queues.

  If your application has used an alias queue, check that the alias points to the correct queue.
- Has the trigger information been specified correctly for this queue?

  Check that your application should have been started; or should a different application have been started?

If these checks do not enable you to solve the problem, you should check your application logic, both for the program sending the message, and for the program receiving it.

## Problems with incorrect output when using distributed queues

If your application uses distributed queues, you should also consider the following points:

- Has MQSeries been correctly installed on both the sending and receiving systems, and correctly configured for distributed queuing?
- Are the links available between the two systems?

  Check that both systems are available, and connected to MQSeries. Check that the connection between the two systems, and the channels between the two queue managers, are active.
- Is triggering set on in the sending system?

### Incorrect output

- Is the message you are waiting for a reply message from a remote system?

  Check that triggering is activated in the remote system.
- Is the queue already full?

  This could mean that an application was unable to put the required message onto the queue. If this is so, check if the message has been put onto the dead-letter queue.

  The dead-letter queue header contains a reason or feedback code explaining why the message could not be put onto the target queue. See the *MQSeries Application Programming Reference* book for information about the dead-letter queue header structure.
- Is there a mismatch between the sending and receiving queue managers?

  For example, the message length could be longer than the receiving queue manager can handle.
- Are the channel definitions of the sending and receiving channels compatible?

  For example, a mismatch in sequence number wrap stops the distributed queuing component. See the *MQSeries Intercommunication* book for more information about distributed queuing.
- Have you started a TCP/IP listener?

  If you are using TCP/IP as a communications protocol for MQSeries communications to the Compaq, a TCP/IP listener process must be running. See "Specifying and controlling TCP/IP listeners" on page 30 for more information.
- Is the TCP/IP listener listening on the correct TCP/IP port?

  The TCP/IP listener listens on a port defined on a `TCPListenerPort` entry in the `TCPConfig` stanza of the QMINI file for your queue manager. See "TCP/IP ports listened on by the queue manager" on page 49 for more information.
- Is the TCP/IP process name correct ?

  If you are using the TCP/IP communications protocol, is your Compaq system using the default process name ($ztc0) for the TCP/IP process? If not, you must alter some of the server classes in your MQSeries pathway to enable the correct process name to be used by MQSeries channels. See "Reconfiguring the MQS-TCPLISnn server class for a nondefault TCP/IP process and port" on page 50 for more information.
- Is the MQSeries SNA Listener configured?

  If:

  - You are running MQSeries channels using SNA as a communications protocol and
  - The channel type on Compaq is one that is waiting to be initiated from a remote MQSeries system (for example, a RECEIVER) and
  - The remote system is having problems starting the channel.

  the PATHWAY MQSeries SNA Listener might not be running for your queue manager. Check that MQSeries SNA Listener has been configured for the queue manager SNAX/APC or ICE PATHWAY. See "LU 6.2 responder processes" on page 352 for more information.
- Is data conversion involved? If the data formats between the sending and receiving applications differ, data conversion is necessary. Automatic conversion occurs when the MQGET is issued if the format is recognized as one of the built-in formats.

  If the data format is not recognized as a built-in format, a data conversion exit can be used to allow you to perform the translation with your own routines. Check that your routine is being loaded correctly.

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

# Error logs

MQSeries for Compaq NSK uses a number of error logs to capture messages concerning the operation of MQSeries itself, any queue managers that you start, and error data coming from the channels that are in use.

The location of the error logs depends on whether the queue manager name is known and whether the error is associated with a client.

- If the queue manager name is known and the queue manager is available:

```
<QMVOL>.<SUBVOL>L.MQERRLG1
```

- If the queue manager is not available:

```
<MQSVOL>.ZMQSSYS.MQERRLG1
```

- First Failure Symptom Trap (FFST) in

```
<QMVOL>.<SUBVOL>.FDnnnnn
```

- See “How to examine the FFSTs” on page 199.

## Log files

The error log subvolume can contain up to three error log files named:

- MQERRLG1
- MQERRLG2
- MQERRLG3

After you have created a queue manager, three error log files are created when they are needed by the queue manager. These files are called MQERRLG1, MQERRLG2, and MQERRLG3, and are placed in the subvolume of each queue manager that you create.

As error or log messages are generated they are placed in MQERRLG1. When MQERRLG1 is filled it is copied to MQERRLG2. Before the copy, MQERRLG2 is copied to MQERRLG3. The previous contents, if any, of MQERRLG3 are discarded.

The latest error messages are thus always placed in MQERRLG1, the other files being used to maintain a history of error messages.

All messages relating to channels are also placed in the appropriate queue manager’s errors files unless the name of their queue manager is unknown or the queue manager is unavailable. When the queue manager name is unavailable or its name cannot be determined, channel-related messages are placed in the system error log (ZMQSSYS.MQERRLG1).

To examine the contents of any error log file, you can use either the fup copy command or your usual Compaq NSK editor in read-only mode. (If you open the error log in update mode, error messages might be lost.)

## Early errors

There are a number of special cases where the above error logs have not yet been established and an error occurs. MQSeries attempts to record any such errors in an error log. The location of the log depends on how much of a queue manager has been established.

If, due to a corrupt configuration file for example, no location information can be determined, errors are logged to an error file that is created at installation time on the ZMQSSYS subvolume in the file MQERRLG1.

For further information about configuration files, see "Chapter 13. Configuration files" on page 173.

## Operator messages

In MQSeries for Compaq NSK, operator messages identify normal errors, typically caused directly by users doing things like using parameters that are not valid on a command. These messages are written to the associated window, if any, and are also written to a file in the queue manager subvolume.

Errors that can be associated with a particular queue manager are logged to MQERRLG1 in the queue manager's log subvolume. Those that cannot be linked to a defined and operational queue manager are logged in the MQERRLG1 file located in subvolume ZMQSSYS.

### Deciphering EC numbers in the MQERRLG file

The EC number in the messages logged in the MQERRLG1 file is a number that is assigned by the ECBOSS for its tracking of the EC process. There is no direct correlation between of the number assigned by the ECBOSS in the MQERRLG1 and number used in the name of the MQS-ECxx PATHWAY server. For example, MQS-EC00 may not get assigned EC number 0. The EC number assigned is associated with the EC process name in the initialization complete message. Use the EC process name for determining the MQS-ECxx PATHWAY server and for problem analysis.

## Example error log

This example shows part of an MQSeries for Compaq NSK error log:

```
...
02/01/01  11:41:56 AMQ8003: MQSeries queue manager started.
EXPLANATION: MQSeries queue manager janet started.
ACTION: None.
-------------------------------------------------------------------
02/01/01  11:56:52 AMQ9002: Channel program started.
EXPLANATION: Channel program 'JANET' started.
ACTION: None.
-------------------------------------------------------------------
02/01/01  11:57:26 AMQ9208: Error on receive from host 'camelot
(9.20.12.34)'.
EXPLANATION: An error occurred receiving data from 'camelot
(9.20.12.34)' over TCP/IP. This may be due to a communications failure.
ACTION: Record the TCP/IP return code 232 (X'E8') and tell the
systems administrator.
---------------------------------------------------------------
02/01/01  11:57:27 AMQ9999: Channel program ended abnormally.
EXPLANATION: Channel program 'JANET' ended abnormally.
ACTION: Look at previous error messages for channel program
'JANET' in the error files to determine the cause of the failure.
-----------------------------------------------------------------
02/01/01  14:28:57 AMQ8004: MQSeries queue manager ended.
EXPLANATION: MQSeries queue manager janet ended.
ACTION: None.
-----------------------------------------------------------------
02/02/01  15:02:49 AMQ9002: Channel program started.
EXPLANATION: Channel program 'JANET' started.
ACTION: None.
-----------------------------------------------------------------
02/02/01  15:02:51 AMQ9001: Channel program ended normally.
EXPLANATION: Channel program 'JANET' ended normally.
ACTION: None.
-----------------------------------------------------------------
02/02/01  15:09:27 AMQ7030: Request to quiesce the queue manager
accepted. The queue manager will stop when there is no further
work for it to perform.
EXPLANATION: You have requested that the queue manager end when
there is no more work for it.  In the meantime, it will refuse
new applications that attempt to start, although it allows those
already running to complete their work.
ACTION: None.
-----------------------------------------------------------------
02/02/01  15:09:32 AMQ8004: MQSeries queue manager ended.
EXPLANATION: MQSeries queue manager janet ended.
ACTION: None.
 ...
```

## EMS events

An EMS event is generated for each error entry made in the MQERRLG1 file. For more information about EMS events, see "Event Management Service (EMS) events" on page 159.

## Dead-letter queues

Messages that cannot be delivered for some reason are placed on the dead-letter queue. You can check whether the queue contains any messages by issuing an MQSC DISPLAY QUEUE command. If the queue contains messages, you can use the provided browse sample application (MQSBCG0E) to browse messages on the queue using the MQGET call. The sample application steps through all the messages on a named queue for a named queue manager, displaying both the message descriptor and the message context fields for all the messages on the named queue.

You must decide how to dispose of any messages found on the dead-letter queue, depending on the reasons for the messages being put on the queue.

Problems may occur if you do not have a dead-letter queue on each queue manager you are using. When you created this dead-letter queue, you must alter the DEADQ attribute of the queue manager using **runmqsc**.

## Configuration files and problem determination

Configuration file errors typically prevent queue managers from being found and result in "queue manager unavailable" type errors.

There are several checks you can make on the configuration files:

- Ensure that the configuration files exist.
- Ensure that they have appropriate permissions.
- Ensure that the MQSeries configuration file references the correct queue manager and directories.

## Using MQSeries trace

MQSeries for Compaq NSK uses the following commands for the trace facility:

- **strmqtrc** – see "strmqtrc (Start MQSeries trace)" on page 286
- **dspmqtrc** – see "dspmqtrc (Display MQSeries formatted trace output)" on page 257
- **endmqtrc** – see "endmqtrc (End MQSeries trace)" on page 266

The trace facility uses one file for each entity being traced, with the trace information being recorded in the appropriate file.

Trace options are specified in the QMINI file.

**Note:** With MQSeries for Compaq NSK, tracing can also be controlled via the Queue Manager menu of the Message Queue Management (MQM) facility.

### Trace file names

Trace file names are constructed in the error log subvolume as follows:

```
TRccpppp
```

where *ccpppp* is the process identifier (PID) of the process producing the trace. The PID is made up of:

cc, the CPU number.
pppp, the Process number.

If the tracing utility encounters a trace file of an identical process identifier that has not been deleted, it replaces the final character of the process number with a letter, giving 26 processes of the same PID the opportunity to write output. For example, the first trace file for PID 00, 0315 would be `TR000315`. For a second process started on completion of process 00, 0315 with the same PID, the trace file would be `TR00031A`.

**Note:** Because of this restriction, trace files should be purged from the system as soon as they have been examined.

## Sample trace data

The following sample is an extract from a trace:

```
 ...
ID     ELAPSED_MSEC     DELTA_MSEC      APPL    SYSCALL KERNEL  INTERRUPT

30d     0 0 MQS CEI Exit!. 12484.1 xcsWaitEventSem rc=10806020
30d     0 0 MQS CEI Exit! 12484.1 zcpReceiveOnLink rc=20805311
30d     0 0 MQS FNC Entry 12484.1 zxcProcessChildren
30d     0 0 MQS CEI Entry. 12484.1 xcsRequestMutexSem
30d     1 0 MQS CEI Entry.. 12484.1 xcsHSHMEMBtoPTR
30d     1 0 MQS CEI Exit... 12484.1 xcsHSHMEMBtoPTR rc=00000000
30d     1 0 MQS FNC Entry.. 12484.1 xllSemGetVal
30d     1 0 MQS FNC Exit... 12484.1 xllSemGetVal rc=00000000
30d     1 0 MQS FNC Entry.. 12484.1 xllSemReq
30d     1 0 MQS FNC Exit... 12484.1 xllSemReq rc=00000000
30d     1 0 MQS CEI Exit.. 12484.1 xcsRequestMutexSem rc=00000000
30d     2 0 MQS CEI Entry. 12484.1 xcsReleaseMutexSem
30d     2 0 MQS CEI Entry.. 12484.1 xcsHSHMEMBtoPTR
30d     2 0 MQS CEI Exit... 12484.1 xcsHSHMEMBtoPTR rc=00000000
30d     2 0 MQS FNC Entry.. 12484.1 xllSemRel
30d     2 0 MQS FNC Exit... 12484.1 xllSemRel rc=00000000
30d     2 0 MQS CEI Exit.. 12484.1 xcsReleaseMutexSem rc=00000000
30d     2 0 MQS CEI Entry. 12484.1 xcsHSHMEMBtoPTR
 ...
```

*Figure 44. Sample trace*

**Notes:**

1. In this example the data is truncated. In a real trace, the complete function names and return codes are present.
2. The return codes are given as values, not literals.

# First Failure Support Technology™ (FFST)

FFST errors are normally severe, and indicate either a configuration problem with the system or an MQSeries internal error. In most cases, the queue manager remains operational, though there may be a brief interruption of service to some or all applications. FFSTs are referenced in the file ZMQSSYS.MQSYSLOG.

## How to examine the FFSTs

The files are named `FD`*`nnnnn`*, where:

*`nnnnn`* Is the process ID reporting the error

When a process creates an FFST report, it also generates an EMS event.

A typical FFST report is shown in Figure 45.

```
MQSeries First Failure Symptom Report
=====================================

Date/Time         :- February 6  12:23:26    2001
Host Name         :- \HURSLEY
PIDS              :- 5724A39
LVLS              :- 510
Product Long Name :- MQSeries for Compaq NonStop Kernel
Vendor            :- IBM
Probe Id          :- RM020011
Application Name  :- MQM
Component         :- rrxOpenSync
Build Date        :- Feb 5 2001
Exe File Name     :- \HURSLEY.$DATA0.ZMQSEXE.MQMCACAL
UserID            :- MQM.MANAGER
Process File Name :- \HURSLEY.$Z734:15441941
Node number       :- 1
CPU               :- 0
PIN               :- 339
QueueManager      :- MT01
Major Errorcode   :- xecF_E_UNEXPECTED_RC
Minor Errorcode   :- Unknown(A)
Probe Type        :- MSGAMQ6118
Probe severity    :- Severity 2: error
Probe Description :- AMQ6118: An internal MQSeries error has occurred.
Text              :- Error creating synch file

Arith1            :- 10  (0xa)
Comment1          :- error 0000000010 in function 0000000020
```

*Figure 45. Sample First Failure Symptom Report*

However, there is one set of problems that they may be able to solve. If the FFST shows “out of resource” or “out of space on device” descriptions, it is likely that the relevant system limit is exceeded.

To resolve the problem, increase the appropriate limit and restart the queue manager.

# Chapter 15. Scalability and performance

This chapter discusses techniques for maximizing the performance and scalability of MQSeries application programs. It contains the following sections:

## Introduction

Tuning for performance and scalability is done to minimize the use of two key resources: CPU and the Disk subsystem. Applications that use less CPU and less disk IO will perform better and scale better (for example, they can be configured to process ever growing workloads by using the using hardware and system software to its fullest).

This chapter addresses techniques to improve the performance of both application programs themselves, and MQSeries. The following sections summarize the broad principles for improving application performance.

### Designing new applications for performance and scalability

Early in the design phase for new applications, you should consider how MQSeries and other subsystems are used. The business need should determine which MQSeries features are needed or are relevant for each application. Some MQSeries features, such as message persistence, carry strong integrity and delivery assurances, which require larger amounts of CPU and disk IO to provide. If these assurances are not required for a particular application, then configuring MQSeries accordingly can yield significant performance gains. This judgment is best made early in the design phase when the driving business need is being examined.

### Designing to minimize or eliminate the use of shared resources

Absorbing growth in message traffic demands that the underlying hardware and system software be used to its fullest. Usually, any resource that is shared becomes a bottleneck, as the load increases. This bottleneck develops either because the degree of sharing has increased (for example, more users sharing the same CPU) or there is simply more of the resource is being consumed (for example, each user is performing more work).

From a performance and scalability perspective, the CPU and the Disk subsystem represent the most often-shared resources, and therefore require most attention.

MQSeries processes can be spread across as many CPUs as you want. On a system with several CPUs, distributing the MQSeries processes across the available CPUs provides better performance than using the default CPU assignment. Similarly, to maximize utilization of the disk subsystem, it is wise to position separate queue files on separate disks volumes, serviced by separate Compaq NSK disk processes, if possible.

© Copyright IBM Corp. 1993, 2001

## Performance tuning is inherently iterative

Achieving objectively better performance requires a measure-tune-remeasure cycle. Each tuning cycle should involve the change of only one major variable so that the effect of that variable can be compared against the effect of other variables. It is usually counter-productive to alter more than one setting at the same time, since some changes may improve overall performance more than others, while still others may reduce it. The performance of an application system is usually determined by the single, limiting, bottleneck. Making a tuning change to a system usually causes some other resource to become the bottleneck. The interplay between changes of this nature emphasizes the need to carefully follow the measure-tune-remeasure method when tuning for performance.

For example, a given application may write large numbers of small persistent messages to a queue. It may be useful to test the effects of making all those messages non-persistent, or writing the same amount of data inside a smaller number of large messages. Both changes improve performance (usually) but without a separate measure-tune-remeasure cycle for each change, it may not be clear which brings the greatest improvement.

# Persistent messages

Persistent messages carry the strongest assurances offered by MQSeries regarding delivery and recoverability. Persistent messages are always stored on hardened media, and therefore survive a queue manager restart. Nonstop TM/MP audits queue files so that reading or writing persistent messages results in disk activity to both the queue file itself, and the TM/MP audit files. TM/MP audit logging is required to maintain transactional integrity for persistent messages, even in the case of a system or hardware failure. The TM/MP audit logging associated with persistent messages must be considered when assessing the performance of an MQSeries application design.

Persistence is a property of a message, not the queue in which it is stored. Queues can store both persistent and non-persistent messages, although the administrator can specify whether new messages are persistent when a putting application does not otherwise specify. (See the Queue DEFPSIST attribute described in *MQSeries Application Programming Reference*).

# Non-persistent messages

Unlike persistent messages, non-persistent messages are not hardened to disk and do not survive a queue manager restart. Depending on the queue server options for the queue, non-persistent messages may get checkpointed to the backup queue server. The primary reason for using non-persistent messages is performance. Persistent messages carry strong assurances for delivery and recoverability, so reading or writing them require disk activity to both the queue files themselves, and the TM/MP audit files. This disk activity reduces the performance of applications that read or write persistent messages, and MQSeries channels that move persistent messages to other queue managers.

Using non-persistent messages whenever possible can dramatically improve the performance because they are not hardened to disk, but are instead cached in memory managed by the Nonstop queue servers.

Persistence is a property of a message, not the queue in which it is stored. Queues can store both persistent and non-persistent messages, although the administrator can specify whether new messages are persistent when a putting application does not otherwise specify (see the Queue DEFPSIST attribute).

## Non-persistent messages and channels

Message channels use synchronization logging at both the sending and receiving end to assure once and once-only delivery of messages sent over the network. This synchronization logging is additional to any audit logging performed by TM/MP (on behalf of the queue manager) when channels read and write messages to queues. Message channels can be configured to not perform synchronization logging when sending and receiving non-persistent messages, by setting the NPMSPEED attribute to FAST. The NPMSPEED channel attribute controls the behavior of both sending and receiving channels that are processing non-persistent messages. When NPMSPEED is set to NORMAL for a channel, non-persistent messages are part of the channel's message batch (as defined by the BATCHINT and BATCHSZ attributes) and require synchronization logging in the same way as persistent messages. Further, when NPMSPEED is set to NORMAL, the channel reads and writes non-persistent messages under syncpoint control, which causes a small amount of TM/MP audit file activity at the beginning and end of a transaction.

When NPMSPEED is set to FAST, non-persistent messages are not part of the channel's current batch and are read and written to queues outside of syncpoint control. Using NPMSPEED(FAST) therefore removes two sources of channel disk activity; the logging done by the channel batch synchronization mechanism, and the TM/MP audit logging that would otherwise be done for reading and writing messages under syncpoint.

NPMSPEED(FAST) is a performance option that trades recoverability of non-persistent messages after a failure for considerably higher performance; using NPMSPEED(FAST) can cause non-persistent messages to be lost if the channel or network fails.

The default value for NPMSPEED is FAST.

# Queue servers and queue files

Queue servers are MQSeries processes that mediate the reading and writing of messages and the storage of those messages. As such, queue servers represent a key MQSeries component worthy of close attention. Queue server configuration can have a major impact on performance of a busy MQSeries system.

Queue servers have responsibility for the physical storage of messages held in queue. It is useful to examine the storage of persistent and non-persistent messages separately, because queue servers manage them in fundamentally different ways.

A queue server can manage one or more queues. When created, a queue is managed by the default queue server. Therefore, by default, all queues are managed by the default queue server unless they are assigned to other queue servers.

For each queue managed by a queue server, the following files exist:
- A queue file
- A queue overflow file
- A touch file

Additionally, there may exist a message overflow file for each large message in the queue (as defined by the queue's message overflow threshold). Message overflow files are discussed in a later section.

## Persistent message storage

Persistent messages are always hardened to disk. The way that persistent messages are stored depends mostly on the size of the message:

| Message Size | How stored |
|---|---|
| < 3000 bytes (approx) | Stored entirely in the queue file. |
| > 3000 bytes (approx) <= Message Overflow threshold | 1 st 3000 bytes (approx) stored in the queue file the rest in the Queue Overflow file |
| > Message Overflow Threshold | 1 st 3000 bytes (approx) stored in the queue file, rest in the Message Overflow File |

## Non-persistent message storage

Non-persistent messages are stored in memory buffers in the queue server's process. For this reason, the queue server performs no disk IO when reading and writing non-persistent messages.

## Queue server CPU distribution

Queue servers are PATHWAY server classes, and are therefore defined in the queue manager's PATHWAY. When the queue manager is created, a default queue server is defined in the PATHWAY with the queue server's primary process running in CPU 0. Since CPU 0 normally contains many high-priority system processes, it is usually better to not run a queue server in CPU 0. Rather, you should identify a number of relatively quiet CPUs, and then create a queue server in each of them. A new queue server does not manage any queues until a queue is explicitly assigned to it.

## Re-assigning queues to queue servers using altmqfls

When you create a queue (using **runmqsc**), MQSeries assigns it to the default queue server. Unless you create new queue servers and assign queues to them, all queues are managed by the single, default, queue server.

In a busy MQSeries system it is neither efficient nor scalable to have all queues assigned to a single queue server. The major reason for creating new queue servers and assigning queues to them is to distribute the queue server CPU load more evenly across the available CPUs.

You can use **altmqfls** to assign a queue to a different queue server. For example:

```
altmqfls --qmgr QMGR --type QLOCAL --server $QS01 TEST.QUEUE
```

For more information about using **altmqfls**, see "altmqfls (Alter queue file attributes)" on page 230.

## Cluster transmit queue: SYSTEM.CLUSTER.TRANSMIT.QUEUE

MQSeries uses a single cluster transmit queue for all clustering operations. If your queue manager is part of a busy cluster, this queue should be assigned to a dedicated queue server, both to maximize performance of clustering operations and to minimize the impact on other applications.

## Changing queue file placement using altmqfls

By default, the three primary queue files (queue, queue overflow and touch file) are stored in the queue manager’s M subvolume. Message overflow files are stored in the queue server’s subvolume.

There are two reasons why you might consider re-locating queue files to other disk volumes:

- To spread the disk IO load more evenly
- To overcome OS limits on the number of open files per disk volume

You can use **altmqfls** to move existing queue files to another disk volume:

```
altmqfls --qmgr QMGR --type QLOCAL --volume $DATA01 TEST.QUEUE
```

To use **altmqfls** to move a queue, the queue must not be open.

You cannot move existing message overflow files. You can however cause new message overflow files to be created elsewhere using the --msgofsubvol option of **altmqfls**:

You can cause new message overflow files to created in a new location:

```
altmqfls --qmgr QMGR --type QLOCAL --msgofsubvol $DATA01.TESTMOF TEST.QUEUE
```

For more information about using **altmqfls**, see “altmqfls (Alter queue file attributes)” on page 230.

## Partitioning queue files

File partitioning is a technique that splits a file across more than one disk volume. The file then logically consists of more than one partition, one on each disk volume.

The advantages of partitioning a queue file are:

- Partitioning spreads the one disk IO load for a single queue across more than disk volume and therefore more than one IO path
- Partitioning allows a logical queue file to be larger than the maximum size of a physical file on a disk volume.

You can partition an existing queue file using standard TACL commands (the FUP utility). MQSeries supplies a TACL script called `PARTIT` that shows how to use FUP to partition an existing queue file.

Once a queue file has been partitioned, the managing queue server attempts to spread new messages evenly across all available partitions. This further enhances the benefits of partitioning.

**Note: altmqfls** cannot move a partitioned queue file.

## Message overflow files

Message overflow files are created for each message that is larger than the defined message overflow threshold. Message overflow files are unstructured files that are not audited by TM/MP. For very large messages, it is more efficient to store most of the message in the message overflow file (unaudited by TM/MP). For smaller messages, it is more efficient to the store the entire message in the TM/MP audited queue files (queue and queue overflow files). The crossover point has been empirically determined to be about 200 KB.

The message overflow threshold is set to its default of 200 KB when a queue is created. The threshold can be changed using the --oflowsize switch of **altmqfls**.

You can use the **altmqfls** command to set the message overflow threshold to new value:

```
altmqfls --qmgr QMGR --type QLOCAL --oflowsize 400000 TEST.QUEUE
```

## Buffering messages during browsing

The queue server can maintain the first *n* bytes of all persistent messages for a queue, in memory. This feature can have a dramatic effect of the performance of applications that are browsing persistent messages in a queue. If the messages that the application is browsing are smaller than this Browse Threshold, then no disk IO is needed to browse each persistent message.

If you wish, you may also include information in, for example, the first 100 bytes of the persistent messages that identify the content of a message. Browsing the first 100 bytes of data of each message can be done with no disk IO in order to locate messages of interest. Once found, the entire message data can be read and dequeued if required.

You can use the --browse option of **altmqfls** to set a browse threshold for a queue. By default, no persistent message data is kept in memory (for example, the Browse Threshold is zero). The maximum value is 25000.

```
altmqfls --qmgr QMGR --type QLOCAL --browse 100 TEST.QUEUE
```

## Other queue server options

Other queue server options, such as whether the queue is loaded from disk into cache when the queue manager starts or, whether the non-persistent messages are checkpointed to the backup queue server, use the --qsoptions parameter with the L, S and C options. These options can be used either alone or combined, to fine tune the queue’s retrievability and reliability.

**Note:** *All* of the --qsoptions SLC are set each time the command is issued. For example, --qsoptions S will unset L and C. The --qsoptions option can be specified once and once only on a command line.

### Load on Startup

Controlled by the **altmqfls** --qsoptions S switch. This option causes the queue server to read the queue files, and build its internal message data structures at queue server startup (typically, at queue manager start) rather than when the queue is first opened.

```
altmqfls --qmgr QMGR --type QLOCAL --qsoptions S TEST.QUEUE
```

Using the option results in less CPU and disk IO activity when a queue is first opened, but causes more queue server activity (CPU and disk IO) during queue manager start.

## Lock In Cache

Controlled by the **altmqfls** --qsoptions L switch. This option causes the queue server to lock in memory, the data structures and chains associated with a queue. The queue's memory data structures are not unloaded to disk, to make room for other queues. The default behavior is to unload to disk, a queue's data structures when required.

```
altmqfls --qmgr QMGR --type QLOCAL --qsoptions L TEST.QUEUE
```

Using these options results in faster access to a queue's memory data structures at the possible expense of other queues.

## Checkpoint NPM

Controlled by the **altmqfls** --qsoptions C switch. This option causes the queue server to checkpoint non-persistent messages with its non-stop backup process. The default behavior is to checkpoint non-persistent message data.

Using this option results in a higher degree of reliability for non-persistent messages, at the cost of greater IPC traffic and greater CPU and memory utilization for both the primary and backup queue server processes.

```
altmqfls --qmgr QMGR --type QLOCAL --qsoptions C TEST.QUEUE
```

## Measure Counter

Controlled by the **altmqfls** --meascount switch. This option causes the queue server to maintain a user-defined Measure Counter with the queue's CURDEPTH. The measure counter is useful when you gather data to assess the overall performance behavior of a system. By using this option, you can correlate message reads and write activity (MQGETs and MQPUTs) to a queue with other system-related variables such as CPU and disk IO.

```
altmqfls --qmgr QMGR --type QLOCAL --meascount TESTCOUNT1 TEST.QUEUE
```

Using this option allows you to accurately assess the performance and scalability of a system using MQSeries, and hence improves the results of your tuning effort.

# CPU assignment

When you start a queue manager, MQSeries for Compaq NSK creates a number of processes. Some of these processes provide the core messaging operations while others perform functions that indirectly support these operations. Processes that provide the core messaging features of MQSeries are busy when applications are busy making MQI messaging calls (MQGETs and MQPUTs).

The following MQSeries and Compaq NSK processes are involved in core messaging operations:
- LQMAs (Local Queue Manager Agents)
- Queue servers
- Status servers
- NSK Disk processes

The following MQSeries and NSK processes are involved in distributed queuing operations:
- MCAs
- Default Status Server
- NSK TCPIP or SNA processes

The following MQSeries processes are involved in support or administrative operations:
- Queue Manager Server
- Repository Servers
- Execution Controllers (ECs and ECBoss)
- Channel Initiators
- Trigger Monitors
- Listeners

A heavily loaded MQSeries system usually shows high CPU utilizations for the processes in the first two categories shown above (core messaging processes and distributed queuing processes).

Distributing the CPU load of MQSeries therefore usually involves spreading the queue manager's LQMAs, queue servers and MCAs across as many CPUs as possible.

The default configuration for a new queue manager is to run processes in CPU 0 (for Nonstop process pairs, the primary processes run in CPU 0 and the backup in CPU 1). This is not adequate for a production environment, and you should be prepared to re-configure the queue manager's PATHWAY to sensibly spread busy processes across as many CPUs as possible. CPU 0 usually contains many high-priority operating system processes and is therefore a poor choice for running busy MQSeries processes.

The CPU distribution of ECs, which are defined in the queue manager's PATHWAY, control the execution of both LQMAs and MCAs. Queue servers are also defined in the queue manager's PATHWAY.

Looking beyond MQSeries processes, the NSK operating system disk processes are an important component of messaging operations (particularly when queue servers are handling persistent messages). Heavy use of distributed queuing over a network necessarily causes the corresponding TCPIP or SNA processes to consume CPU. You should consider the number and CPU placement of these processes when assessing the overall performance profile on an MQSeries installation.

### CPU assignment

For information on assigning processes to CPUs, the configuration of disk processes or TCPIP refer to the relevant Compaq NSK system documentation. For information on the configuration of SNA processes, refer to relevant SNAX or ICE documentation For information on configuring TCPIP or SNA processes, see "Appendix M. Setting up communications" on page 351.

# FASTPATH binding application programs

FASTPATH binding is a feature of the MQI that is designed to make MQSeries applications run more efficiently. FASTPATH binding can be used to reduce the overhead inherent in all MQI verbs issued by MQSeries application programs. Applications that use FASTPATH binding are referred to as trusted applications because of the proximity of the queue manager software and memory to the customer's application software. Errors in trusted applications can damage MQSeries data structures and can compromise queue manager integrity.

## Background

When an application program executes an MQCONN verb, MQSeries creates (or reuses) a special process called a Local Queue Manager Agent (LQMA). The LQMA services all subsequent MQI calls made by the application using that connection handle. On MQSeries for Compaq NSK, the LQMA may be running in the same or a different CPU as the connecting application.

Since the LQMA is a separate process, an application program does not have direct access to the memory or files used by MQSeries. An errant application program cannot therefore damage the LQMA. In this way, MQSeries software and data structures that are critical to its operation are isolated from the customer's application software and data. This isolation comes at a price. The MQCONN verb cannot complete until the new LQMA process is created (or an existing one is re-used), but the greater cost results from the fact that information must be passed to the LQMA each time the application issues an MQI verb.

This MQI information is passed to the LQMA using an Interprocess Communications (IPC) mechanism. IPC requests may be intra-CPU (if the LQMA happens to be running in the same CPU as the connecting application) or the more expensive inter-CPU.

## Reducing MQI overhead

Application designers can use FASTPATH binding as a way of removing the application-LQMA IPC overhead associated with each MQI verb (the possible LQMA process creation is also avoided). When FASTPATH binding is enabled for an application, no separate LQMA process is used. Instead, the components of MQSeries normally contained in the LQMA, are loaded into the user's process (for example, the connecting application's process).

Subsequent MQI verbs issued by the application require no IPC activity with the LQMA, since the MQSeries software and data structures (normally stored in the LQMA process) are held locally within the application's process. Note that other IPC activity may still occur when the queue manager needs to communicate with other MQSeries processes such as queue servers. FASTPATH binding does not remove all IPC activity, but it does remove an important source of IPC activity.

## Enabling FASTPATH binding

To use FASTPATH binding, connect to the queue manager using the MQCONNX verb with the MQCNO_FASTPATH_BINDING option. The value (if any) of the MQCONNECTTYPE PARAM influences the behavior of MQCONNX. If the MQCONNECTTYPE PARAM is present, its value must be FASTPATH to allow MQCONNX to setup a FASTPATH connection. Once a FASTPATH binding connection is established, all other MQI verbs behave as they would for a standard binding connection, with the exceptions noted in the section below.

## Restrictions when using FASTPATH binding

When using STANDARD binding connections, application software is isolated from internal MQSeries data. This isolation is removed when a trusted application establishes a FASTPATH binding connection. Errors in trusted applications can therefore damage MQSeries data structures and can compromise queue manager integrity. This must be taken in consideration when assessing whether to use FASTPATH bindings for a given application.

The following additional considerations apply to trusted applications:

- Trusted applications **must** explicitly Disconnect from the queue manager (for example, issue MQDISC).
- Trusted applications must be stopped before the **endmqm** command is issued. Trusted applications must run as Administrator user id. (the User id corresponding to mqm principal).
- Trusted applications can run only in CPUs that contain a running MQSeries repository server.

# Chapter 16. Data integrity and availability

This chapter describes concepts of data integrity and availability and how these important aspects of a system apply to the management and configuration of MQSeries for Compaq NSK V5.1. This chapter describes the levels of data integrity and availability you can expect from MQSeries and the configuration choices that can influence these levels. It contains the following sections:

- "Data integrity"
- "Availability" on page 212
- "Persistent and non-persistent data" on page 213
- "Database consistency" on page 215
- "Critical processes" on page 218
- "Clusters" on page 223
- "Configuration considerations for availability" on page 224
- "Configuration considerations for data integrity" on page 224

You need to have read and understood "Chapter 2. MQSeries for Compaq NSK V5.1 architecture" on page 19 to properly understand and use the information in this chapter.

## Data integrity

The concept of data integrity can be understood best by considering the following desirable aspects of the storage and management of data, particularly for on-line transaction processing applications:

- When a record of data is written or read from a record in a database, the data must not be corrupted, duplicated or lost without an error indication during the transfer.
- When data is required to be accessed concurrently by multiple processes, these processes must be presented with the same view of the data and the data must be protected from corruption, duplication or loss.
- When a set of consistent changes are required to data in multiple databases, the changes must either be all made or none made.

For MQSeries, the data integrity requirements for data storage listed above are just as applicable to messaging operations (for example, MQPUT and MQGET) on queues. Note that consistency of multiple database changes must be preserved across *and between* application databases and MQSeries queues.

MQSeries for Compaq NSK V5.1 is designed to maintain data integrity for persistent data operations through any single point of failure (hardware or software). In fact, data integrity can be maintained in several cases through multiple points of failure. This does not imply that non-persistent messages are unreliable; queue server architecture provides features for making non-persistent messages as reliable as persistent ones, except in the case of a catastrophic system failure. (For more on queue server architecture, see "Chapter 2. MQSeries for Compaq NSK V5.1 architecture" on page 19.)

With MQSeries for Compaq NSK V5.1, data integrity is provided by a combination of fundamental features of the Compaq NSK system software and hardware, and the MQSeries software itself.

© Copyright IBM Corp. 1993, 2001

### Data integrity

There are several ways in which the level of data integrity can be influenced by choices in the configuration of MQSeries:

- Choice of message persistence by the application
- Choice of storage technique for persistent messages
- Choice of non-persistent message tuning options
- Choice of queue server configuration options
- Configuration of hardware supporting queue files
- Use and configuration of NonStop TM/MP
- Use and configuration of Remote Database Facility (NonStop RDF).

Each is described later in this chapter.

## Availability

Availability in general terms is a measure of the time that an application, or service is operational and usable compared to elapsed time. Thus *continuous availability* expresses the ultimate aim of all such systems. Of course, such measurements mean nothing without a corresponding time period associated with the measurement—since it is easy to claim 100% Availability over a short period of time.

In a real-world situation over a reasonable operational time span, a system will suffer a number of different types of challenge to its availability:

- Hardware and system software failures
- Failures within the application software itself
- The need to make changes to any aspect of the system for preventative maintenance
- Traffic or transaction load that exceeds design constraints or resource limitations

As for data integrity, with MQSeries for Compaq NSK V5.1, availability is provided by a combination of fundamental features of the Compaq NSK system software and hardware, and MQSeries itself.

It is important to recognize that (at least in its current form) MQSeries for Compaq NSK does not aim to provide a level of continuous availability equivalent to that provided by Compaq NSK system software such as the file system. There are in fact some components of the queue manager that do provide this level of availability, but the queue manager as a whole does not.

MQSeries for Compaq NSK, V5.1 is intended to provide a level of availability such that on any single point of failure (hardware or software):

- The queue manager connections that suffer interruption or discontinuation of service are limited to those with components that suffer the failure directly (for example, on a CPU failure, connections that fail should only be those that are provided by LQMA processes that were running in that CPU)
- The queue manager remains available for new connection attempts without manual intervention being required from system administrators
- Access to a queue manager object (for example a local queue) must not be prevented from any connection other than those directly affected by the failure.

In addition to these Compaq NSK specific features of MQSeries, there are several features that are common to all MQSeries Version 5.1 platforms that you can make use of to enhance the availability of MQSeries for Compaq NSK. There are several ways in which in which the level of availability can be influenced by choices in the configuration of MQSeries:

- Choice of application design
- Choice of PATHWAY configuration options
- Use and configuration of standard MQSeries functions (including clusters)
- Choice of hardware supporting MQSeries
- Choice of non-persistent message configuration options
- Use and configuration of NonStop TM/MP
- Use and configuration of NonStop RDF.

## Persistent and non-persistent data

When used in relation to MQSeries, the term *persistence* implies several qualities to data:

- A change to persistent data survives queue manager restart
- Persistent data is stored in non-volatile media
- Persistent data satisfies the highest requirement for data integrity provided by the particular operating environment
- Persistent data operations trade this higher level of integrity for speed and resource utilization.

Most administrative operations are made to persistent data, since the configuration databases of MQSeries must have the highest level of data integrity to minimize the risk that the availability of MQSeries is seriously degraded. The speed or resource utilization of most administrative operations is not of prime concern, since they are performed infrequently.

Examples of persistent administrative operations:

- Change to a queue's attributes
- Change to a channel's attributes
- Creating a new queue, process, namelist or channel.

Examples of non-persistent administrative operations:

- Starting or stopping a channel
- Inquiring about the attributes of a queue manager object.

MQSeries for Compaq NSK provides several choices for the way messages are stored, based on the choice between persistent and non-persistent made by the application when enqueuing a message, and on queue level configuration choices made by the system administrator.

### Persistent messages

Persistent messages are always stored on disk. As system administrator, you can choose between two storage techniques for persistent messages on a queue by queue basis. The choice of which type of storage technique to use is based on message size since the primary purpose for implementing different storage techniques is to improve the performance of messaging operations on very large messages.

## Persistent and non-persistent data

All persistent messages have a single record in the audited queue file. This record contains the headers and important control information about the message, plus as much message data as can be accommodated within the maximum record size of 4096 bytes.

The fastest and most efficient mechanism for storage of small and medium size persistent messages under TM/MP control is to store overflow message data in the queue overflow file in multiple records, using the same basic technique as used in MQSeries for Compaq NSK Version 2.2.0.1. All data for these messages is logged in TM/MP and is therefore fully recoverable from audit trails if necessary.

For large messages (over about 200 KB of data), the most efficient mechanism turns out to be the use of a dedicated message overflow file which is unaudited. The data that is written to a message overflow file does not therefore get written to the TM/MP audit trail, saving CPU and disk IO and can also be transferred in large blocks.

The aspects of data integrity that differ slightly between these two mechanisms are expressed in Table 6:

*Table 6. Queue Overflow compared with Message Overflow*

| Data Integrity aspect | Queue Overflow method | Message Overflow method |
|---|---|---|
| Amount of data that is audited by TM/MP | All message data is audited | Only the data that will fit in the Q-file record is audited |
| Recoverability of data from audit trail in case of multiple failures resulting in total volume loss. | Entire message is recoverable from audit trail | Only the message header and first part of data is recoverable from audit trail. |
| Maximum size of message possible | Limited by number of record locks per volume per transaction imposed by ENSCRIBE. For a non-partitioned file, this is approximately 20 MB for a default ENSCRIBE configuration. The practical limit may be smaller than this due to physical memory limitations. | Limited only by available disk space or 100 MB which is the maximum permissible message size for MQSeries. |
| Compatibility with Remote Database Facility (RDF) | Fully compatible (specify the MQRDF environment parameter). | Incompatible with RDF. Since message data is not audited, RDF cannot be used to propagate message operations using message files to the backup system. |
| Fault-tolerance to disk hardware problems | Fully fault-tolerant to any single point of failure if mirrored disks are employed and since all data is audited, file recovery can be performed in the event of failure of both disks in a mirrored pair, or the only disk if not mirrored. | Fully fault-tolerant to any single point of failure if mirrored disks are employed to hold message overflow files. Message data cannot be recovered in the event of total volume failure. |

In summary, for persistent messages, both storage techniques attain a very high level of data integrity, but there are some limitations for message overflow files because not all message data is audited. Normally these limitations are only of concern for very large messages, and in such cases can be addressed by the use of segmentation (to split an application message into smaller physical messages).

## Non-persistent messages

Non-persistent messages (NPM) are normally stored in memory. If a queue server is managing a large amount of non-persistent data and reaches a threshold whereby it is close to running out of virtual memory, then the queue server will force NPM to disk. Occasionally the queue server will also copy NPM to disk files for certain administrative operations while changes are made to configuration online. When applications access NPM they are always resident in memory.

The queue server manages the storage of all messages for the queues that it is responsible for, and for each queue provides a configuration option to control the level of data integrity applied to NPM. The configuration parameter controls whether the queue server checkpoints NPM to the backup process, so that the NPM are as tolerant as persistent messages to the failure of the CPU containing the primary queue server process.

The price paid for the use of NPM checkpointing is that the queue server consumes more CPU (primary and backup), transfers more data to the backup during checkpointing and consumes more memory in the backup process since it has to store the message again.

If NPM checkpointing is not enabled for a queue, and the primary queue server process terminates abnormally (due to process or CPU failure) then any NPM that were present on the queue before the takeover are discarded since they were only stored in the primary process.

The option of fault-tolerant non-persistent messages is not available on any other MQSeries platform, and for consistency across the product line, NPM checkpointing is enabled by default (use **altmqfls** to disable this feature). You should consider carefully whether non-persistent messages meet your needs for data integrity. If they do, then you should be able to take advantage of their significant resource utilization savings and performance gains.

# Database consistency

Database consistency must be preserved both internally by MQSeries, and externally when the Syncpoint option is used for messaging operations. Both are vital for data integrity and availability.

## Internal database consistency

All critical database files within the queue manager are audited by TM/MP. The queue manager processes must therefore use transactions to make changes to them. All changes are logged in the TM/MP audit trail and also (if used) duplicated using RDF to one or more disaster recovery systems. Thus the highest level of data integrity for internal databases can be assured.

The use of TM/MP to protect internal databases helps ensure that on system failures as well as software failures the integrity of the critical databases is not compromised. This means that restarting (automatically or manually) the processes or services that use these databases is much more likely to be successful, leading to higher availability.

## External database consistency

The coordination of changes to and consistency of external databases with MQSeries databases is enabled by the use of TM/MP within MQSeries and by applications. MQSeries messaging operations may be made under syncpoint control, which requires the application to have an active current transaction (either inherited from another process or started using TM/MP BEGINTRANSACTION service). This transaction is inherited by the queue manager and any storage or critical database update that is required is performed using this transaction. TM/MP ensures that the appropriate audit trail entries are recorded for all disk IO performed under the transaction in whatever process performs the update.

TM/MP also maintains a consistent view of the updates that have been made but not yet committed by holding record locks on the affected records in all database files. For example, a row in a SQL table that has been inserted under TM/MP control cannot be updated or deleted until the transaction commits.

When the queue manager replies to the application process (and returns from the MQI), the application may continue to do more work under this transaction—by using MQSeries to enqueue or dequeue more messages or performing database updates of its own using ENSCRIBE or NonStop SQL. When the application is ready to make the changes to MQSeries queues and, other databases permanent, the ENDTRANSACTION service of TM/MP is called, which commits the changes to all databases, system wide at the same time.

If the application determines that an error has occurred during the processing of the transaction, and some updates to databases have been performed, then the application should call ABORTTRANSACTION to cause TM/MP to back-out the changes to all databases, system wide at the same time. This could cause, for example, a message to be replaced on a queue after is has been de-queued in a Syncpoint MQGET operation as well as the removal of a prior insert into an SQL database table. With careful application design, these errors can be handled to maintain consistency and enhance data integrity and availability for applications.

When MQSeries performs a syncpoint MQPUT or MQGET operation, it adjusts the queue depth at the time of the operation on the assumption that the transaction will eventually be committed. Thus the queue depth includes the number of uncommitted messages that are on the queue as well as the committed ones. If the transaction eventually aborts (either deliberately or due to failure) MQSeries adjusts the queue depth to maintain a fully accurate value. This is an improvement from prior V2.2 releases of MQSeries on Compaq NSK, where it was impossible for the queue manager to determine the outcome of transactions and so maintain an accurate depth under all conditions.

“OpenTMF” describes the mechanism by which this is possible.

## OpenTMF

OpenTMF is the informal name for a new internal interface to the NonStop TM/MP product which MQSeries, with Compaq’s assistance, has been able to use to determine the outcome of a transaction that the queue manager uses to perform syncpoint messaging operations.

This new feature of TM/MP is the foundation for the introduction of the heterogeneous transaction processing capability of NonStop TM/MP, introduced with D42. OpenTMF allows MQSeries to register as a participant in any transaction it has inherited from applications. TM/MP then sends MQSeries a notification at the completion of the transaction to tell it whether the transaction completed successfully or was backed out. This new notification from TM/MP allows MQSeries to:

- Keep accurate queue depth counts under all conditions
- Keep other internal status information relating to local queues accurate
- Control the availability of non-persistent messages involved in syncpoint operations
- Improve the efficiency and response time of waited MQGET operations

From the system administrator's point of view, the use of OpenTMF is visible in only one way: MQSeries processes are visible in a list of resource managers that can be produced using the STATUS RESOURCEMANAGER command of the TMFCOM utility. All MQSeries processes that use OpenTMF appear in this list as VOLATILE resource managers named automatically by TM/MP.

No special administrative actions are required for this new use of TM/MP—MQSeries uses and manages it automatically. You must ensure that the RMOPENPERCPU (maximum number of VOLATILE and RECOVERABLE resource managers per CPU) configuration parameter of TM/MP is set to a value that is larger than the maximum number of queue servers and status servers that can run in a single CPU across the system. Note that you need to allow for Backup processes since these servers are NonStop process pairs. The default value of 128 is usually adequate for most installations. The *Compaq NSK NonStop TM/MP Configuration and Planning Guide* describes the subject of resource managers and heterogeneous transaction processing.

## NonStop Tuxedo

MQSeries can coordinate messaging operations for OSS applications using NonStop Tuxedo, since this product is based on NonStop TM/MP and uses the same facilities for heterogeneous transaction processing as does MQSeries.

The Compaq NSK NonStop Tuxedo *System Application Development Guide* provides information about the use of the NonStop Tuxedo transaction environment and how it interacts with TM/MP.

## Interleaved application transactions

With MQSeries for Compaq NSK, applications can take advantage of the unique transaction environment in ways that are not possible on other platforms. In general on Compaq NSK, a process may manage multiple transactions concurrently. An update to an audited database is always performed under the control of the current transaction and the application can switch to any one of the other active transactions before committing any of them. This allows an application to perform multiple MQSeries syncpoint messaging operations concurrently.

## MQSeries' critical database files

The critical audited database files for MQSeries are described in Table 7 on page 218.

## Database consistency

*Table 7. Critical audited database files*

| Descriptive Name | Location/Name | Use |
|---|---|---|
| Object Catalog | Data Subvolume/OBJCAT and ABJCAT | Holds the attributes of each queue, process and namelist object as well as the queue manager itself. |
| Non-Client Channel Definitions | Data Subvolume/CHDEFS | Holds attributes of each non-client channel (SENDER, RECEIVER, SERVER, REQUESTER and CLUSTER channel types) |
| Client Channel Definitions | Data Subvolume/CCHDEFS | Holds attributes of each client channel (SVRCONN channel types) |
| OAM database | Data Subvolume/OAMDB | Holds permissions (access rights) for each object and OAM principal authorized to access the queue manager. |
| Principal database | Data Subvolume/PRIDB and PRIDBA | Holds the name of each authorized OAM Principal and the Guardian User Identifier that the principal corresponds to. |
| EC control file | Data Subvolume/RUNTIME | Holds information used by EC Boss and ECs to coordinate startup and recovery operations. |
| Namelist definitions | Data Subvolume/Lxxxxxxx | Holds the content of each namelist object defined. |
| Queue files | Message Subvolume/Qxxxxxxx | Holds one header information and some data for every persistent message on a local queue. |
| Queue overflow files | Message Subvolume/Oxxxxxxx | Holds data for all medium to large size persistent messages on a local queue. |
| Object Touch files | Data and Message Subvolumes/Txxxxxxx | Used to detect administrative changes to the attributes of any object. |
| Channel Sync files | Sync Subvolume/Sxxxxxxx | Holds channel sync information for a channel instance. |

# Critical processes

Table 8 on page 219 describes the critical processes of the queue manager, and shows how MQSeries is protected from and can recover from their failure due to software or system failures. In the table below, disaster refers to cases of multiple system failures, or total system loss.

Table 8. Protection methods used for critical processes

| Process | Protection methods used | Recovery processing |
|---|---|---|
| Queue Server | NonStop process pair. Maintains accurate status of local queues and messages at all times, except in cases of disaster. | Re-initializes from audited databases after catastrophic failure. No other recovery required. NPM will be lost in cases of disaster. |
| Status Server | NonStop process pair. Maintains accurate status of non-local queue objects, and channel status at all times, except in cases of disaster. | Re-initializes from audited databases after catastrophic failure. No other recovery required. |
| Local Queue Manager Agent (LQMA) | Connection is marked as broken for application (2009).<br>The repository manager garbage collection cleans up registration areas of the cluster cache that are left by failed processes.<br>EC, EC Boss, queue server and Status Server immediately recognize failure via NSK IPC connection and correct/adjust status data appropriately.<br>TM/MP aborts any active transaction that was active and used by the process at the time of failure. | None. An LQMA services one connection. The connection is dropped on failure and a new connection must be initiated by the application. |
| Message Channel Agent | Status Server immediately notices failure via NSK IPC connection and marks the channel status appropriately.<br>Status Server ensures retry of outbound channels that fail.<br>Adopt MCA feature can be used to allow restart of the failed channel.<br>Channel Synchronization data is audited by TM/MP and is used by MQSeries to preserve the integrity of the channel. In rare case of in-doubt situation that cannot be resolved automatically, standard administrative facilities exist for resolving.<br>All other protection methods as for the LQMA process. | None. An MCA services one channel. The channel stops on failure and automated facilities of MQSeries exist to restart the channel.<br>MQSeries for Compaq NSK V5.1 also introduces new features such as channel heartbeats and clustering which increase the availability of channels. |

*Table 8. Protection methods used for critical processes (continued)*

| Process | Protection methods used | Recovery processing |
|---|---|---|
| Channel Initiator | Multiple Initiators may be configured to provide higher availability by spreading channel initiation queues across multiple Channel Initiators.<br>The Channel Initiator is normally run as a PATHWAY server class, configured to AUTOSTART on failure a number of times (default 10) within a fixed 10 minute time interval.<br>Standard PATHWAY configuration options can be used to configure alternate CPUs to be used in the event of a CPU failure. | None. Default Status Server maintains accurate channel status under all conditions and is responsible for channel retry. Channel Initiator uses the triggering capabilities of MQSeries to cause initiation. |
| Command Server | Command Server performs administrative commands in syncpoint, so that consistency is maintained.<br>Standard PATHWAY features as for the Channel Initiator. | None. Restart causes a new connection to the queue manager. |
| EC Boss | Standard PATHWAY protection features as described above. | EC Boss coordinates recovery with ECs using the audited RUNTIME file. ECs re-register with the EC Boss and continue processing. |
| EC | Standard PATHWAY protection features as described above. | EC coordinates recovery with EC Boss using the audited RUNTIME file. EC re-registers with the EC Boss and continues processing. |
| TCP/IP Listener | Multiple TCP/IP Listener processes can be configured to provide higher availability by spreading channels across multiple ports, IP addresses or TCP/IP Server processes.<br>Standard PATHWAY protection features as described above. | None. TCP/IP Listener attempts to connect to any of the ports configured for the queue manager on restart that are available. |

Table 8. Protection methods used for critical processes (continued)

| Process | Protection methods used | Recovery processing |
|---|---|---|
| Repository Manager or Repository Cache Manager | On failure of the repository manager or repository cache manager, current or new users of the cache in the same CPU experience no interruption of access to the cache or clustered operations.<br>The repository manager maintains a consistent hardened version of the Cluster Cache on the Repository queue at all times. When changes are made, they are made in syncpoint with the MQGET of the message from the Cluster Command Queue that causes the change, thereby maintaining consistency even if the repository manager fails during the hardening.<br>Standard PATHWAY protection features as described above. | On recovery, a Repository process will become the repository manager if one does not yet exist in the queue manager. Otherwise it will assume the role of a Repository Cache Server for the CPU in which it is running.<br>A repository manager or repository cache manager re-attaches to the cache if it still exists in memory. If not, the cache is reloaded from the disk if it is present. If the disk file doesn't exist, a new cache is created in the CPU and initialized from the Repository queue. |
| Queue Manager Server | Queue Manager Server performs the retrieval of expired messages and generation of expiry reports in syncpoint so that failure of the server will not cause inconsistency.<br>Standard PATHWAY protection features as described above. | A restarted queue manager server will re-synchronize automatically with the queue servers as they report the current set of expired messages requiring reports each time they perform housekeeping. |

*Table 8. Protection methods used for critical processes (continued)*

| Process | Protection methods used | Recovery processing |
|---|---|---|
| Applications – STANDARD-bound | Failures within MQSeries are detected by connection broken or unexpected error returns from MQI calls. The application should call MQDISC and then MQCONN again to reestablish connection with the Queue Manager.<br>The queue manager detects failure of the application process immediately. The queue manager performs an implicit MQDISC on behalf of the application causing all open resources to be closed and released.<br>Any in progress syncpoint operations are aborted by TM/MP and MQSeries reacts to this performing the appropriate adjustments to local queues.<br>Application code that contains errors is not able to corrupt queue manager critical databases or shared resources since the only shared memory that is in the address space of the application is read-only and no critical database files are directly accessed. | None. Applications connect to the queue manager anew. |

*Table 8. Protection methods used for critical processes (continued)*

| Process | Protection methods used | Recovery processing |
|---|---|---|
| Applications – FASTPATH-bound | Failures within MQSeries are detected by unexpected error returns from MQI calls. The application should call MQDISC and then MQCONNX again to reestablish connection with the queue manager.<br>The queue manager detects failure of the application process immediately. The queue manager performs an implicit MQDISC on behalf of the application causing all open resources to be closed and released.<br>Any in progress syncpoint operations are aborted by TM/MP and MQSeries reacts to this by performing the appropriate adjustments to local queues.<br>Application code that contains errors are able to corrupt queue manager critical databases and shared resources since they have access to the read/write Repository Cache and internal Queue Manager structures that have the potential to corrupt the Object Catalog. Note that the Queue structures and messages themselves are safe since only queue servers access them. | None. Applications on restart connect anew. |

## Clusters

MQSeries for Compaq NSK V5.1 clusters are aimed at reducing the administration requirements of an MQSeries network and also to enhance the overall availability and scalability of MQSeries as a distributed service.

Queue Managers that belong to clusters can MQPUT to queues that are advertised to the cluster as if they are local queues. The MQSeries clustering function deals with the administration and management of all the definitions and channels required to transfer the message to the destination queue.

Clustered queues may be defined on more than one queue manager within a cluster. This creates multiple instances of a queue within the cluster. An application puts to only one instance of a queue as chosen by the Cluster Workload Manager (CWLM), a component of MQSeries. This choice may be made when the queue is opened, or dynamically for every put. The CWLM can determine the best instance of a cluster queue to use based on whether the channel to the instance is running or not, and on certain other factors like network priority and also application consideration via the Cluster Workload Management Exit.

Clusters can therefore provide an MQSeries network-level availability enhancement. MQSeries on Compaq NSK is a good choice to act as a Full Repository for clusters due to the reliability and scalability of its operation.

## Configuration considerations for availability

This section summarizes the configuration options enhancing the availability of MQSeries for Compaq NSK V5.1 and its applications on Compaq NSK:

- Configure PATHWAY with alternate CPUs for all server classes to protect against CPU failures
- Consider the use of clustering for enhanced availability of MQSeries network resources
- Consider the use of non-persistent messages with checkpointing enabled to obtain high performance with high availability for suitable message types
- Ensure that your TM/MP configuration is sized to cope with the peak predicted demand of MQSeries and its applications
- Consider using message overflow files to reduce the audit trail requirement for very large messages
- If a disaster recovery requirement exists, consider the use of RDF for creating and maintaining a backup site for MQSeries
- Ensure that CPUs run with enough available physical memory to cope with peak demands of MQSeries and its applications
- Ensure that sufficient swap space is available for the CPUs that hold MQSeries and its applications.

## Configuration considerations for data integrity

This section summarizes the configuration options enhancing the data integrity of MQSeries for Compaq NSK V5.1 and its applications on Compaq NSK:

- Determine which message or transaction types carried by MQSeries require which level of data integrity as provided by persistent messages and non-persistent messages
- Determine whether message overflow files are a suitable storage mechanism for storing any very large messages that you need to use
- Determine whether non-persistent messages require checkpointing or whether only some do. The different types of message should be put to different queues to enable different checkpointing options to be specified
- Ensure that when applications require the highest data integrity that syncpoint operations using persistent messages are employed
- If a disaster recovery requirement exists, configure and use RDF to create and maintain a duplicate backup of the MQSeries environment.

# Part 2. Reference

© Copyright IBM Corp. 1993, 2001

# Chapter 17. The MQSeries control commands

This chapter contains reference material for the control commands used with MQSeries for Compaq NSK.

## Control commands summary

The following control commands are supported by MQSeries for Compaq NSK via TACL macros and compiled programs:

- altmqfls (alter queue file attributes)
- altmqusr (alter MQSeries user information)
- cleanrdf (RDF housekeeping utility)
- cnvclchl (convert client channel definitions)
- crtmqcvx (data conversion)
- crtmqm (create queue manager)
- dltmqm (delete queue manager)
- dspmqaut (display authority)
- dspmqcsv (display command server)
- dspmqfls (display MQSeries file attributes)
- dspmqtrc (display MQSeries formatted trace output)
- dspmqusr (display MQSeries user information)
- endmqcsv (end command server)
- endmqm (end queue manager)
- endmqtrc (end MQSeries trace)
- instmqm (install MQSeries for Compaq NSK)
- runmqchi (run channel initiator)
- runmqchl (run channel)
- runmqdlq (run dead-letter queue handler)
- runmqlsr (run TCP/IP listener)
- runmqsc (run MQSeries commands)
- runmqtrm (start trigger monitor)
- setmqaut (set/reset authority)
- strmqcsv (start command server)
- strmqm (start queue manager)
- strmqtrc (start MQSeries trace)
- upgmqm (upgrade V2.2.0.1 queue manager)

Detailed descriptions of these commands are provided in the remainder of this chapter.

**Notes:**

1. Flags, which are single-character identifiers preceded by a dash (for example, -v on the **runmqsc** command), must be specified in lowercase.
2. Usage messages are displayed if control commands are invoked with -?, ?, or with no parameters when parameters are expected.

## Using names

The names for the following MQSeries objects can be a maximum of 48 characters:

- Queue managers
- Queues
- Process definitions

© Copyright IBM Corp. 1993, 2001

## Names

The maximum length of channel names is 20 characters.

The characters that can be used for all MQSeries names are:
- Uppercase A - Z
- Lowercase a - z
- Numerics 0 - 9
- Period (.)
- Underscore (_)
- Forward slash (/)
- Percent sign (%)

**Notes:**

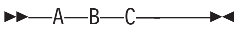
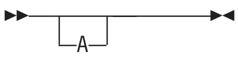
1. Forward slash and percent are special characters. If you use either of these characters in a name, the name must be enclosed in double quotation marks whenever it is used.
2. Leading or embedded blanks are not allowed.
3. National language characters are not allowed.
4. Names may be enclosed in double quotation marks, but this is essential only if special characters are included in the name.

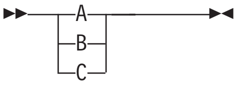
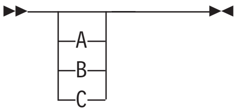
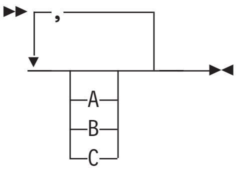
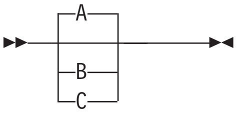
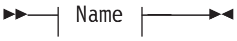
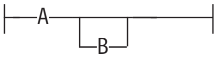
# How to read syntax diagrams

This chapter contains syntax diagrams (sometimes referred to as "railroad" diagrams).

Each syntax diagram begins with a double right arrow and ends with a right and left arrow pair. Lines beginning with a single right arrow are continuation lines. You read a syntax diagram from left to right and from top to bottom, following the direction of the arrows.

Other conventions used in syntax diagrams are:

| Convention | Meaning |
|---|---|
|  | You must specify values A, B, and C. Required values are shown on the main line of a syntax diagram. |
|  | You may specify value A. Optional values are shown below the main line of a syntax diagram. |

| Convention | Meaning |
|---|---|
|  | Values A, B, and C are alternatives, one of which you must specify. |
|  | Values A, B, and C are alternatives, one of which you may specify. |
|  | You may specify one or more of the values A, B, and C. Any required separator for multiple or repeated values (in this example, the comma (,)) is shown on the arrow. |
|  | Values A, B, and C are alternatives, one of which you may specify. If you specify none of the values shown, the default A (the value shown above the main line) is used. |
| <br>**Name:**<br> | The syntax fragment Name is shown separately from the main syntax diagram. |

# altmqfls (Alter queue file attributes)

## Purpose

You use the **altmqfls** command to alter queue file attributes. A single **altmqfls** command can perform only one of the following three groups of operations at any one time:

- Move the message files that belong to a predefined local queue to a different volume to distribute disk I/O across volumes, or
- Change the size of the queue and overflow files associated with a local queue. This operation cannot be performed if the queue is open, or
- Change the queue server options associated with a local queue. These options allow you optimize the way the queue server handles storage associated with the queue, and controls the checkpointing of non-persistent messages. In addition, you can change the queue server associated with a local queue, or the status server associated with an alias, remote or model queue, or process and associate a measure counter with a queue.

**altmqfls** does not permit the queue to be reloaded while it is in use.

**Note:** The user interface and command line options for **altmqfls** have changed substantially from Version 2.2.0.1. The changes reflect the new functionality and provide more meaningful names for command line options. As well, all parameters start with '--' to comply with POSIX rules. You will need to update any scripts that invoke **altmqfls** to reflect the changes because MQSeries Version 5.1 will reject the Version 2.2.0.1 command line options.

## Syntax

```
►►──altmqfls── --type ObjectType──┬──────────────────────┬───────────────────────────►
                                  └─ --qmgr QMgrName─────┘

►──┬──────────────────────────┬──┬────────────────────────────────┬─────────────────►
   └─ --server ServerName─────┘  │                 ┌──────────┐   │
                                 │                 ▼          │   │
                                 └─ --qsoptions──────┬─L─┬────┴───┘
                                                     ├─C─┤
                                                     └─S─┘

►──┬──────────────────────────┬──┬──────────────────────────────────────┬───────────►
   └─ --browse MemBrowsePM────┘  └─ --oflowsize MsgOverflowThresh───────┘

►──┬──────────────────────────┬──┬──────────────────────────────────┬───────────────►
   └─ --volume VolumeName─────┘  └─ --msgofsubvol MsgOvflSubvol─────┘

►──┬──────────────────────────────────┬──ObjectName──────────────────────────────►◄
   └─ --meascount MeasureCounter──────┘
```

## Required parameters

`ObjectName`
Is the name of the permanent local queue whose message files are to be relocated. The queue must not be open, nor must it contain uncommitted messages.

**--type** `ObjectType`
Identifies a permanent queue. `ObjectType` must be specified and may one of the following:

| | |
|---|---|
| **ql or qlocal** | A local queue |
| **qa or qalias** | An alias queue |
| **qr or qremote** | A remote queue |
| **qm or qmodel** | A model queue |
| **proc or process** | A process |

## Optional parameters

**--qmgr** `QMgrName`
Is the name of the queue manager to which the local queue belongs. The queue manager must have been started. If no queue manager name is specified, the default queue manager is used.

**--server** `ServerName`
Is the name of a the status server process or queue server process that is to be responsible for the status data for this object. If the object is a local queue or model queue, use a queue server name. If the object is anything other than a local queue or model queue, use the status server name.

When an object is created, the *ServerName* is set to DEFAULT to indicate that the default server is responsible for the queue. When the responsibility for a queue is changed, the queue must not be in use and all non-persistent messages are discarded during the change.

**--volume** *VolumeName*
Is a Compaq NSK volume name (for example, $DEV). This value is required if you are using **altmqfls** to move message files to a different volume. This parameter can be specified only with type and object name. It is not allowed in combination with the other options.

**--qsoptions LSC**
Use the L, S and C options, either alone or combined, to fine tune the queue's retrievability and reliability.

**Note:** *All* of the --qsoptions SLC are set each time the command is issued. For example, --qsoptions S will unset L and C. The --qsoptions option can be specified once and once only on a command line. You must specify at least one option, but you can specify more than one.

**C** Specifies that the non persistent messages are checkpointed to the backup queue server, providing fault tolerance at the expense of CPU loading required to handle the extra checkpointing, extra IPC messages and extra memory required to store the messages.

Use this option if you want a high degree of recoverability for non-persistent messages.The default for this parameter when a queue is created is to be set (that is, checkpointed).

**L** Specifies that the queue server locks in memory the data structures and chains associated with a queue. Normally the storage associated with a queue is a candidate for removal from the queue server's address space when it is no longer being accessed. Use this option for faster access to a queue's memory data structures at the possible expense of other queues.

The default for this parameter when a queue is created is not set (that is, not locked in memory).

**S** Specifies that the queue server loads the local queue from disk into cache when the queue manager is started up. Normally the messages for a queue are loaded when first referenced by an application. If this option is set, the queue is loaded when the queue manager starts. The default for this parameter when a queue is created is not set (that is, not loaded on startup).

Use this option to reduce CPU and disk IO activity when a queue is first opened, at the cost of an increase in queue server activity (CPU and disk IO) during queue manager startup.

**None** Specifies that no options are set.

**--browse** *MemBrowsePM*
Specifies a maximum number of bytes of data of each persistent message to keep in the queue server's cache (as well as on disk). During a browse operation on a persistent message, the queue manager normally reads the data for a message from disk storage and returns it to the application. If this parameter is set to a value other than zero, the specified number of bytes of data will also be kept in memory and the browse operation will return this data to the application without having to access the disk. By using this parameter, you can increase the memory resources in use by the queue server.

The minimum value of this parameter is zero (0) bytes, the maximum value is 25,000 bytes. The default for this parameter when a queue is created is zero.

**--oflowsize** *MsgOverflowThresh*
Specifies the minimum message size for the use of a message overflow file to store the message data. Persistent messages that are smaller than this threshold are stored in the queue overflow file. Persistent messages of the threshold size or larger will have their bulk data stored in a dedicated message overflow file. The default for this parameter when a queue is created is 200,000 bytes.

**--msgofsubvol** *MsgOvflSubvol*
Specifies a subvolume on the volume that the queue resides on where the queue server creates new message overflow files. All queues will initially use their queue manager message subvolume by default.

**--meascount** *MeasureCounter*
Specifies the name of a MEASURE counter which, if part of an active measurement, is initialized to the current depth and then incremented and decremented by the queue server responsible for the queue when messages are added and removed.

## Return codes

**0** Command completed normally
**10** Command completed but not entirely as expected
**20** An error occurred during processing

## Examples

1. In the following example, message files belonging to the local queue `flint.queue`, owned by queue manager `target.queue.mgr`, are moved to volume $DATA3.

```
altmqfls --qmgr target.queue.mgr --type ql --volume $DATA3 flint.queue
```

2. In the following example, the queue server process name is being changed for an object:

```
altmqfls --qmgr target.queue.mgr --type ql  --server $TQS2 flint.queue
```

This command results in local queue `flint.queue`, which belongs to `target.queue.mgr` being moved to a queue server with a process name of $TQS2.

## Related commands

**dspmqfls** Display MQSeries files

# altmqusr (Alter MQSeries user information)

## Purpose

Use the **altmqusr** command to define or remove a principal corresponding to a Compaq NSK user ID that will have access to MQSeries.

## Syntax

```
►►──altmqusr── -m QMgrName── -p PrincipalName──┬─ -u CompaqUserId─┬──────────────►◄
                                               └─ -r──────────────┘
```

## Description

You can use this command to:

- Create a principal (that is, to grant access to a queue manager to a Compaq NSK user ID).
- Remove a principal (that is, to revoke access to a queue manager from a Compaq user ID).
- Change a principal definition.

When -u *CompaqUserId* is specified, **altmqusr** creates a principal, if one does not already exist, or changes the existing definition. *CompaqUserId* can be specified as a Compaq Administrative user ID or, if SAFEGUARD is running, as a SAFEGUARD alias.

When -r is specified, the principal is deleted from the principal database.

You must specify either -u *CompaqUserId* or -r.

## Required parameters

**-m** *QMgrName*
Is the name of the queue manager to which the principal belongs.

**-p** *PrincipalName*
Is the name of the principal to be created, changed, or removed.

**-u** *CompaqUserId*
Is the Compaq NSK Administrative user ID or SAFEGUARD alias to be associated with the principal definition.

**-r** Specifies that the principal definition is to be removed from the queue manager.

## Return codes

| | |
|---|---|
| **0** | Successful operation |
| **36** | Invalid arguments supplied |
| **69** | Storage not available |
| **71** | Unexpected error |

## Examples

To add a principal mquser1 mapping to a Compaq user ID mqtest.fred:

```
altmqusr -m MT02 -p mquser1 -u mqtest.fred
```

To add a principal mquser2 mapping to group group.user01:

```
altmqusr -m MT02 -p mquser2 -u group.user01
```

To see the results of these commands, use the **dspmqusr** command, as described in "dspmqusr (Display MQSeries user information)" on page 258.

To remove principal mquser1:

```
altmqusr -m MT02 -p mquser1 -r
```

## Related commands

| | |
|---|---|
| **dspmqusr** | Display MQSeries user information |

# cleanrdf (Perform RDF housekeeping)

## Purpose

Use the **cleanrdf** utility to perform routine housekeeping on the primary system queue manager in an RDF environment. The **cleanrdf** utility completes the removal of files that have been logically deleted on both the primary and backup systems. The utility also duplicates some non-audited databases to the correct location on the backup system.

Note that the utility invoked by **cleanrdf** traverses the entire object catalog and message database, so some degradation of performance is likely to occur while the utility is running.

The non-audited database files duplicated to the backup system by the utility are:
- MQERRLG1
- MQSINI
- QMINI
- PATHCTL
- SHUTDOWN
- SYNCHIDX
- TRACEOPT
- UMQSINI

## Syntax

```
►►──cleanrdf──-b BkupSystem──┬──────────────┬──────────────────────────►◄
                             └─-m QMgrName──┘
```

## Required parameters

**-b** *BkupSystem*

Is the Compaq NSK system name of the RDF backup site for this queue manager. *BkupSystem* is specified in the form \*name* (as is standard in the Compaq NSK environment).

## Optional parameters

**-m** *QMgrName*

Is the name of the queue manager for which **cleanrdf** is to be run. If no queue-manager name is specified, **cleanrdf** is run against the default queue manager.

## Return codes

**0** Command completed normally
**20** An error occurred during processing

## Examples

In the following example, **cleanrdf** is run against the queue manager `test.queue.mgr`. Compaq NSK node `\HAWK` has been configured as the backup RDF site for this queue manager.

```
cleanrdf -b \HAWK -m test.queue.mgr
```

# cnvclchl (Convert client channel definitions)

## Purpose

Use the **cnvclchl** command to convert the client channel definition file, created for CLNTCONN channels by MQSC, from a Compaq structured file to an unstructured format acceptable to MQSeries clients.

**Note:**

> Version 2.2.0.1 definition files do not work with Version 5.1 clients because the format of the records have changed. After MQSeries for Compaq NSK V5.1 is installed, you can rerun the **cnvclchl** command to create definition files that will work with Version 5.1 clients.

## Syntax

```
►►──cnvclchl── -m QMgrName──┬─────────────────┬──────────────────────────►◄
                            └─ -o OutputFile──┘
```

## Required parameters

**-m** *QMgrName*
Identifies the queue manager that owns the channel definitions file (CCHDEFS) to be converted. This value is required.

## Optional parameters

**-o** *OutputFile*
Identifies the file that will contain the converted definitions. The default filename is AMQCLCHL.

## Examples

The following command converts the Compaq structured client channel definition file for queue manager MT01 to an unstructured file. Two client connection channel definitions are contained in the output file AMQCLCHL, SYSTEM.DEF.CLNTCONN and SOLARIS_TO_Compaq:

```
$DATA01 SZMON 330> cnvclchl -m MV4

MQSeries client channel table being converted

Opening Compaq NSK v5.1 CLNTCONN table

Opening Common v5.1 CLNTCONN table AMQCLCHL for output

Writing Common v5.1 CLNTCONN table entry for SYSTEM.DEF.CLNTCONN

Closing Compaq NSK v5.1 CLNTCONN table
Closing Common v5.1 CLNTCONN table
MQSeries client channel table conversion complete.
```

# crtmqcvx (Data conversion)

## Purpose

Use the **crtmqcvx** command to create a fragment of code that performs data conversion on data type structures. The command generates a C function that can be used in an exit to convert your C structures.

The command reads an input file containing a structure or structures to be converted. It then writes an output file containing a code fragment or fragments to convert those structures.

For further information about this command and how to use it, refer to the *MQSeries Application Programming Guide*.

## Syntax

```
►►──crtmqcvx──SourceFile──TargetFile──────────────────────────────────────►◄
```

## Required parameters

*SourceFile*
Specifies the input file containing the C structures to be converted.

*TargetFile*
Specifies the output file containing the code fragments generated to convert the structures.

## Return codes

| | |
|---|---|
| **0** | Command completed normally |
| **10** | Command completed with unexpected results |
| **20** | An error occurred during processing |

## Examples

The following example shows the results of using the data conversion command against a source C structure. The command issued is:

```
crtmqcvx source target
```

The input file, `source` looks like this:

```
/* This is a test C structure which can be converted by the */
/* crtmqcvx utility                                         */

struct my_structure
{
    int    code;
    MQLONG value;
};
```

The output file, `target`, produced by the command is shown below. You can use these code fragments in your applications to convert data structures. However, if you do so, you should understand that the fragment uses macros supplied in the MQSeries header file MQSVMHTH in subvolume ZMQSLIB.

```
MQLONG Convertmy_structure(
             PMQBYTE *in_cursor,
             PMQBYTE *out_cursor,
             PMQBYTE in_lastbyte,
             PMQBYTE out_lastbyte,
             MQHCONN hConn,
             MQLONG  opts,
             MQLONG  MsgEncoding,
             MQLONG  ReqEncoding,
             MQLONG  MsgCCSID,
             MQLONG  ReqCCSID,
             MQLONG  CompCode,
             MQLONG  Reason)
{
    MQLONG ReturnCode = MQRC_NONE;

    ConvertLong(1); /* code */

    AlignLong();
    ConvertLong(1); /* value */

Fail:
    return(ReturnCode);
}
```

# crtmqm (Create queue manager)

## Purpose

Use the **crtmqm** command to create a local queue manager. Once a queue manager has been created, use the **strmqm** command to start it.

Creating a queue manager automatically creates the associated system and default objects.

## Syntax

```
►►──crtmqm──┬─────────┬──┬──────────────────────────────┬──┬───────────┬──────────►
            └─ -c Text─┘  └─ -d DefaultTransmissionQueue─┘  └─ -e NumECs─┘

►──┬────────────────────────┬──┬─────────┬──┬───────────────┬──────────────────────►
   └─ -h MaximumHandleLimit─┘  └─ -l CCSID─┘  └─ -m MachIniFile─┘

►──┬──────────────────┬──┬─────┬──┬──────────────────┬─────────────────────────────►
   └─ -p DefaultPrefix─┘  └─ -q ─┘  └─ -t IntervalValue─┘

►──┬────────────────────┬──┬─────────────────────────────────┬──┬─────┬────────────►
   └─ -u DeadLetterQueue─┘  └─ -x MaximumUncommittedMessages─┘  └─ -z ─┘

►─ -n PATHMONProcessName── -o HomeTerminalName── -s StatusServerName──────────────►

►─ -v QueueServerName──QMgrName───────────────────────────────────────────────────►◄
```

## Required parameters

**-n** *PATHMONProcessName*
The process name of the TS/MP PATHMON process for the queue manager. This process name must be unique in the system.

**-o** *HomeTerminalName*
Home terminal device name. ($DDDD.#SS). For example, $TRM1.#A.

**-s** *StatusServerName*
The process name to be given to the default status server for the queue manager. The process name must be unique in the system.

**-v** *QueueServerName*
Specifies a unique process name to be given to the default queue server process for this queue manager.

*QMgrName*
The name of the queue manager to be created. The name can contain up to 48 characters. This must be the last item in the command.

## Optional parameters

**-c** *Text*

Some text (up to 64 characters) that describes this queue manager. The default is all blanks.

If special characters are required, the description must be enclosed in double quotation marks.

**-d** *DefaultTransmissionQueue*

The name of the local transmission queue that remote messages are placed on if a transmission queue is not explicitly defined for their destination. There is no default.

**-e** *NumECs*

The number of EC processes in the queue manager. The default is 1.

**-h** *MaximumHandleLimit*

In MQSeries for Compaq NSK, this parameter is ignored.

The maximum number of handles that any one application can have open at the same time. Specify a value in the range 1 through 999 999 999. The default value is 256.

**-l** *CCSID*

Qmgr CCSID. The default value is 819.

**-m** *MachIniFile*

Overrides the default MQSINI file location and that specified in the environment variable MQMACHINIFILE.

**-p** *DefaultPrefix*

The volume for the queue manager. Overrides the `QMDefaultVolume` entry in the MQSINI file.

**-q** Specifies that this queue manager is to be made the default queue manager. The new queue manager replaces any existing queue manager as the default.

If you accidentally use this flag and wish to revert to an existing queue manager as the default queue manager, you can edit the `DefaultQueueManager` stanza in the MQSeries configuration file.

**-t** *IntervalValue*

The trigger-time interval in milliseconds for all queues controlled by this queue manager. This value specifies the time after the receipt of a trigger-generating message when triggering is suspended. That is, if the arrival of a message on a queue causes a trigger message to be put on the initiation queue, any message arriving on the same queue within the specified interval does not generate another trigger message.

You can use the trigger time interval to ensure that your application is allowed sufficient time to deal with a trigger condition before it is alerted to deal with another on the same queue. You may wish to see all trigger events that happen; if so, set a low or zero value in this field.

Specify a value in the range 0 through 999 999 999. The default is 999 999 999 milliseconds, a time of more than 11 days. Allowing the default to be taken effectively means that triggering is disabled after the first trigger message. However, triggering can be reenabled by an application servicing the queue using an alter queue command to reset the trigger attribute.

**-u** *DeadLetterQueue*
The name of the local queue that is to be used as the dead-letter (undelivered-message) queue. Messages are put on this queue if they cannot be routed to their correct destination.

By default, there is no dead-letter queue.

**-x** *MaximumUncommittedMessages*
In MQSeries for Compaq NSK, this parameter is ignored.

Specifies the maximum number of uncommitted messages under any one syncpoint. That is, the sum of:

- The number of messages that can be retrieved from queues
- The number of messages that can be put on queues
- Any trigger messages generated within this unit of work

This limit does not apply to messages that are retrieved or put outside syncpoint control.

Specify a value in the range 1 through 10 000. The default value is 1000 uncommitted messages.

**-z** Suppresses error messages.

This flag is normally used within MQSeries to suppress unwanted error messages. As use of this flag could result in loss of information, you are recommended not to use it when entering commands on a command line.

## Return codes

| | |
|---|---|
| **0** | Queue manager created |
| **8** | Queue manager already exists |
| **49** | Queue manager stopping |
| **69** | Storage not available |
| **70** | Queue space not available |
| **71** | Unexpected error |
| **72** | Queue manager name error |
| **111** | Queue manager created. However, there was a problem processing the default queue manager definition in the product configuration file. The default queue manager specification may be incorrect. |

## Examples

1. This command creates a default queue manager named `Paint.queue.manager`, which is given a description of `Paint Shop`:

```
crtmqm -c "Paint Shop" -n $PANT -o $TRM1.#A -s $PNT1 -v $PQS1 Paint.queue.manager
```

2. In this example, another queue manager, `travel`, is created. The trigger interval is defined as 5000 milliseconds (or 5 seconds) and its dead-letter queue is specified as SYSTEM.DEAD.LETTER.QUEUE.

```
crtmqm -t 5000 -u SYSTEM.DEAD.LETTER.QUEUE -n $TRAV -o $TRM1.#A -s $TRV1 -v $TQS1 travel
```

Once a trigger event is generated, further trigger events are disabled for five seconds.

## Related commands

| | |
|---|---|
| **strmqm** | Start queue manager |
| **endmqm** | End queue manager |
| **dltmqm** | Delete queue manager |

# dltmqm (Delete queue manager)

## Purpose

Use the **dltmqm** command to delete a specified queue manager. All objects associated with this queue manager are also deleted. Before you can delete a queue manager you must end it using the **endmqm** command.

## Syntax

```
►►──dltmqm──┬──────┬──QMgrName──────────────────────────────────────►◄
            └─ -z ─┘
```

## Required parameters

*QMgrName*
Specifies the name of the queue manager to be deleted.

## Optional parameters

**-z** Suppresses error messages.

## Return codes

| | |
|---|---|
| **0** | Queue manager deleted |
| **5** | Queue manager running |
| **16** | Queue manager does not exist |
| **69** | Storage not available |
| **71** | Unexpected error |
| **72** | Queue manager name error |
| **112** | Queue manager deleted. However, there was a problem processing the default queue manager definition in the product configuration file. The default queue manager specification may be incorrect. |

## Examples

1. The following command deletes the queue manager `saturn.queue.manager`:

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

2. The following command deletes the queue manager `travel` and also suppresses any messages caused by the command:

```
dltmqm -z travel
```

## Related commands

| | |
|---|---|
| **crtmqm** | Create queue manager |
| **strmqm** | Start queue manager |
| **endmqm** | End queue manager |

# dspmqaut (Display authority)

## Purpose

Use the **dspmqaut** command to display the current authorizations to a specified object. Only one group may be specified.

If a user ID is a member of more than one group, examine the authorizations of each group to determine all the authorizations that apply to the user ID.

## Syntax

```
►►──dspmqaut──┬─────────────────┬──┬──────────────────┬── -t ObjectType──────────►
              └─ -m QMgrName────┘  └─ -n ObjectName───┘

►──┬─ -g GroupName─────┬──┬─────────────────────────┬───────────────────────────►◄
   └─ -p PrincipalName─┘  └─ -s ServiceComponent────┘
```

## Required parameters

**-g** *GroupName*
Specifies the name of the user group on which the inquiry is to be made. You can specify only ***one*** name, which must be the name of an existing user group. You must specify either -g *GroupName* or -p *PrincipalName*.

**-p** *PrincipalName*
Specifies the name of the principal for which the authorizations to the specified object are to be displayed. You must specify either -g *GroupName* or -p *PrincipalName*.

**-t** *ObjectType*
Specifies the type of object on which the inquiry is to be made. Possible values are:

| | |
|---|---|
| **queue or q** | A queue or queues matching the object type parameter |
| **qmgr** | A queue manager object |
| **process or prcs** | A process |
| **namelist or nl** | A namelist |

## Optional parameters

**-m** *QMgrName*
Specifies the name of the queue manager on which the inquiry is to be made.

**-n** *ObjectName*
Specifies the name of the object on which the inquiry is to be made.

This is a required parameter ***unless*** it is the queue manager itself.

You must specify the name of a queue manager, queue, or process definition.

**-s** *ServiceComponent*
This parameter applies only if you are using installable authorization services, otherwise it is ignored.

If installable authorization services are supported, this parameter specifies the name of the authorization service to which the authorizations apply. This parameter is optional; if it is not specified, the authorization update is made to the first installable component for the service.

## Returned parameters

This command returns an authorization list, which can contain none, one, or more authorization parameters. Each authorization parameter returned means that any user ID in the specified group has the authority to perform the operation defined by that parameter.

Table 9 shows the authorities that can be given to the different object types.

*Table 9. Security authorities from the dspmqaut command*

| Authority | Queue | Process | Qmgr | Namelist |
|---|---|---|---|---|
| all | ✓ | ✓ | ✓ | ✓ |
| alladm | ✓ | ✓ | ✓ | ✓ |
| allmqi | ✓ | ✓ | ✓ | ✓ |
| altusr | | | ✓ | |
| browse | ✓ | | | |
| chg | ✓ | ✓ | ✓ | ✓ |
| chgaut | ✓ | ✓ | ✓ | |
| clr | ✓ | | | |
| connect | | | ✓ | |
| crt | ✓ | ✓ | ✓ | ✓ |
| dlt | ✓ | ✓ | ✓ | ✓ |
| dsp | ✓ | ✓ | ✓ | ✓ |
| get | ✓ | | | |
| inq | ✓ | ✓ | ✓ | ✓ |
| passall | ✓ | | | |
| passid | ✓ | | | |
| put | ✓ | | | |
| set | ✓ | ✓ | ✓ | |
| setall | ✓ | | ✓ | |
| setid | ✓ | | ✓ | |

The following list defines the authorizations associated with each parameter:

**all** Use all operations relevant to the object.

**alladm** Perform all administration operations relevant to the object.

**allmqi** Use all MQI calls relevant to the object.

**altusr** Specify an alternate user ID on an MQI call.

**browse** Retrieve a message from a queue by issuing an MQGET call with the BROWSE option.

**chg** Change the attributes of the specified object, using the appropriate command set.

**chgaut** Specify authorizations for other groups of users on the object, using the **setmqaut** command.

**clr** Clear a queue (PCF command Clear queue only).

**connect** Connect the application to the specified queue manager by issuing an MQCONN call.
**crt** Create objects of the specified type, using the appropriate command set.
**dlt** Delete the specified object, using the appropriate command set.
**dsp** Display the attributes of the specified object, using the appropriate command set.
**get** Retrieve a message from a queue by issuing an MQGET call.
**inq** Make an inquiry on a specific queue by issuing an MQINQ call.
**passall** Pass all context.
**passid** Pass the identity context.
**put** Put a message on a specific queue by issuing an MQPUT call.
**set** Set attributes on a queue from the MQI by issuing an MQSET call.
**setall** Set all context on a queue.
**setid** Set the identity context on a queue.

The authorizations for administration operations, where supported, apply to these command sets:

- Control commands
- MQSC commands
- PCF commands

## Return codes

| | |
|---|---|
| **0** | Successful operation |
| **36** | Invalid arguments supplied |
| **40** | Queue manager not available |
| **49** | Queue manager stopping |
| **69** | Storage not available |
| **71** | Unexpected error |
| **72** | Queue manager name error |
| **133** | Unknown object name |
| **145** | Unexpected object name |
| **146** | Object name missing |
| **147** | Object type missing |
| **148** | Invalid object type |
| **149** | Entity name missing |

## Examples

The following example shows a command to display the authorizations on queue manager `saturn.queue.manager` associated with user group `staff`:

```
dspmqaut -m saturn.queue.manager -t qmgr -g staff
```

The results from this command are:

```
Entity staff has the following authorizations for object :
        get
        browse
        put
        inq
        set
        connect
        altusr
        passid
        passall
        setid
```

## Related commands

| | |
|---|---|
| **setmqaut** | Set or reset authority |

# dspmqcsv (Display command server)

## Purpose

Use the **dspmqcsv** command to display the status of the command server for the specified queue manager.

The status can be one of the following:

- Starting
- Running
- Running with SYSTEM.ADMIN.COMMAND.QUEUE not enabled for gets
- Ending
- Stopped

## Syntax

```
►►──dspmqcsv──QMgrName──────────────────────────────────────────────────►◄
```

## Required parameters

*QMgrName*
Specifies the name of the local queue manager for which the command server status is being requested.

## Return codes

| | |
|---|---|
| **0** | Command completed normally |
| **10** | Command completed with unexpected results |
| **20** | An error occurred during processing |

## Examples

The following command displays the status of the command server associated with `venus.q.mgr`:

```
dspmqcsv venus.q.mgr
```

## Related commands

| | |
|---|---|
| **strmqcsv** | Start a command server |
| **endmqcsv** | End a command server |

# dspmqfls (Display MQSeries file attributes)

## Purpose

Use the **dspmqfls** command to display the real file system name for all MQSeries objects that match a specified criterion. You can use this command to identify the files associated with a particular MQSeries object. This is useful for backing up specific objects. See "Volume structure" on page 56 for further information about name transformation.

You can also use the **dspmqfls** command to display the current settings for the queue server process-name and options of an object.

## Syntax

```
►►──dspmqfls──┬─────────────────┬──┬─────────────────┬──ObjectName────────────►◄
              └─ -t ObjectType──┘  └─ -m QMgrName───┘
```

## Required parameters

*ObjectName*
: Specifies the name of the MQSeries object. The name is a string with no flag and is a required parameter. If the name is omitted an error is returned.

This parameter supports a wild card character * at the end of the string.

## Optional parameters

**-m** *QMgrName*
: Specifies the name of the queue manager for which files are to be examined. If this parameter is omitted, the command operates on the default queue manager.

**-t** *ObjType*
: Specifies the MQSeries object type. The following list shows the valid object types. The abbreviated name is shown first followed by the full name.

| | |
|---|---|
| *** or all** | All object types; this is the default |
| **q or queue** | A queue or queues matching the object name parameter |
| **ql or qlocal** | A local queue |
| **qa or qalias** | An alias queue |
| **qr or qremote** | A remote queue |
| **qm or qmodel** | A model queue |
| **qmgr** | A queue manager object |
| **prcs or process** | A process |
| **ctlg or catalog** | An object catalog |
| **nl or namelist** | A namelist |

**Note:** The **dspmqfls** command displays the names of all the files for the queue.

## Return codes

**0** Command completed normally

**10** Command completed but not entirely as expected
**20** An error occurred during processing

## Examples

1. The following command displays the details of all objects with names beginning SYSTEM.ADMIN that are defined on the default queue manager:

```
dspmqfls SYSTEM.ADMIN*
```

2. The following command displays file details for all processes with names beginning PROC defined on queue manager RADIUS:

```
dspmqfls -m RADIUS -t prcs PROC*
```

3. The following command displays file information for MY.LOCAL.QUEUE:

```
dspmqfls -m MT02 -t q MY.LOCAL.QUEUE

MQSeries Display MQ Files

CONNECTING.
QLOCAL    MY.LOCAL.QUEUE
    $DATA0.MT02M.QMYXLOCA
    $DATA0.MT02M.OMYXLOCA
    $DATA0.MT02M.TMYXLOCA

    Queue/Status Server:               DEFAULT
    Persistent message browse buffer:  0
    Message overflow threshold:        200000
    Queue Server Options:              None
    Message overflow subvolume:        QMGR001O
    Measure Counter:
```

   - QMYXLOCA is the queue file, OMYXLOCA is the overflow file for the queue, and TMYXLOCA is the touch file in the queue manager's data directory.

4. The following example shows an **altmqfls** command that sets load on startup and checkpoint NPM queue server options. The **dspmqfls** command displays the results of the **altmqfls** command.

```
altmqfls --qmgr MT02 --type ql --qsoptions SC MY.LOCAL.QUEUE

dspmqfls -m MT02 -t q MY.LOCAL.QUEUE

MQSeries Display MQ Files

CONNECTING.
QLOCAL    MY.LOCAL.QUEUE
    $DATA4.MT02M.QMYXLOCA
    $DATA4.MT02M.OMYXLOCA
    $DATA4.MT02M.TMYXLOCA

 Queue/Status Server:               DEFAULT
    Persistent message browse buffer:  0
    Message overflow threshold:        200000
    Queue Server Options:              SC
    Message overflow subvolume:        QMGR001O
    Measure Counter:
```

## Related commands

| | |
|---|---|
| **altmqfls** | Alter queue volume |

# dspmqtrc (Display MQSeries formatted trace output)

## Purpose

Use the **dspmqtrc** command to display MQSeries formatted trace output. For more information about using MQSeries trace, see “Using MQSeries trace” on page 198.

## Syntax

```
►►──dspmqtrc── -t FormatTemplate ──InputFileName─────────────────────────────►◄
```

## Required parameters

*InputFileName*
Specifies the name of the file containing the unformatted trace. For example `$DATA.MQTRACE.AMQ12345.`.

**-t** *FormatTemplate*
Specifies the name of the template file containing details of how to display the trace. A trace-format template file, AMQTRC, is provided in subvolume ZMQSSMPL.

## Related commands

| | |
|---|---|
| **endmqtrc** | End MQSeries trace |
| **strmqtrc** | Start MQSeries trace |

# dspmqusr (Display MQSeries user information)

## Purpose

Use the **dspmqusr** command to display information about a specified principal, or all principals for the queue manager.

## Syntax

```
►►──dspmqusr── -m QMgrName──┬──────────────────┬──────────────────────►◄
                            └─ -p PrincipalName─┘
```

## Description

You can use this command to:

- Display all principals, or a particular principal, defined for a queue manager.
- Display the Compaq NSK Administrative and SAFEGUARD file-sharing groups corresponding to the Compaq NSK user ID associated with each principal.

## Required parameters

**-m** *QMgrName*
Is the name of the queue manager to which the principals belong.

## Optional parameters

**-p** *PrincipalName*
Is the name of the principal to be displayed.

## Return codes

**0** Successful operation
**36** Invalid arguments supplied
**69** Storage not available
**71** Unexpected error

## Examples

1. This example shows **dspmqusr** for a newly created queue manager:

```
dspmqusr -m MT02

Principal    Userid  Username              Alias GroupName         GroupType
               0.1
NOBODY         0.0
mqm           20.255 MQM.MANAGER              n    MQM                    a
```

The principal database contains the principal mqm, which maps to the user name of the user who created the queue manager.

2. This example shows output from **dspmqusr** after additional principals have been added with **altmqusr**:

```
dspmqusr -m MT02

Principal    Userid  Username               Alias GroupName          GroupType
               0.1
NOBODY         0.0
mqm           20.255 MQM.MANAGER               n    MQM                    a
mquser1       50.3   MQTEST.FRED               n    MQTEST                 a
                                                    MQM                    s
mquser2        1.1   GROUP.USER01              n    GROUP                  a
```

Principal `mquser1`, which maps to Compaq user ID `MQTEST.FRED`, has been added. `FRED` is a member of group `MQTEST` and a member of group MQM using SAFEGUARD aliasing.

Principal `mquser2` maps to Compaq user ID `GROUP.USER01`.

## Related commands

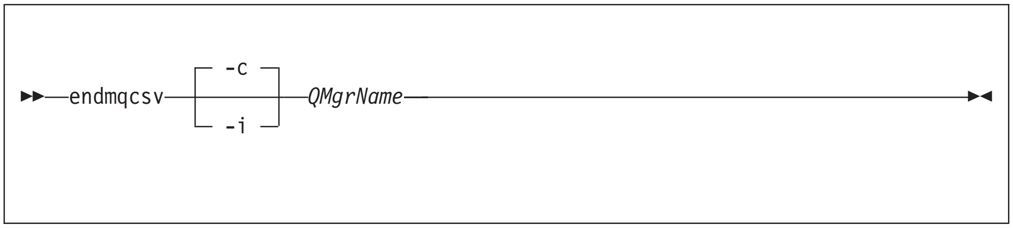
| | |
|---|---|
| **altmqusr** | Alter MQSeries user information |

# endmqcsv (End command server)

## Purpose

Use the **endmqcsv** command to stop the command server on the specified queue manager.

## Syntax



## Required parameters

*QMgrName*
Specifies the name of the queue manager for which the command server is to be ended.

## Optional parameters

**-c** Specifies that the command server is to be stopped in a controlled manner. The command server is allowed to complete the processing of any command message that it has already started. No new message is read from the command queue.

This is the default.

**-i** Specifies that the command server is to be stopped immediately. Actions associated with a command message currently being processed may not be completed.

## Return codes

| | |
|---|---|
| **0** | Command completed normally |
| **10** | Command completed with unexpected results |
| **20** | An error occurred during processing |

## Examples

1. The following command stops the command server on queue manager `saturn.queue.manager`:

```
endmqcsv  -c saturn.queue.manager
```

The command server can complete processing any command it has already started before it stops. Any new commands received remain unprocessed in the command queue until the command server is restarted.

2. The following command stops the command server on queue manager `pluto` immediately:

```
endmqcsv -i pluto
```

## Related commands

| | |
|---|---|
| **strmqcsv** | Start a command server |
| **dspmqcsv** | Display the status of a command server |

# endmqm (End queue manager)

## Purpose

Use the **endmqm** command to end (stop) a specified local queue manager. This command stops a queue manager in one of three modes:

- Normal or quiesced shutdown
- Immediate shutdown
- Preemptive shutdown

The attributes of the queue manager and the objects associated with it are not affected. You can restart the queue manager using the **strmqm** (Start queue manager) command.

To delete a queue manager, you must stop it and then use the **dltmqm** (Delete queue manager) command.

## Syntax

```
              ┌─ -c ─┐
►►──endmqm────┼──────┼───┬──────┬───┬───────────┬─────────────────────────►◄
              ├─ -i ─┤   └─ -z ─┘   └─QMgrName──┘
              └─ -p ─┘
```

## Optional parameters

*QMgrName*
: Is the name of the message queue manager to be stopped. If no name is specified, the default queue manager is stopped.

**-c** Controlled (or quiesced) shutdown. The queue manager stops but only after all applications have disconnected. Any MQI calls currently being processed are completed. This is the default.

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

**-p** Preemptive shutdown.

***Use this type of shutdown only in exceptional circumstances.*** For example, when a queue manager does not stop as a result of a normal **endmqm** command.

The queue manager stops without waiting for applications to disconnect or for MQI calls to complete. This can give unpredictable results for MQI applications. All processes in the queue manager that fail to stop are terminated 30 seconds after the command is issued.

**-z** Suppresses error messages on the command.

## Return codes

**0** Queue manager ended

| | |
|---|---|
| **16** | Queue manager does not exist |
| **36** | Invalid arguments |
| **40** | Queue manager not available |
| **69** | Storage not available |
| **71** | Unexpected error |
| **72** | Queue manager name error |

## Examples

The following examples show commands that end (stop) the specified queue managers.

1. This command ends the default queue manager in a controlled way. All applications currently connected are allowed to disconnect.

```
endmqm
```

2. This command ends the queue manager named `saturn.queue.manager` immediately. All current MQI calls complete, but no new ones are allowed.

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

## Related commands

| | |
|---|---|
| **crtmqm** | Create a queue manager |
| **strmqm** | Start a queue manager |
| **dltmqm** | Delete a queue manager |

# endmqtrc (End MQSeries trace)

## Purpose

Use the **endmqtrc** command to end tracing for a specified queue manager.

For more information about using MQSeries trace, see “Using MQSeries trace” on page 198.

## Syntax

```
►►──endmqtrc──┬─ -a ──────────────────┬────────────────────────────────────►◄
              └─ -m QMgrName──┬──────┬┘
                              └─ -e ─┘
```

## Required parameters

**-m** *QMgrName*
Is the name of the queue manager for which tracing is to be ended.

A queue manager name can be specified on the same command as the -e flag.

**-a** If this flag is specified, all tracing is ended.

This flag ***must*** be specified alone.

## Optional parameters

**-e** If this flag is specified, early tracing is ended on the named queue manager.

## Return codes

| | |
|---|---|
| **AMQ5611** | This message is issued if arguments that are not valid are supplied to the command. |

## Examples

This command ends tracing of data for a queue manager called `QM1`:

```
endmqtrc -m QM1
```

## Related commands

| | |
|---|---|
| **dspmqtrc** | Display formatted trace output |
| **strmqtrc** | Start MQSeries trace |

# instmqm (Install MQSeries for Compaq NSK)

## Purpose

Use the **instmqm** command to install MQSeries for Compaq NSK or update license information.

## Syntax

```
►►──instmqm──┬──────┬──────────────────────────────────────────────►◄
             └─ -l ─┘
```

## Optional parameters

**-l** Invokes **instmqm** for license information updates.

# runmqchi (Run channel initiator)

## Purpose

Use the **runmqchi** command to run a channel initiator process. For more information about the use of this command, refer to the *MQSeries Intercommunication* book.

## Syntax

```
►►──runmqchi──┬─────────────────────┬──┬───────────────┬──────────────►◄
              └─ -q InitiationQName─┘  └─ -m QMgrName──┘
```

## Optional parameters

**-q** *InitiationQName*
Specifies the name of the initiation queue to be processed by this channel initiator. If no value is specified, SYSTEM.CHANNEL.INITQ is used.

**-m** *QMgrName*
Specifies the name of the queue manager on which the initiation queue exists. If the name is omitted, the default queue manager is used.

## Return codes

| | |
|---|---|
| **0** | Command completed normally |
| **10** | Command completed with unexpected results |
| **20** | An error occurred during processing |

If errors occur that result in return codes of either 10 or 20, you should review the queue manager error log that the channel is associated with for the error messages. You should also review the system error log, as problems that occur before the channel is associated with the queue manager are recorded there. For more information about error logs, see “Error logs” on page 195.

# runmqchl (Run channel)

## Purpose

Use the **runmqchl** command to start either a sender (SDR), requester (RQSTR), or fully qualified server channel.

The channel runs asynchronously. To stop the channel, issue the MQSC command STOP CHANNEL.

## Syntax

```
►►──runmqchl── -c ChannelName──┬─────────────────┬───────────────────────►◄
                               └─ -m QMgrName──┘
```

## Required parameters

**-c** *ChannelName*
Specifies the name of the channel to start.

## Optional parameters

**-m** *QMgrName*
Specifies the name of the queue manager with which this channel is associated. If no name is specified, the default queue manager is used.

## Return codes

**0** Command completed normally
**10** Command completed with unexpected results
**20** An error occurred during processing

If return codes 10 or 20 are generated, review the error log of the associated queue manager for the error messages. You should also review the MQSeries system error logs (located in ZMQSSYS) because problems that occur before the channel is associated with the queue manager are recorded there.

# runmqdlq (Run dead-letter queue handler)

## Purpose

Use the **runmqdlq** command to start the dead-letter queue (DLQ) handler, a utility that processes messages on a dead-letter queue.

## Syntax

```
►►──runmqdlq──┬──────────────────────────────┬──┬─────────────┬──────────────────────►◄
              └─ QName ──┬────────────────┬──┘  └─ RulesTable─┘
                         └─ QMgrName ─────┘
```

## Optional parameters

*QName*
Is the name of the dead-letter queue to be processed.

If you specify a *QName* value, it overrides any INPUTQ value specified in a rules table. If no (nonblank) name is specified either on input to **runmqdlq** or in the rules table, the dead-letter queue associated with the queue manager named on the *QMgrName* parameter is processed.

*QMgrName*
Is the name of the queue manager that owns the queue to be processed.

If you specify a *QMgrName* value, it overrides any INPUTQM value specified in a rules table. If no (nonblank) name is specified either on input to **runmqdlq** or in the rules table, the queue is assumed to belong to the default queue manager.

*RulesTable*
Is the name of the file containing the rules table, which must contain at least one rule.

By default, the **runmqdlq** command takes its input from the standard IN file. When the command is processed, the results and a summary are put into a report that is sent to the standard OUT file. Alternatively, by redirecting the input from a file, you can apply a rules table to the specified queue.

If no rules table is specified on input to **runmqdlq**, rules and actions must be specified interactively.

In this case, the DLQ handler:

- Reads its input from the keyboard.
- Does not start to process the named queue until it receives an end_of_file (Ctrl-Y) character.

  The MQSC rules for comment lines and for joining lines also apply to the DLQ handler input parameters.

For more information about rules tables and how to construct them, see “DLQ handler rules table” on page 146.

# runmqlsr (Run listener)

## Purpose

The runmqlsr (Run listener) command runs a TCP/IP listener process.

## Syntax

```
►►──runmqlsr──┬────────┬──┬─────────┬──┬─────────────┬──────────────►◄
              └─-t tcp─┘  └─-p Port─┘  └─-m QMgrName─┘
```

## Description

When run from a TACL prompt, **runmqlsr** does not return the control to the TACL prompt until the listener terminates. That is, **runmqlsr** is run waited.

The TACL prompt returns only if there is a failure or the listener stops. If the terminal (TACL) is stopped before **runmqlsr**, the listener is unable to access its home terminal or out file. Before **runmqlsr** is invoked, all PARAMs (such as MQEMSEVENTS) must be defined.

For these reasons, you are recommended to start and stop the listener from the queue manager's PATHWAY, which gives a greater degree of control.

## Optional parameters

**-p** *Port*

Port number for TCP/IP. If a value is not specified, the port number specified on a `TCPListenerPort` entry in the `TCPConfig` stanza in the QMINI file is used. The default value is 1414. If multiple listener ports are defined in QMINI, the next available port is used.

If the PARAM MPORTNUMBER is specified in the TACL environment, or a PATHWAY server class definition for the program, **runmqlsr** listens on the specified port, instead of the one in the QMINI file.

If none of the ports specified in QMINI is free, or the port specified on the **runmqlsr** command is not available, **runmqlsr** fails.

**-m** *QMgrName*

Specifies the name of the queue manager. If no name is specified, the command operates on the default queue manager.

**-t tcp**

Identifies TCP/IP as the transmission protocol.

If the DEFINE =TCPIPˆPROCESSˆNAME exists in the TACL environment, or a PATHWAY server class definition for the program, **runmqlsr** uses the Guardian TCP/IP server process instead of the default.

If the PARAM MQPORTNUMBER is specified, or PATHWAY server class definition for the program is specified, runmqlsr listens on the specified port, instead on the one listed in the QMINI file.

This is the only valid value (and the default) in MQSeries for Compaq NSK.

## Return codes

**0** Command completed normally

**10** Command completed with unexpected results

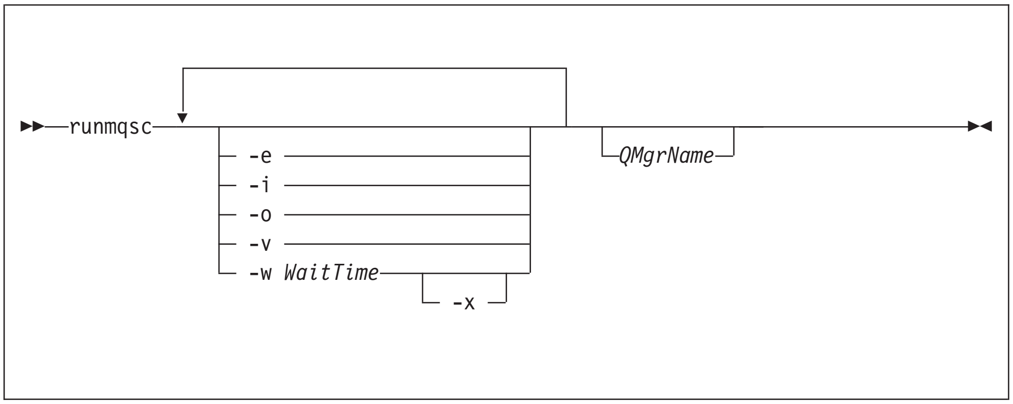
**20** An error occurred during processing

# runmqsc (Run MQSeries commands)

## Purpose

Use the **runmqsc** command to issue MQSC commands to a queue manager. MQSC commands enable you to perform administration tasks, for example defining, altering, or deleting a local queue object. MQSC commands and their syntax are described in the *MQSeries MQSC Command Reference*.

## Syntax



## Description

You can invoke the **runmqsc** command in three modes:

**Verify mode**
MQSC commands are verified but not actually run. An output report is generated indicating the success or failure of each command. This mode is only available on a local queue manager.

**Direct mode**
MQSC commands are sent directly to a local queue manager.

**Indirect mode**
MQSC commands are run on a remote queue manager. These commands are put on the command queue on a remote queue manager and are run in the order in which they were queued. Reports from the commands are returned to the local queue manager.

The **runmqsc** command takes its input from the standard IN file. When the commands are processed, the results and a summary are put into a report that is sent to the standard OUT file.

By taking the standard IN file from the keyboard, you can enter MQSC commands interactively.

By redirecting the input from a file you can run a sequence of frequently-used commands contained in the file. You can also redirect the output report to a file.

**Note:** To run this command, your user ID must belong to user group MQM.

## Optional parameters

**-e** Prevents source text for the MQSC commands from being copied into a report. This is useful when you enter commands interactively.

**–i** Input file name

**–o** Output file name

**-v** Specifies verification mode; this verifies the specified commands without performing the actions. This mode is available locally only. The -w and -x flags are ignored if they are specified at the same time.

**-w** *WaitTime*
Specifies indirect mode, that is, the MQSC commands are to be run on another queue manager. You must have the required channel and transmission queues set up for this.

*WaitTime*
Specifies the time, in seconds, that **runmqsc** waits for replies. Any replies received after this are discarded, however, the MQSC commands are still run. Specify a time between 1 and 999 999 seconds.

Each command is sent as an Escape PCF to the command queue (SYSTEM.ADMIN.COMMAND.QUEUE) of the target queue manager.

The replies are received on queue SYSTEM.MQSC.REPLY.QUEUE and the outcome is added to the report. This can be defined as either a local queue or a model queue.

Indirect mode operation is performed through the default queue manager.

This flag is ignored if the -v flag is specified.

**-x** Specifies that the target queue manager is running under . This flag applies only in indirect mode. The -w flag must also be specified. In indirect mode, the MQSC commands are written in a form suitable for the MQSeries for MVS/ESA command queue.

*QMgrName*
Specifies the name of the target queue manager on which the MQSC commands are to be run. If omitted, the MQSC commands run on the default queue manager.

## Return codes

**00** MQSC command file processed successfully.
**10** MQSC command file processed with errors-report contains reasons for failing commands.
**20** Error-MQSC command file not run.

## Examples

1. Enter this command at the TACL prompt:

```
runmqsc
```

Now you can enter MQSC commands directly. No queue manager name was specified, therefore the MQSC commands are processed on the default queue manager.

2. The following example shows how to specify that MQSC commands are verified only:

```
runmqsc -i $SYSTEM.CONFIG.MQSCIN  -v BANK
```

   This verifies the MQSC command file $SYSTEM.CONFIG.MQSCIN. The queue manager name is `BANK`. The output is displayed in the current window.

3. This command runs an MQSC command file against the queue manager called `BANK`:

```
runmqsc -i MQSCFILE -o $TEST.MQ.MQSCOUT BANK
```

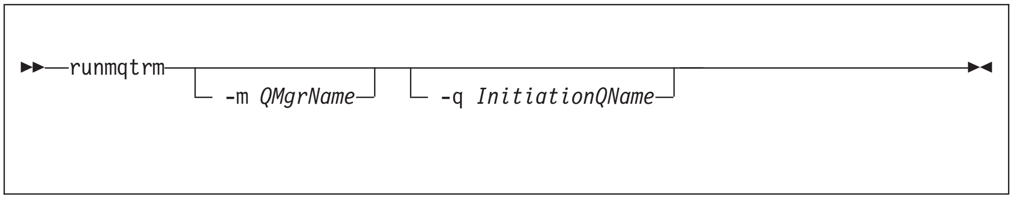
   In this example, the output is directed to file `$TEST.MQ.MQSCOUT`. The input file is `MQSCFILE` in the current subvolume.

# runmqtrm (Start trigger monitor)

## Purpose

Use the **runmqtrm** command to invoke a trigger monitor. For further information about using trigger monitors, refer to the *MQSeries Application Programming Guide*.

## Syntax



## Optional parameters

**-m** *QMgrName*
Specifies the name of the queue manager on which the trigger monitor operates. If this parameter is omitted, the trigger monitor operates on the default queue manager.

**-q** *InitiationQName*
Specifies the name of the initiation queue to be processed. If this parameter is omitted, SYSTEM.DEFAULT.INITIATION.QUEUE is used.

## Return codes

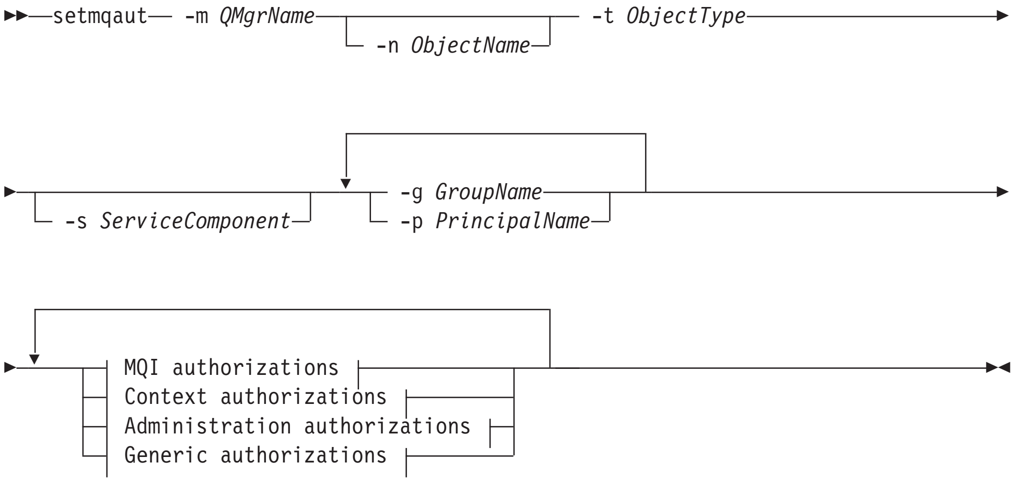
**10** Trigger monitor interrupted by an error.
**20** Error—trigger monitor not run.

# setmqaut (Set/reset authority)

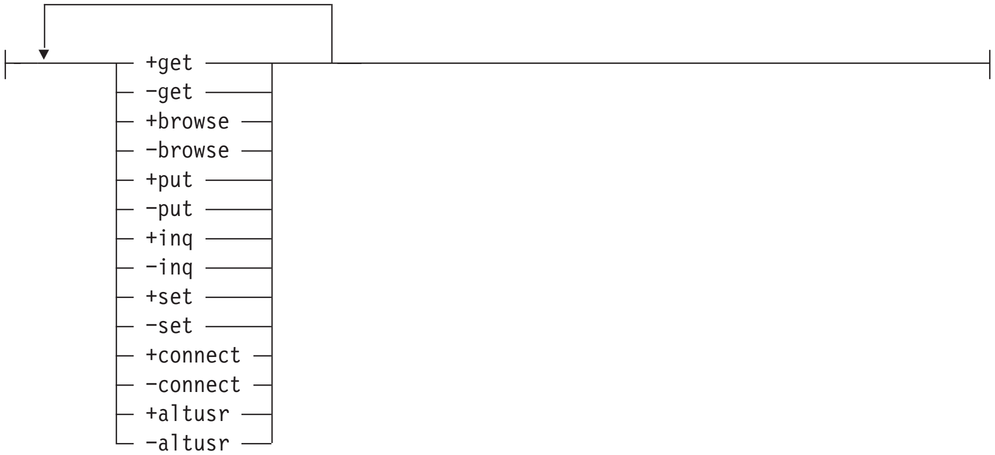
## Purpose

Use the **setmqaut** command to change the authorizations to an object or to a class of objects. Authorizations can be granted to, or revoked from, any number of principals or groups.

## Syntax



**MQI authorizations:**



**Context authorizations:**

```
    ┌───────────────────────┐
    ▼                       │
├─────────┬─ +passid ──┬────┴──────────────────────────────────────┤
          ├─ -passid ──┤
          ├─ +passall ─┤
          ├─ -passall ─┤
          ├─ +setid ───┤
          ├─ -setid ───┤
          ├─ +setall ──┤
          └─ -setall ──┘
```

**Administration authorizations:**

```
    ┌──────────────────┐
    ▼                  │
├─────────┬─ +crt ─┬───┴──────────────────────────────────────────┤
          ├─ -crt ─┤
          ├─ +dlt ─┤
          ├─ -dlt ─┤
          ├─ +chg ─┤
          ├─ -chg ─┤
          ├─ +dsp ─┤
          ├─ -dsp ─┤
          ├─ +clr ─┤
          └─ -clr ─┘
```

**Generic authorizations:**

```
    ┌──────────────────────┐
    ▼                      │
├─────────┬─ +allmqi ─┬────┴──────────────────────────────────────┤
          ├─ -allmqi ─┤
          ├─ +alladm ─┤
          ├─ -alladm ─┤
          ├─ +all ────┤
          └─ -all ────┘
```

## Description

You can use this command both to *set* an authorization, that is, give a user group permission to perform an operation, and to *reset* an authorization, that is, remove the permission to perform an operation. You must specify the user groups to which the authorizations apply and also the queue manager, object type, and object name of the object. You can specify any number of groups in a single command.

The authorizations that can be given are categorized as follows:

- Authorizations for issuing MQI calls
- Authorizations for MQI context
- Authorizations for issuing commands for administration tasks
- Generic authorizations

Each authorization to be changed is specified in an authorization list as part of the command. Each item in the list is a string prefixed by '+' or '–'. For example, if you include `+put` in the authorization list, you are giving authority to issue MQPUT calls against a queue. Alternatively, if you include `-put` in the authorization list, you are removing the authorization to issue MQPUT calls.

Authorizations can be specified in any order provided that they do not clash. For example, specifying `allmqi` with `set` causes a clash.

You can specify as many groups or authorizations as you require in a single command.

If a user ID is a member of more than one group, the authorizations that apply are the union of the authorizations of each group to which that user ID belongs.

## Required parameters

**-g** *GroupName*

Specifies the name of the user group whose authorizations are to be changed. You can specify more than one group name, but each name must be prefixed by the -g flag.

You must specify at least one principal or group.

**-m** *QMgrName*

Specifies the name of the queue manager of the object for which the authorizations are to be changed. The name can contain up to 48 characters.

**-p** *PrincipalName*

Specifies the name of the principal for which the authorizations are to be changed. You can specify more than one principal name, but each name must be prefixed by the -p flag.

You must specify at least one principal or group.

**-t** *ObjectType*

Specifies the type of object for which the authorizations are to be changed.

Possible values are:

- **q** or **queue**
- **prcs** or **process**
- **qmgr**
- **nl** or **namelist**

## Optional parameters

**-n** *ObjectName*

Specifies the name of the object for which the authorizations are to be changed.

This is a required parameter ***unless*** it is the queue manager itself. You must specify the name of a queue manager, queue, or process, but must not use a generic name.

**-s** *ServiceComponent*

This parameter applies only if you are using installable authorization services, otherwise it is ignored.

If installable authorization services are supported, this parameter specifies the name of the authorization service to which the authorizations apply. This parameter is optional; if it is not specified, the authorization update is made to the first installable component for the service.

*Authorizations*

Specifies the authorizations to be given or removed. Each item in the list is prefixed by a '+' indicating that authority is to be given, or a '–', indicating that authorization is to be removed. For example, to give authority to issue an MQPUT call from the MQI, specify `+put` in the list. To remove authority to issue an MQPUT call, specify `-put.`

Table 10 shows the authorities that can be given to the different object types.

*Table 10. Specifying authorizations for different object types*

| Authority | Queue | Process | Qmgr | Namelist |
|---|---|---|---|---|
| all | ✓ | ✓ | ✓ | ✓ |
| alladm | ✓ | ✓ | ✓ | ✓ |
| allmqi | ✓ | ✓ | ✓ | ✓ |
| altusr |  |  | ✓ |  |
| browse | ✓ |  |  |  |
| chg | ✓ | ✓ | ✓ | ✓ |
| clr | ✓ |  |  |  |
| connect |  |  | ✓ |  |
| crt | ✓ | ✓ | ✓ | ✓ |
| dlt | ✓ | ✓ | ✓ | ✓ |
| dsp | ✓ | ✓ | ✓ | ✓ |
| put | ✓ |  |  |  |
| inq | ✓ | ✓ | ✓ | ✓ |
| get | ✓ |  |  |  |
| passall | ✓ |  |  |  |
| passid | ✓ |  |  |  |
| set | ✓ | ✓ | ✓ |  |
| setall | ✓ |  | ✓ |  |
| setid | ✓ |  | ✓ |  |

### Authorizations for MQI calls

**altusr** Use an alternate user ID in a message.

See the *MQSeries Application Programming Guide* for more information about alternate user IDs.

**browse**
Retrieve a message from a queue by issuing an MQGET call with the BROWSE option.

**connect**
Connect the application to the specified queue manager by issuing an MQCONN call.

**get** Retrieve a message from a queue by issuing an MQGET call.

**inq** Make an inquiry on a specific queue by issuing an MQINQ call.

**put** Put a message on a specific queue by issuing an MQPUT call.

**set** Set attributes on a queue from the MQI by issuing an MQSET call.

**Note:** If you open a queue for multiple options, you have to be authorized for each of them.

**Authorizations for context**

**passall**
Pass all context on the specified queue. All the context fields are copied from the original request.

**passid** Pass identity context on the specified queue. The identity context is the same as that of the request.

**setall** Set all context on the specified queue. This is used by special system utilities.

**setid** Set identity context on the specified queue. This is used by special system utilities.

**Authorizations for commands**

**chg** Change the attributes of the specified object.

**clr** Clear the specified queue (PCF Clear queue command only).

**crt** Create objects of the specified type.

**dlt** Delete the specified object.

**dsp** Display the attributes of the specified object.

**Authorizations for generic operations**

**all** Use all operations applicable to the object.

**alladm**
Perform all administration operations applicable to the object.

**allmqi** Use all MQI calls applicable to the object.

## Return codes

| | |
|---|---|
| **0** | Successful operation |
| **36** | Invalid arguments supplied |
| **40** | Queue manager not available |
| **49** | Queue manager stopping |
| **69** | Storage not available |
| **71** | Unexpected error |
| **72** | Queue manager name error |
| **133** | Unknown object name |
| **145** | Unexpected object name |
| **146** | Object name missing |
| **147** | Object type missing |
| **148** | Invalid object type |
| **149** | Entity name missing |
| **150** | Authorization specification missing |
| **151** | Invalid authorization specification |

## Examples

1. This example shows a command that specifies that the object on which authorizations are being given is the queue `orange.queue` on queue manager `saturn.queue.manager`.

```
setmqaut -m saturn.queue.manager -n orange.queue -t queue -g tango +inq +alladm
```

The authorizations are being given to user group `tango` and the associated authorization list specifies that user group `tango`:

- Can issue MQINQ calls.
- Has authority to perform all administration operations on that object.

2. In this example, the authorization list specifies that user group `foxy`:
   - Cannot issue any calls from the MQI to the specified queue.
   - Has authority to perform all administration operations on the specified queue.

```
setmqaut -m saturn.queue.manager -n orange.queue -t queue -g foxy -allmqi +alladm
```

## Related commands

| | |
|---|---|
| **dspmqaut** | Display authority |

## strmqcsv (Start command server)

### Purpose

Use the **strmqcsv** command to start the command server for the specified queue manager. This enables MQSeries to process commands sent to the command queue.

### Syntax

```
►►──strmqcsv──QMgrName──────────────────────────────────────────────►◄
```

### Required parameters

*QMgrName*
Specifies the name of the queue manager for which the command server is to be started.

### Return codes

| | |
|---|---|
| **0** | Command completed normally |
| **10** | Command completed with unexpected results |
| **20** | An error occurred during processing |

### Examples

The following command starts a command server for queue manager `earth`:

```
strmqcsv earth
```

### Related commands

| | |
|---|---|
| **endmqcsv** | End a command server |
| **dspmqcsv** | Display the status of a command server |

# strmqm (Start queue manager)

## Purpose

Use the **strmqm** command to start a local queue manager. Only after the queue manager is available to process connections or other requests, will the **strmqm** command return to the command line.

## Syntax

```
►►──strmqm──┬────┬──┬────┬──┬──────────┬──────────────────────────►◄
            └ -c ┘  └ -z ┘  └─QMgrName─┘
```

## Optional parameters

**-c** Starts the queue manager, redefines the default and system objects, then stops the queue manager. (The default and system objects for a queue manager are created initially by the crtmqm command.) Any existing system and default objects belonging to the queue manager are replaced if you specify this flag.

*QMgrName*
Specifies the name of a local queue manager to be started. If omitted, the default queue manager is started.

**-z** Suppresses error messages.

This flag is used within MQSeries to suppress unwanted error messages. Because using this flag could result in loss of information, you should not use it when entering commands on a command line.

## Return codes

| | |
|---|---|
| **0** | Queue manager started |
| **3** | Queue manager being created |
| **5** | Queue manager running |
| **16** | Queue manager does not exist |
| **49** | Queue manager stopping |
| **69** | Storage not available |
| **71** | Unexpected error |
| **72** | Queue manager name error |

## Examples

The following command starts the queue manager `account`:

```
strmqm account
```

## Related commands

| | |
|---|---|
| **crtmqm** | Create a queue manager |
| **dltmqm** | Delete a queue manager |
| **endmqm** | End a queue manager |

# strmqtrc (Start MQSeries trace)

## Purpose

Use the **strmqtrc** command to enable tracing. This command can be run whether tracing is enabled or not. If tracing is already enabled, the trace options in effect are modified to those specified on the latest invocation of the command.

For more information about using MQSeries trace, see "Using MQSeries trace" on page 198.

## Syntax

```
►►──strmqtrc──┬──────────────┬──┬────┬──┬──────────────┬──────────────────────────►
              └─-m QMgrName──┘  └─-e─┘  └─-l MaxSize───┘

    ┌──────────────────┐
    ▼                  │
►──┬───────────────┬───┴─────────────────────────────────────────────────────────►◄
   └─-t TraceType──┘
```

## Optional parameters

**-m** *QMgrName*
Is the name of the queue manager to be traced. If no name is specified, the default queue manager is used.

The specified queue manager does not have to be running or even to exist. Consequently, it is possible to trace the creation or startup of a queue manager.

A queue manager name can be specified on the same command as the -e flag. If more than one trace specification applies to a given entity being traced, the trace includes all of the specified options.

**-e** If this flag is specified, early tracing is requested. This involves trace information being written, before the processes know to which MQSeries component they belong. Any process, belonging to any component of any queue manager, traces its early processing if this flag is specified. The default, if this flag is not specified, is not to perform early tracing.

**-l** *MaxSize*
The value of *MaxSize* denotes the maximum size of a trace file (AMQnnnn.TRC) in millions of bytes. For example, if you specify a MaxSize of 1, the size of the trace is limited to 1 million bytes.

When a trace file reaches the specified maximum, it is renamed from AQnnnn.TRC to AMQnnnn.TRS and a new AMQnnnn.TRC file is started. All trace files are restarted when the maximum limit is reached. If a previous copy of an AMQnnnn.TRS file exists, it will be deleted.

**-t** *TraceType*
Defines which points during processing can be traced. One or more of the following options can be supplied:

**all** Output data for every trace point in the system. This is also the default if the -t flag is not specified.

**api** Output data for trace points associated with the MQI and major queue manager components.

**comms** Output data for trace points associated with data flowing over communications networks.

**csflows** Output data for trace points associated with processing flow in common services.

**lqmflows** Output data for trace points associated with processing flow in the local queue manager.

**remoteflows** Output data for trace points associated with processing flow in the communications component.

**otherflows** Output data for trace points associated with processing flow in other components.

**csdata** Output data for trace points associated with internal data buffers in common services.

**lqmdata** Output data for trace points associated with internal data buffers in the local queue manager.

**remotedata** Output data for trace points associated with internal data buffers in the communications component.

**otherdata** Output data for trace points associated with internal data buffers in other components.

**versiondata** Output data for trace points associated with the version of MQSeries running.

**commentary** Output data for trace points associated with comments in the MQSeries components.

If this flag is omitted, all trace points are enabled and a full trace generated.

**Note:** If multiple trace types are supplied, each ***must*** have its own -t flag. Any number of -t flags can be specified, provided that each has a valid trace type associated with it.

It is not an error to specify the same trace type on multiple -t flags.

## Return codes

**AMQ7024**
This message is issued if arguments that are not valid are supplied to the command.

**AMQ8304**
The maximum number of nine concurrent traces is already running.

## Examples

This command enables tracing of data from common services and the local queue manager, for a queue manager called QM1.

```
strmqtrc -m QM1 -t csdata -t lqmdata
```

## Related commands

| | |
|---|---|
| **dspmqtrc** | Display formatted trace output |
| **endmqtrc** | End MQSeries trace |

# upgmqm (Upgrade V2.2.0.1 queue manager)

## Purpose

This command upgrades a Version 2.2.0.1 queue manager for use with MQSeries for Compaq NSK V5.1. The utility invoked by **upgmqm** sends progress messages to the terminal from which it is invoked. When the utility completes, the named queue manager is ready for use with MQSeries for Compaq NSK V5.1. Queue manager attributes new in Version 5.1 are set to their default values. You can alter these in the usual way.

Because the functionality of the new Version 5.1 status server is different from the MQSS Server in Version 2.2.0.1, this upgrade deletes all existing MQS-Status*nn* server classes from your existing PATHWAY configuration and replaces them with one default MQS-Status00 server class and one default MQS-Queue00 server class. This means that after you have upgraded your queue manager, any objects that are assigned to your present MQSS server processes will need to be re-assigned to your new status server processes or queue server processes, depending on which is appropriate. (In Version 5.1, only local queues are assigned to the queue servers and all other objects are assigned to the status servers.) You may also need to change any existing scripts that refer to your status server classes in PATHWAY.

If you elected not to clean up during the upgrade, you can delete the following files at your convenience:

*Table 11. Examples of files that can be deleted after an upgrade*

| Location | Files | Example |
|---|---|---|
| Subvolume indicated by -p option on **upgmqm** | All files | $VOL.scratch |
| Queue managers 'S' subvolume | Z* | $VOL.myv2201S |

These examples assume that your existing queue manager resides on $VOL.myv2201? subvolumes.

**Note:** If a Version 2.2.0.1 queue manager is not upgraded using **upgmqm**, no control commands will work for that queue manager. This includes **dltmqm**, which will fail with trying to open the principal database. A queue manager from Version 2.2.0.1 no longer needed under Version 5.1 must be removed prior to the code upgrade, or upgraded using **upgmqm**, then removed.

## Syntax

```
►►──upgmqm──-m QMgrName──-v DefaultQueueServer──-p SubvolumePath──────────────►

►─-s DefaultStatusServer──────────────────────────────────────────────────────►◄
```

## Required parameters

**-m** *QMgrName*
Specifies the name of the queue manager to which the **upgmqm** utility is to be applied.

**-v** *DefaultQueueServer*
A unique process name for the default queue server for the queue manager.

**-p** *SubvolumePath*
A subvolume path ($VOL.SUBVOL) that the upgrade utility can use for working files. This subvolume **must** be on the same volume as the queue manager. Specify only the subvolume part of the path; do not specify the volume name. The **upgmqm** command checks that the subvolume does not already exist, before accepting the subvolume as valid.

**-s** *DefaultStatusServer*
A unique process name for the default status server for the queue manager.

## Examples

This example upgrades a Version 2.2.0.1 queue manager `Myv2201qm` with a default queue server name of `$MYQS`, a default status server name of `$MYSS`, and uses subvolume `$VOL.scratch` for the working files (where `$VOL` is the volume on which the queue manager resides):

```
upgmqm -m Myv2201qm -p scratch -s $MYSS -v $MYQS
```

# Part 3. Appendixes

© Copyright IBM Corp. 1993, 2001

# Appendix A. MQSeries for Compaq NSK at a glance

## Program and part number

- 5724-A39 MQSeries for Compaq NSK, Version 5 Release 1, part number 0791003

## Hardware requirements

Minimum hardware requirements are:

- Any of the Compaq NSK range of machines supported by Guardian D45 or later D4x, G06 or later G0x.
- Specific hardware in support of user-selected network transport protocols.

You are also recommended to have one or more mirrored data disks with specified space requirements for TMF audit space and the MQSeries database.

## Software requirements

Minimum software requirements are:

- Compaq NSK Guardian D45 or later D4x (K-series hardware) and G06 or later G0x (S-series hardware) operating systems, including TM/MP (TMF), ENSCRIBE, and EMS.
- TS/MP (PATHWAY) to match operating system.
- SCF for configuration, command, and control of TCP and SNA network transports.

For SNA connectivity:

- SNAX/APC and SNAX/XF or SNAX/APN to match operating system

or

- Insession ICE Version 3.2 or later

For TCP/IP connectivity:

- TCP/IP to match operating system.

To use the OSS-based parts of MQSeries (MQI bindings, OSS applications, Java bindings, you require the OSS product version compatible with the operating system.

Transaction logging is maintained with the Compaq TM/MP (TMF) product.

## Security

MQSeries for Compaq NSK uses the security features of the NSK file system, which provide file-level access control to USER and GROUP for read, write, execute, and purge operations. SAFEGUARD is not required for the use of MQSeries for Compaq NSK; however, the product is compatible with a SAFEGUARD environment.

© Copyright IBM Corp. 1993, 2001

### Security

All MQSeries resources are owned by a single user ID in group MQM. To administer MQSeries with either the SCOBOL menus or **runmqsc**, you must be logged in with a user ID assigned or linked to the MQM group.

## Maintenance functions

MQSeries functions with:

- The Message Queue Management (MQM) facility using SCOBOL requester configuration screens in a PATHWAY environment.
- The **runmqsc** command-line interface.
- SCF utility for configuration, command and control functionality to maintain TCP/IP and SNA environments for Compaq network protocol offerings.
- ICE utilities provided with that product for control of ICE LU 6.2 interface.
- MQSeries Explorer (not included with MQSeries for Compaq NSK).
- Any other product or utility that uses standard PCF commands for remote administration.

## Compatibility

The MQI for MQSeries for Compaq NSK V5.1, is compatible with existing applications running on MQSeries for Tandem NonStop Kernel V2.2.0.1, with maintenance fix PTF U473441.

### Supported compilers

MQSeries for Compaq NSK V5.1 is built using the Common Runtime Environment (CRE) to link all objects. This method imposes the following requirements on users of versions of the MQI prior to Version 2.2.0.1:

1. All pre-D45 COBOL and C object code must be recompiled with the D45 (or later) compiler to integrate the CRE linkage.
2. All pre-D45 TAL object code must be recompiled with a D45 (or later) compiler and you must ensure that the TAL program is compliant with the special programming considerations specified in the *Common Run-time Environment Programmer's Guide*. More detailed information on each of these programming considerations is provided in the *TAL Programmer's Guide*.
3. For object code produced with native compilers on D45, a separate binding is provided.
4. C programs must use the WIDE memory model (32–bit integers).
5. COBOL programs must conform to the requirements of the CRE.
6. In TAL programs, all integers passed to the MQI functions must be 32 bits (or be cast to 32 bit with the $INT32() macro).

The MQSeries programs themselves are compiled and linked using the native mode tools for Guardian NSK. Native mode applications normally link with the queue manager SRL directly unless the application already uses a Private SRL. In this case, since applications are restricted to using at most a single Private SRL, the application must either link with the static MQI binding library, or the code that resides in the application's private SRL must be combined with the MQSeries SRL into a new Private SRL.

## License management

You must enter the system type to define the program entitlement. This parameter can be entered at installation time or at any subsequent time in the event of a license upgrade being purchased. At startup this value is checked against the physical Compaq machine configuration. If the license registration and program entitlement are insufficient, a warning message is issued.

## Language selection

A supplied message text file is encoded in the 7-bit character set that is native to the Compaq NSK operating system. MQSeries for Compaq NSK lets the national language be specified when the product is installed. The message language defaults to U.S. English.

## Internationalization

MQSeries for Compaq NSK lets the CCSID be specified when the queue manager is created (Although the CCSID call also be changed after the queue manager is created.) The queue manager CCSID defaults to 819. MQSeries for Compaq NSK supports character-set conversion into the configured CCSID of the queue manager. For information about the CCSIDs that can be specified for an MQSeries for Compaq NSK queue manager, including those that provide support for the euro character, see the *MQSeries Application Programming Reference*.

# Appendix B. System defaults

When you create a queue manager using the **crtmqm** control command, the system objects and default objects are created automatically.

- The system objects are those MQSeries objects required for the operation of a queue manager or channel.
- The default objects define all of the attributes of an object. When you create an object, such as a local queue, any attributes that you do not specify explicitly are inherited from the default object.

*Table 12. System and default objects for queues*

| **Object Name** | **Description** |
|---|---|
| SYSTEM.DEFAULT.ALIAS.QUEUE | Default alias queue |
| SYSTEM.DEFAULT.LOCAL.QUEUE | Default local queue |
| SYSTEM.DEFAULT.MODEL.QUEUE | Default model queue. |
| SYSTEM.DEFAULT.REMOTE.QUEUE | Default remote queue. |
| SYSTEM.DEAD.LETTER.QUEUE | Sample dead-letter (undelivered-message) queue |
| SYSTEM.DEFAULT.INITIATION.QUEUE | Default initiation queue |
| SYSTEM.CICS.INITIATION.QUEUE | Default CICS® initiation queue |
| SYSTEM.ADMIN.COMMAND.QUEUE | Administration command queue. Used for remote MQSC commands, and PCF commands. |
| SYSTEM.MQSC.REPLY.QUEUE | MQSC reply-to-queue. This a model queue that creates a temporary dynamic queue for replies to remote MQSC commands. |
| SYSTEM.ADMIN.QMGR.EVENT | Event queue for queue manager events. |
| SYSTEM.ADMIN.PERFM.EVENT | Event queue for performance events. |
| SYSTEM.ADMIN.CHANNEL.EVENT | Event queue for channel events. |
| SYSTEM.CHANNEL.INITQ | Channel initiation queue. |
| SYSTEM.CHANNEL.SYNCQ | The queue which holds the synchronization data for channels. (This object is created but not used In MQSeries for Compaq NSK. Channel Syncq information is stored in TM/MP protected databases.) |
| SYSTEM.CLUSTER.COMMAND.QUEUE | The queue used to carry messages to the repository queue manager. |
| SYSTEM.CLUSTER.REPOSITORY.QUEUE | The queue used to store all repository information. |
| SYSTEM.CLUSTER.TRANSMIT.QUEUE | The transmission queue for all messages to clusters. |

*Table 13. System and default objects for channels*

| **Object Name** | **Description** |
|---|---|
| SYSTEM.DEF.SENDER | Default sender channel |
| SYSTEM.DEF.SERVER | Default server channel |

© Copyright IBM Corp. 1993, 2001

*Table 13. System and default objects for channels (continued)*

| Object Name | Description |
|---|---|
| SYSTEM.DEF.RECEIVER | Default receiver channel |
| SYSTEM.DEF.REQUESTER | Default requester channel |
| SYSTEM.DEF.SVRCONN | Default server connection channel |
| SYSTEM.DEF.CLNTCONN | Default client connection channel |
| SYSTEM.AUTO.RECEIVER | Dynamic reciever channel |
| SYSTEM.AUTO.SVRCONN | Dynamic server-connection channel |
| SYSTEM.DEF.CLUSRCVR | Default receiver channel for the cluster used to supply default values for any attributes not specified when a CLUSRCVR channel is created on a queue manager in a cluster. |
| SYSTEM.DEF.CLUSSDR | Default sender channel for the cluster used to supply default values for any attributes not specified when CLUSSDR channel is created on a queue manager in the cluster. |

*Table 14. System and default objects for namelists*

| Object Name | Description |
|---|---|
| SYSTEM.DEFAULT.NAMELIST | Default namelist |

*Table 15. System and default objects for processes*

| Object Name | Description |
|---|---|
| SYSTEM.DEFAULT.PROCESS | Default process definition |

# Appendix C. Setting TACL environment variables for MQSeries for Compaq NSK

MQSeries creates and uses a number of Compaq NSK environment variables, or PARAMs. When setting these PARAMs, consider the following points:

- The MQDEFAULTPREFIX PARAM must be present in the environment of all programs. The TS/MP (Pathway) configuration established automatically by the **crtmqm** command ensures that these PARAMs are set correctly for any queue manager server processes. Users of MQSeries applications and control commands must ensure that the TACLs and TS/MP configurations used also specify these variables.
- You are recommended to include the PARAM statements in your TACLCSTM files so that, when you log on, these PARAMs are created correctly, and any programs run from the TACL inherit the correct values. The following environment variable should also be modified to allow location of MQSeries executables:

```
#SET #PMSEARCH $SYSTEM.ZMQSEXE [#PMSEARCH]
```

**MQCONNECTTYPE**

This PARAM, if present, can be used to disable the ability of applications to use FASTPATH connections. If this PARAM is set to the value STANDARD, applications are only able to use STANDARD connections, even if they request FASTPATH. Any other value is treated as if the PARAM was not specified (and therefore applications, if they request it, are able to use FASTPATH connections).

**MQDEFAULTPREFIX**

The name of the volume containing the installed subvolume, ZMQSSYS. This PARAM must be correctly defined in all environments.

For example:

```
    PARAM MQDEFAULTPREFIX $data00
```

**MQEMSEVENTS**

This PARAM enables MQSeries EMS events. For example, to switch on all EMS events for MQSeries, you set the PARAM MQEMSEVENTS as follows:

```
    PARAM MQEMSEVENTS 127
```

**MQMACHINIFILE**

The location of the MQSINI file for the installation. The default value is *MQDEFAULTPREFIX*.ZMQSSYS.MQSINI. This PARAM is required only if a nondefault location is required.

For example:

© Copyright IBM Corp. 1993, 2001

```
PARAM MQMACHINIFILE $data00.altinst.mqsini
```

**MQRDF**

If this PARAM is set ON, MQSeries changes the behavior of the delete operation to work with RDF for audited files. If this PARAM is not defined, or is set to anything other than ON, the MQSeries delete operation functions normally. If used, this PARAM must be set in the TACL environment of any user that runs administrative programs, and in the environment of all TS/MP server classes configured in the queue manager.

**MQRDFFUPPROCESSNAME**

This PARAM is interpreted only by the **cleanrdf** utility. It is used to specify a Guardian process name that will be assigned to the FUP server process that the **cleanrdf** utility creates. If this PARAM is not defined, the FUP server process name is assigned by the operating system.

**MQRDFFUPPROGNAME**

This PARAM is interpreted only by the **cleanrdf** utility. It is used to specify the fully qualified name of the FUP executable file to be used by the utility. The default value is `<default system name>.$SYSTEM.SYS00.FUP`.

**MQSNOAUT**

If this PARAM is set to 1 when **crtmqm** is run, the new queue manager is created with the OAM disabled.

For example:

```
PARAM MQSNOAUT 1
```

**MQLISTENPORTNUM**

If this PARAM is set then the TCP/IP Listener process uses it to find out which port to listen on. If the PARAM is not specified, the port is determined from the QMINI file TCP/IP stanza.

**SAVE-ENVIRONMENT ON**

Required when running application programs to ensure the Common Run-Time Environment (CRE) passes PARAMs from the environment to the application program.

For example:

```
PARAM SAVE-ENVIRONMENT ON
```

If this PARAM is not set, applications receive return code 2058, indicating a queue manager name error.

This PARAM is required for TAL or COBOL applications only, running as non-native programs.

## Queue server tuning parameters

The following PARAMS, if defined in the TACL environment of a queue server, can be used to override the built-in defaults of the queue server, for various housekeeping operations:

**MQQSHKEEPINT**
If this PARAM is set, a numeric value in seconds may be specified to override the default houskeeping interval (60s) of the queue server. The houskeeping interval controls the frequency at which the queue server looks at queues to detect expired messages, and examines its memory utilization in order to optimize operations.

**MQQSSIGTIMEOUT**
If this PARAM is set, a numeric value is seconds my be specified to override the default timeout (60s) for the delivery of a signal IPC to an application that has initiated an MQGET with the MQGMO_SET_SIGNAL option. If a queue server is unable to deliver the signal within this timeout (once the conditions for generating the signal have been met) the queue server logs the fact and then cancels the signal.

**MQQSMAXBATCHEXPIRE**
If this PARAM is set, a numeric value may be specified to override the default maximum number of expired Persistent Messages (100) that will be discarded within a single transaction during housekeeping by a queue server. When Persistent Messages expire, they must be physically removed from the queue databases, which requires an internal TM/MP transaction. This PARAM allows control over the maximum number of messages that will be removed within a single TM/MP transaction.

**MQQSMAXMSGSEXPIRE**
If this PARAM is set, a numeric value may be specified to override the default maximum number of expired messages (300) that will be detected and discarded within a single housekeeping instance of a queue server.

## TACL environment variables

# Appendix D. Comparing command sets

Table 16 through Table 21 on page 305 compare the facilities available from the different administration command sets.

**Note:** Only those MQSC commands that apply to MQSeries for Compaq NSK are shown.

*Table 16. Commands for queue manager administration*

| PCF | MQSC | Control |
|---|---|---|
| Change Queue Manager | ALTER QMGR | – |
| (Create queue manager)* | – | crtmqm |
| (Delete queue manager)* | – | dltmqm |
| Inquire Queue Manager | DISPLAY QMGR | – |
| (Stop queue manager)* | – | endmqm |
| Ping Queue Manager | PING QMGR | – |
| (Start queue manager)* | – | strmqm |
| **Note:** * Not available as PCF commands. | | |

*Table 17. Commands for command server administration*

| Description | Control |
|---|---|
| Display command server | dspmqcsv |
| Stop command server | endmqcsv |
| Start command server | strmqcsv |
| **Note:** As an alternative to the control commands, you may use PATHCOM commands, as described in “TS/MP (PATHWAY) administration” on page 29. There are no MQSC or PCF equivalents of commands in this group. | |

© Copyright IBM Corp. 1993, 2001

## Comparing command sets

*Table 18. Commands for queue administration*

| PCF | MQSC |
|---|---|
| Change Queue | ALTER QLOCAL<br>ALTER QALIAS<br>ALTER QMODEL<br>ALTER QREMOTE |
| Clear Queue | CLEAR QLOCAL |
| Copy Queue | DEFINE QLOCAL(x) LIKE(y)<br>DEFINE QALIAS(x) LIKE(y)<br>DEFINE QMODEL(x) LIKE(y)<br>DEFINE QREMOTE(x) LIKE(y) |
| Create Queue | DEFINE QLOCAL<br>DEFINE QALIAS<br>DEFINE QMODEL<br>DEFINE QREMOTE |
| Delete Queue | DELETE QLOCAL<br>DELETE QALIAS<br>DELETE QMODEL<br>DELETE QREMOTE |
| Inquire Queue | DISPLAY QUEUE |
| Inquire Queue Names | DISPLAY QUEUE |
| **Note:** There are no control commands for these functions. | |

*Table 19. Commands for process administration*

| PCF | MQSC |
|---|---|
| Change Process | ALTER PROCESS |
| Copy Process | DEFINE PROCESS(x) LIKE(y) |
| Create Process | DEFINE PROCESS |
| Delete Process | DELETE PROCESS |
| Inquire Process | DISPLAY PROCESS |
| Inquire Process Names | DISPLAY PROCESS |
| **Note:** There are no control commands for these functions. | |

*Table 20. Commands for channel administration*

| PCF | MQSC | Control |
|---|---|---|
| Change Channel | ALTER CHANNEL | – |
| Copy Channel | DEFINE CHANNEL(x) LIKE(y) | – |
| Create Channel | DEFINE CHANNEL | – |
| Delete Channel | DELETE CHANNEL | – |
| Inquire Channel | DISPLAY CHANNEL | – |
| Inquire Channel Names | DISPLAY CHANNEL | – |
| Inquire Channel Status | DISPLAY CHSTATUS | – |
| Ping Channel | PING CHANNEL | – |
| Reset Channel | RESET CHANNEL | – |
| Resolve Channel | RESOLVE CHANNEL | – |
| Start Channel | START CHANNEL | runmqchl |
| Start Channel Initiator | – | runmqchi |
| Start Channel Listener | – | runmqlsr |
| Stop Channel | STOP CHANNEL | – |
| **Note:** In MQSeries for Compaq NSK, use TS/MP or the control command **runmqlsr** to start TCP/IP channel listeners. For more information, see “Specifying and controlling TCP/IP listeners” on page 30 and “runmqlsr (Run listener)” on page 271. | | |

*Table 21. Other control commands*

| Description | Control |
|---|---|
| Alter queue volume, queue server message storage options | altmqfls |
| Add, delete, or alter MQSeries principals | altmqusr |
| RDF housekeeping utility | cleanrdf |
| Convert client channel definition table | cvclchl |
| Create MQSeries conversion exit | crtmqcvx |
| Display authority | dspmqaut |
| Display files used by objects; queue server message storage options configured for an object | dspmqfls |
| Display MQSeries formatted trace output | dspmqtrc |
| Display MQSeries principals | dspmqusr |
| End MQSeries trace | endmqtrc |
| Install MQSeries for Compaq NSK | instmqm |
| Run dead-letter queue handler | runmqdlq |
| Run MQSC commands | runmqsc |
| Run trigger monitor | runmqtrm |
| Set or reset authority | setmqaut |
| Start MQSeries trace | strmqtrc |
| Upgrade V2201 queue manager | upgmqm |

*Table 21. Other control commands (continued)*

| Description | Control |
|---|---|
| **Note:** As an alternative to the control command **runmqtrm**, you may use PATHCOM commands, as described in “TS/MP (PATHWAY) administration” on page 29. There are no MQSC or PCF equivalents of commands in this group. | |

# Appendix E. Stopping and removing queue managers manually

If the normal methods for stopping and removing queue managers fail, you can resort to the more drastic methods described here.

## Stopping a queue manager manually

The normal method of stopping queue managers, using the **endmqm** command, should work even in the event of failures within the queue manager. In exceptional circumstances, if this method of stopping a queue manager fails, use the following procedure to stop it manually:

1. Find the process IDs of the queue manager programs that are still running.
2. FUP LISTOPENS on the TRACEOPT file in the queue manager's data subvolume gives CPU, PIN of processes belonging to the queue manager.
3. End the queue manager processes that are still running. Use the **STOP** command, together with the process IDs discovered in the previous step.

   End the processes in the following order:

   a. `MQECBOSS` – EC Boss
   b. `MQEC` – ECs
   c. Any other processes that are still running

**Note:** Manual ending of the queue manager may result in FFSTs being taken, and the production of FD files. This should not be regarded as a defect in the queue manager.

The queue manager should restart normally, even if it was ended by using the preceding method.

If you want to delete the queue manager after stopping it manually, use the **dltmqm** command as normal. If, for some reason, this command fails to delete the queue manager, the manual process detailed in "Removing queue managers manually" can be used.

## Removing queue managers manually

To remove queue managers manually:

1. Ensure that there are no queue manager processes running for the queue manager you want to remove.
2. Edit the MQSINI file to remove the queue manager stanza and if necessary, modify the default queue manager stanza. Note the location of the queue manager files before deleting the stanza.
3. Delete all files in all subvolumes of the queue manager using the FUP PURGE command. For example, FUP PURGE $VOL.QMSVOL*.*.

© Copyright IBM Corp. 1993, 2001

## Removing queue managers manually

# Appendix F. MQSeries and Compaq NonStop Server for Java

MQSeries for Compaq NSK is compatible with the Compaq NonStop Server for Java, Version 1.5 and later. The product supports the full set of MQSeries Java classes. The *MQSeries Using Java* book describes these in detail.

MQSeries can operate in conjunction with NonStop Server for Java in two ways:

- As a Servlet running in the context of the Compaq iTP Webserver
- As a Java application running directly from the command line

The *Compaq NonStop Server for Java 1.5 Reference* gives information on using Servlets.

To access MQSeries from Java, either using Servlets or applications, it is necessary to create a custom version of the NonStop JVM that links in the following MQSeries product libraries:

- The Product SRL MQSRLLIB, from the Guardian ZMQSEXE subvolume
- Java binding archive libMQSESSION.a from /opt/mqm/lib

The Compaq Java documentation provides information on how to relink the JVM. The MQSeries samples directory (`/opt/mqm/samp`) contains two sample make files:

**MakeJVM.smp**
This is a sample make file illustrating how to rebuild the NonStop JVM to provide access to MQSeries. To use this make file, modify it to reflect the installed location of your MQSeries product libraries.

**MakeJava.smp**
Sample make file to build a Java application.

## Transactional considerations

The Java language does not provide direct access to the TM/MP interface. MQSeries supports transactions in Java via the JTS *Current Class*. The table below describes the *Current* interface.

*Table 22. Java language interface*

| Function | Description |
|---|---|
| begin() | Start a new transaction and associate it with the calling thread. |
| commit(boolean) | Commit the transaction associated with the calling thread. |
| get_control() | Obtain a Control object representing the transaction associated with the calling thread. |
| get_status() | Obtain the status of the transaction associated with the calling thread. |
| get_transaction_name() | Obtain a descriptive name of the transaction associated with the calling thread. |
| resume(ControlRef) | Set or resume association of a transaction with the calling thread. |

© Copyright IBM Corp. 1993, 2001

*Table 22. Java language interface (continued)*

| Function | Description |
|---|---|
| rollback() | Roll back the transaction associated with the calling thread. |
| suspend() | Suspend the association of the calling thread with a transaction context. |

For more information on JTS, refer to the Compaq NSK NonStop Java documentation or the JDK documentation.

# Appendix G. MQSC supported by MQSeries for Compaq NSK

This appendix lists the MQSeries commands (MQSC) supported by MQSeries for Compaq NSK. For information about the syntax of these commands, see the *MQSeries MQSC Command Reference*.

*Table 23. MQSC supported by MQSeries for Compaq NSK*

| **Command** | **Description** | **As described in** *MQSeries Command Reference* |
|---|---|---|
| ALTER CHANNEL | Change channel attributes. | Yes, but with the exceptions described in "CONNAME" on page 314. |
| ALTER NAMELIST | Alter a list of names. | Yes |
| ALTER PROCESS | Change process attributes. | Yes |
| ALTER QALIAS | Change attributes of an alias queue. | Yes |
| ALTER QLOCAL | Change attributes of a local queue. | Yes, but with the exceptions described in "HARDENBO and NOHARDENBO" on page 313. |
| ALTER QMGR | Change queue manager attributes. | Yes, but with the exceptions described in "MAXUMSGS and MAXHANDS" on page 313. |
| ALTER QMODEL | Change attributes of a model queue. | Yes, but with the exceptions described in "HARDENBO and NOHARDENBO" on page 313. |
| ALTER QREMOTE | Change attributes of a local definition of a remote queue, a queue-manager alias, or a reply-to queue alias. | Yes |
| CLEAR QLOCAL | Clear messages from a local queue. | Yes |
| DEFINE CHANNEL | Create a channel definition. | Yes, but with the exceptions described in "CONNAME" on page 314. |
| DEFINE NAMELIST | Define a list of names. | Yes |
| DEFINE PROCESS | Create a process definition. | Yes |
| DEFINE QALIAS | Create an alias-queue definition. | Yes |
| DEFINE QLOCAL | Create a local-queue definition. | Yes, but with the exceptions described in "HARDENBO and NOHARDENBO" on page 313. |
| DEFINE QMODEL | Create a model-queue definition. | Yes, but with the exceptions described in "HARDENBO and NOHARDENBO" on page 313. |

© Copyright IBM Corp. 1993, 2001

Table 23. MQSC supported by MQSeries for Compaq NSK (continued)

| **Command** | **Description** | **As described in** *MQSeries Command Reference* |
|---|---|---|
| DEFINE QREMOTE | Create a local definition of a remote queue, a queue-manager alias, or a reply-to-queue alias. | Yes |
| DELETE CHANNEL | Delete a channel definition. | Yes |
| DELETE NAMELIST | Delete a list of names. | Yes |
| DELETE PROCESS | Delete a process definition. | Yes |
| DELETE QALIAS | Delete an alias-queue definition. | Yes |
| DELETE QLOCAL | Delete a local-queue definition. | Yes |
| DELETE QMODEL | Delete a model-queue definition. | Yes |
| DELETE QREMOTE | Delete a local definition of a remote queue. | Yes |
| DISPLAY CHANNEL | Display a channel definition. | Yes, but with the exceptions described in “CONNAME” on page 314. |
| DISPLAY CHSTATUS | Display the status of one or more channels. | Yes, but with the exceptions described in “Channel Status information (DISPLAY CHSTATUS)” on page 313 and “CONNAME” on page 314. |
| DISPLAY CLUSQMGR | Display the status of one or more channels. | Yes, but with the exceptions described in “CONNAME” on page 314. |
| DISPLAY NAMELIST | Display a list of names. | Yes |
| DISPLAY PROCESS | Display a process definition. | Yes |
| DISPLAY QMGR | Display queue-manager attributes. | Yes, but with the exceptions described in “MAXUMSGS and MAXHANDS” on page 313. |
| DISPLAY QUEUE | Display queue attributes. | Yes, but with the exceptions described in “HARDENBO and NOHARDENBO” on page 313. |
| PING CHANNEL | Test a channel. | Yes |
| PING QMGR | Test whether queue manager is responding to commands. | Yes |
| REFRESH CLUSTER | Discard all locally held cluster information and force it to be rebuilt. | Yes |
| RESET CHANNEL | Reset the message sequence number for a channel. | Yes |
| RESET CLUSTER | Perform special operations on clusters. | Yes |

Table 23. MQSC supported by MQSeries for Compaq NSK (continued)

| Command | Description | As described in *MQSeries Command Reference* |
|---|---|---|
| RESOLVE CHANNEL | Resolve in-doubt messages on sender or server channel. | Yes |
| RESUME QMGR | Inform other queue managers in a cluster that the local queue manager is available again for processig and can be sent messages. | Yes |
| START CHANNEL | Start a channel. | Yes |
| STOP CHANNEL | Stop a channel. | Yes |
| SUSPEND QMGR | Inform other queue managers in a cluster that the local queue manager is not available for processing and cannot be sent messages. | Yes |

If you build MQSC commands into a script, there must be no more than 72 characters on each line.

## Attributes of MQSC

This section provides information about MQSC attributes that is specific to MQSeries for Compaq NSK.

### Channel Status information (DISPLAY CHSTATUS)

The DISPLAY CHSTATUS command is implemented as described in *MQSeries MQSC Command Reference* except that channel status is updated only at the boundaries of batch processing. Channel status information is not updated for every message transfer because of the potential impact on the performance of channels. This means the common status data values are identical for both the current and saved sets.

### MAXUMSGS and MAXHANDS

The queue manager object attributes MAXUMSGS and MAXHANDS are ignored. This affects the following commands:
ALTER QMGR
DISPLAY QMGR

### HARDENBO and NOHARDENBO

In MQSeries for Compaq NSK, the local and model queue attributes HARDENBO and NOHARDENBO are ignored. The `Backoutcount` of a message is always hardened for persistent messages and never hardened for non-persistent messages. This affects the following commands:
ALTER QLOCAL
ALTER QMODEL
DEFINE QLOCAL
DEFINE QMODEL
DISPLAY QUEUE

## CONNAME

The CONNAME attribute of TCP channels can optionally take an additional field at the start of the value, specifying the name of a specific Guardian TCP/IP Server process to be used for the channel. This affects the following commands:

ALTER CHANNEL
DEFINE CHANNEL
DISPLAY CHANNEL
DISPLAY CHSTATUS
DISPLAY CLUSQMGR

## USERDATA for triggered programs

Data passed to the trigger monitor via the USERDATA attribute of MQSC DEFINE PROCESS or ALTER PROCESS must be in double quotation marks if it is a string containing spaces. For example, if this USERDATA `-o $DISK.VOLUME.PROGRAM -d 1` is to be passed to the trigger monitor, it must be specified on input to MQSC in double quotation marks, as follows:

```
'" -o $DISK.VOLUME.PROGRAM -d 1"'
```

If you display the process definition via MQSC, it appears as follows:

```
 "-o $DISK.VOLUME.PROGRAM -d 1"
```

# Using exit names as attributes of objects

Wherever exit names are specified in attributes of objects, they will be in a format specific to MQSeries for Compaq NSK.

# Appendix H. Application Programming Reference

The following sections are new to MQSeries for Compaq NSK, and should be used in conjunction with the *MQSeries Application Programming Reference*.

## Structure data types

This section describes changes to data types.

| **Structure Data Type** | **Supported in V2.x** | **Supported in V5.1** | **Works as described in** *MQSeries Application Programming Reference* |
|---|---|---|---|
| MQBO – Begin Options | No | No | |
| MQCH – CICS Bridge Header | No | Yes | Yes |
| MQCNO – Connect Options | No | Yes | Yes but with some additional notes. See “MQCNO – Connect Options” on page 316 for more information. |
| MQDH – Distribution Header | No | Yes | Yes |
| MQDLH – Dead Letter Header | Yes | Yes | Yes |
| MQGMO – Get Message Options | Yes | Yes | Yes but with some additional notes. See “MQGMO – Get Message Options” on page 316 for more information. |
| MQIH – IMS Bridge Header | Yes | Yes | Yes |
| MQMD – Message Descriptor | Yes | Yes | Yes but with some additional notes. See “MQMD – Message Descriptor” on page 317 for more information. |
| MQMDE – Message Descriptor Extension | No | Yes | Yes |
| MQOD – object Descriptor | Yes | Yes | Yes |
| MQOR – Object Record | No | Yes | Yes |
| MQPMO – Put Message Options | Yes | Yes | Yes but with some additional notes. See “MQPMO – Put Message Options” on page 317 for more information. |
| MQPMR – Put Message Record | No | Yes | Yes |
| MQRMH – Message Reference Header | No | Yes | Yes |
| MQRR – Response Record | No | Yes | Yes |

© Copyright IBM Corp. 1993, 2001

## Structure data types

| Structure Data Type | Supported in V2.x | Supported in V5.1 | Works as described in *MQSeries Application Programming Reference* |
|---|---|---|---|
| MQTM – Trigger Message | Yes | Yes | Yes |
| MQTMC2 – Trigger Message Character Format | Yes | Yes | Yes |
| MQWIH – Workload Information Header | No | Yes | Yes |
| MQXQH – Transmission Queue Header | Yes | Yes | Yes |

This section describes the following MQSeries structure data types:

# MQCNO – Connect Options

The MQCNO data structure is as specified in *MQSeries Application Programming Reference* with the following additional notes:

- The unit of execution is defined as a process
- MQCNO_FASTPATH_BINDING can be used only in a process that has a single connection to a queue manager
- MQCNO_FASTPATH_BINDING requires that the application is run under the user ID that is a part of the MQM Administrative User Group that created the queue manager
- The Guardian parameter MQCONNECTTYPE can be used in association with the bind type specified by the *Options* field, to control the type of binding used. If this parameter is specified, it should have the value **FASTPATH** or **STANDARD**; if it has some other value, it is ignored. The value of the parameter is case sensitive.
- MQSeries for Compaq NSK supports MCNO_VERSION_2 as well as MQCNO_VERSION_1, but the *ClientConnOffset* and *ClientConnPtr* fields are ignored.

# MQGMO – Get Message Options

The MQGMO structure is an input/output parameter of the MGET call. Note the following information about the MQGMO_SET_SIGNAL, MQGMO_WAIT, MQGMO_SYNCPOINT, and MQGMO_NO_SYNCPOINT options in MQSeries for Compaq NSK:

- If you want the application to proceed with other work while waiting for a message to arrive, consider using the signal option MQGMO_SET_SIGNAL instead of MQGMO_WAIT. However, the signal option is environment specific and should not be used by applications that are to be ported between different environments.
- If there is more than one MQGET call waiting for the same message with a mixture of wait and signal options, each waiting call is considered equally. It is an error to specify MQGMO_SET_SIGNAL with MQGMO_WAIT. It is also an error to specify this option with a queue handle for which a signal is outstanding.
- If an application specifies MQGET with MQGMO_SET_SIGNAL and a WaitInterval of 0, the MQGMO_SET_SIGNAL option will be ignored and treated as an MQGET with MQGMO_NO_WAIT.

This means that an application must be prepared to receive MQRC_NO_MSG_AVAILABLE on an MQGET with MQGMO_SET_SIGNAL if the WaitInterval can ever be zero. Applications receive a signal IPC only if:

- The application experiences MQRC_SIGNAL_REQUEST_ACCEPTED from the MQGET (indicates that a signal has been *posted*)
- the application has been able to process the file_open_ system message and accept the signal IPC within the queue server's timeout for signal delivery. This is 60s by default, but may be overridden for a queue server by specifying the MQQSSIGTIMEOUT PARAM in the environment of the queue server.

The queue manager logs the failure to deliver an IPC message to an application if it has not been able to open the process and send the IPC before the timeout expires. At this point the queue manager will not attempt delivery again. Applications should be resilient to this by not waiting indefinitely for an IPC signal.

- MQGMO_SYNCPOINT_IF_PERSISTENT is now supported
- If neither of the options MQGMO_SYNCPOINT or MQGMO_NO_SYNCPOINT is set, MQSeries for Compaq NSK defaults to MQGMO_SYNCPOINT.
- MQSeries for Compaq NSK does not support the *MsgToken* field.

## MQMD – Message Descriptor

The MQMD structure contains the control information that describes a message. Please note the following information:

- The *BackoutCount* functions as described in the *MQSeries Application Programming Reference*. This is a count of the number of times the message has been previously returned by the MQGET call as part of a unit of work, and subsequently backed out. It is provided as an aid to the application in detecting processing errors that are based on message content. In Version 2.2.0.1, the BackoutCount was estimated.
- In MQSeries for Compaq NSK, the discarding of a message (and generation of a report, if required) is not performed during an MQGET call, but is under the control of the queue server that performs periodically, according to settings for the queue manager.
- The value of the UserIdentifier field, when set by the queue manager during an MQPUT or MQPUT1 is the MQSeries Principal name found in the queue manager's Principal database corresponding to the effective user identifier of the application.

## MQPMO – Put Message Options

The MQPMO structure is an input/output parameter of the MQPUT and MQPUT1 calls. Please note the following information about the MQPMO_NO_SYNCPOINT option in MQSeries for Compaq NSK:

- If neither of the options MQPMO_SYNCPOINT or MQPMO_NO_SYNCPOINT is set, MQSeries for Compaq NSK defaults to MQPMO_SYNCPOINT.

## MQI calls

This section describes changes to the following MQI calls:

| MQI Call Description | Supported in V2.x | Supported in V5.1 | Works as described in *MQSeries Application Programming Reference* |
|---|---|---|---|
| MQBACK – Back Out Changes | Returns error[1] | Returns error[1] | Yes |
| MQBEGIN – Begin Unit of Work | No | Returns error[1] | Yes |
| MQCLOSE – Close object | Yes | Yes | Yes but with some additional notes. See "MQCLOSE – Close Object" on page 319 for more information. |
| MQCMIT – Commit changes | No | Returns error[1] | Yes |
| MQCONN – Connect queue manager | Yes | Yes | Yes |
| MQCONNX – Connect Queue manager (Extended) | No | Yes | Yes |
| MQDISC – Disconnect queue manager | Yes | Yes | Yes but with some additional notes. See "MQDISC – Disconnect queue manager" on page 319 for more information. |
| MQGET – Get Message | Yes | Yes | Yes |
| MQINQ – Inquire About Object Attributes | Yes | Yes | Yes but with some additional notes. See "MQINQ – Inquire about object attributes" on page 319 for more information. |
| MQOPEN – Open object | Yes | Yes | Yes but with some additional notes. See "MQOPEN – Open Object" on page 319 for more information. |
| MQPUT – Put Message | Yes | Yes | Yes |
| MQPUT1 – Put One Message | Yes | Yes | Yes |
| MQSET – Set Object Attributes | Yes | Yes | Yes but with some additional notes. See "MQSET– Set Object Attributes" on page 319 for more information. |
| MQSYNC – Synchronize Statistics Updates | Returns error[2 on page 319] | Returns error[2 on page 319] | Yes |

**Notes:**

1. The MQI call can be issued by the application but always returns completion code MQCC_FAILED and reason code MQRC_ENVIRONMENT_ERROR.

2. This call always returns a *CompCode* of MQCC_OK and a reason code of MQRC_NONE.

## MQCLOSE – Close Object

The MQCLOSE call, which is the inverse of the MQOPEN call, relinquishes access to an object.

On MQSeries for Compaq NSK, if there is a MQGET request with the MQGMO_SET_SIGNAL option outstanding against the queue handle being closed, the request is canceled. Signal requests for the same queue but lodged against different handles (*Hobj*) are not affected (unless it is a dynamic queue that is being deleted, in which case, they are also canceled.)

For a FASTPATH application opening or closing a dynamic queue, MQSeries may start and end a TM/MP transaction in order to update audited databases. If the application has opened the TM/MP T-file (because it can initiate multiple transactions) then ENDTRANSACTION is a no-waited operation, and the application will receive a completion for the transaction initiated by MQSeries. Applications should review their design to determine if this is the case and verify that the logic handling completions can cope with ENDTRANSACTION completions that are caused by MQSeries.

## MQDISC – Disconnect queue manager

The MQDISC call, which is the inverse of MQCONN, breaks the connection between the MQSeries queue manager and the application program.

Usage Note 3 in the *MQSeries Application Programming Reference* is incorrect. On MQSeries for Compaq NSK an implicit syncpoint does not occur if a queue manager coordinated unit of work is in progress when MQDISC is called. This is because the NSK queue manager cannot be a coordinator of a UOW. Coordination is provide by the TM/MP subsystem.

## MQINQ – Inquire about object attributes

The MQPUT call returns an array of integers and a set of character strings containing the attributes of an object.

## MQOPEN – Open Object

The MQOPEN call establishes access to an object. On MQSeries for Compaq NSK, the *MaxHandles* attribute of the queue manager is ignored.

For a FASTPATH application opening or closing a dynamic queue, MQSeries may start and end a TM/MP transaction in order to update audited databases. If the application has opened the TM/MP T-file (because it can initiate multiple transactions) then ENDTRANSACTION is a no-waited operation, and the application will receive a completion for the transaction initiated by MQSeries. Applications should review their design to determine if this is the case and verify that the logic handling completions can cope with ENDTRANSACTION completions that are caused by MQSeries.

## MQSET– Set Object Attributes

The MQSET call changes the attributes of an object represented by a handle. The object must be a queue. On MQSeries for Compaq NSK, the MQIA_DIST_LISTS selector is supported.

For a FASTPATH application changing an object's attributes using MQSET, MQSeries will start and end a TM/MP transaction in order to update audited databases. If the application has opened the TM/MP T-file (because it can initiate multiple transactions) then ENDTRANSACTION is a no-waited operation, and the application will receive a completion for the transaction initiated by MQSeries. Applications should review their design to determine if this is the case and verify that the logic handling completions can cope with ENDTRANSACTION completions that are caused by MQSeries.

# Attributes of MQSeries objects

In MQSeries for Compaq NSK, the attributes of all objects are as described in the *MQSeries Application Programming Reference*, with the following exceptions and additions.

## Attributes for all queues

In MQSeries for Compaq NSK, the attributes of all queues are as described in the *MQSeries Application Programming Reference*, with the following exceptions and additions.

The AlterationDate and AlterationTime attributes are updated only when administrative changes are made to attributes of an object. CurrentQDepth, OpenInputCount and OpenOutputCount attributes may only be changed dynamically. QDepthHighCount, QDepthLowEvent, QDepthMaxEvent and QServiceIntervalEvent may be changed both dynamically and administratively, but only the administrative changes (such as is performed using MQSC commands or via MQSET) will cause a change in the AlterationDate and AlterationTimes attributes.

## Attributes of local and model queues

In MQSeries for Compaq NSK,

- the *Archive* attribute is ignored.
- The *HardenGetBackout* attribute is ignored because the backout count is not saved to disk. There is no ability to archive messages.
- For persistent messages, the *BackoutCount* attribute is always hardened. For non-persistent messages, the *BackoutCount* attribute is never hardened. If, however, the Local Queue has its -q server C option attribute set, *BackoutCount* will be checkpointed to the backup queue server. Messages checkpointed in this way are resilient against queue server failure. To maintain compatibility with other MQSeries platforms, the attribute may be queried by the MQINQ call using the MQIA_HARDEN_GET_BACKOUT selector.

## Attributes of queue managers

- *MaxMsgLength* is 100 MB.
- *CommandLevel* is MQCMDL_LEVEL_510.
- *SyncPoint* is MQSP_AVAILABLE.
- The value of *CodedCharSetId* is as specified when the queue manager instance was created.
- *MaxHandles* attribute is ignored. It is not possible to specify a maximum number of open handles for MQSeries for Compaq NSK. The maximum value will be determined by system resource constraints.

- *MaxUncommittedMsgs* attribute is ignored. It is not possible to specify a maximum number of messages to be allowed within a single unit of work. The maximum value is determined by resource constraints.
- CCSID can be altered.

## Data conversion

Refer to "Appendix L. User exits" on page 341 which describes the scheme for supporting all exits on MQSeries for Compaq NSK V5.1. The mechanism has changed from previous versions to support a more consistent and portable exit implementation.

# Appendix I. Building and running applications

The sample programs and the sample compilation and binding scripts, provided in subvolume ZMQSSMPL, illustrate the main features of the MQI in MQSeries for Compaq NSK, and demonstrate how to compile and bind an application.

## Writing applications

This section provides updated information for some minor differences between the standard Version 5.1 MQI interface, as documented in the *MQSeries Application Programming Guide*, and the MQI interface for MQSeries for Compaq NSK. Use this section to update the *MQSeries Application Programming Guide* for MQSeries for Compaq NSK V5.1.

### Using MQGET Wait Interval and Channel DISCINT and HBINT

When performing MQGET using MQGMO_NO_SYNCPOINT, a TM/MP transaction is started and ended by MQSeries only when a persistent message is available that satisfies the retrieval criteria. No consideration needs to be given by applications to the value of WaitInterval for no syncpoint operation.

For an MQGET issued with the MQGMO_SYNCPOINT or MQGMO_SYNCPOINT_IF_PERSISTENT option, the TM/MP transaction is under the control of the user application which issues the BEGINTRANSACTION. The wait interval should not exceed the TMF Autoabort timeout value and ideally should be small to avoid pinning a significant amount of the TM/MP audit trail (values under a minute should normally be used). Specifying wait unlimited on a lightly used queue or a queue that is idle overnight may cause the autoabort timeout to be exceeded and a MQRC_UOW_CANCELLED (2297) to be returned to the MQGET when a message becomes available on the queue. Having a high wait interval or using unlimited can cause TMF audit trails to be pinned, eventually leading (if uncorrected) to the TM/MP subsystem disabling transactions on a system-wide basis.

Similarly the DISCINT value and HBINT value for sending channels controls the length of a TM/MP transaction. Channels are capable of cycling transactions when idle to allow long disconnect intervals, without having a detrimental affect on TM/MP audit trails.

A parameter, MQTRANSACTIONLIFE, can be used to control the refreshing of the TM/MP transaction for channel disconnect intervals and heartbeats that are zero. This is useful if a longer or shorter TM/MP transaction life is desired or to change the amount TM/MP activity the idle channel produces. A higher value will produce less, a lower setting more.

Add to each MQSeries MQS-ECxx PATHWAY server class:

```
PARAM MQTRANSACTIONLIFE <number>
```

where <*number*> is a number such as 100.

© Copyright IBM Corp. 1993, 2001

This parameter overrides the use of the 10 second default TM/MP transaction refresh interval. For example, a channel with a disconnect interval unlimited without the parameter would cause a refresh approximately every 10 seconds while it waits for a message to arrive.

## Unit of work (transaction) management

Transaction management is performed under the control of Compaq's TM/MP product, rather than by MQSeries itself.

The effects of this difference are:

- The default SYNCPOINT option for the MQPUT and MQGET calls is SYNCPOINT, rather than NO_SYNCPOINT.
- To use the default (SYNCPOINT) option for MQPUT, MQGET and MQPUT1 operations, the application must have an active TM/MP Transaction that defines the unit of work to be committed. An application initiates a TM/MP transaction by calling the BEGINTRANSACTION() function. All MQPUT, MQPUT1 and MQGET operations performed by the application while this transaction is active are within the same unit of work (transaction). Any other database operations performed by the application are also within this UOW. Note that there are system-imposed limits on the number and size of messages that can be written and deleted within a single TM/MP transaction. When the application has completed the UOW, the TM/MP transaction is ended (the UOW is committed) using the ENDTRANSACTION() function. If any error is encountered, the application can cancel the TM/MP transaction (backout the UOW) using the ABORTTRANSACTION() function. Consequently, the standard Version 5 functions MQCMIT(), MQBACK() and MQBEGIN() are not supported on this product. If they are called, an error is returned.
- If an application uses the NO_SYNCPOINT option for MQPUT, MQGET and MQPUT1 operations, MQSeries starts a TM/MP transaction itself, performs the queueing operation, and commits the transaction before returning to the application. Each operation is therefore performed in its own UOW and, once complete, cannot be backed out by the application using TM/MP.
- It is necessary for MQSeries to start a TM/MP transaction itself for a NO_SYNCPOINT operation only if the message is persistent and therefore requiring update to a TM/MP protected queue file.
- A TM/MP transaction does not need to be active for MQI calls other than MQGET, MQPUT and MQPUT1.
- The MQRC_SYNCPOINT_LIMIT_REACHED reason code is used by MQSeries for Compaq NSK to inform an application that the system-imposed limit on the number of I/O operations within a single TM/MP transaction has been reached. If the application specified the SYNCPOINT option, it should cancel the transaction (backout the UOW) and retry with a smaller number of operations in that UOW.
- The MQRC_UOW_CANCELED reason code informs the application that the UOW (TM/MP transaction) has been canceled, either by the system itself (TM/MP imposes some system-wide resource-usage thresholds that will cause this), by user action, or by the initiator of the transaction itself.
- The MQRC_BACKED_OUT reason code informs the application that MQSeries was forced to cancel the UOW because of an error, or Primary Queue Server failure. The application should call ABORTTRANSACTION (if the operation was syncpoint) and retry.
- The MQRC_SYNCPOINT_NOT_AVAILABLE reason code informs the application that MQSeries was unable to start or use a TM/MP transaction that

was required in order to complete an operation. Typically this indicates a problem with TM/MP, and additional information may be available in the error log or from FFSTs produced by the queue manager.

## General design considerations

Note that:

- The MQI library (bound into the application process) does not open $RECEIVE and does not open $TMP (TM/MP transaction pseudo-file) itself, so you may code your application to use these features.
- The MQI library uses a SERVERCLASS_SEND_() call in initial communication with the queue manager. While connected, it maintains a minimum of two process file opens (with the LINKMON process and a Local Queue Manager Agent) and a small number of disk file opens (fewer than 10). Process opens are also made to any queue servers that support local queues that are opened for input, output or browse as a result of an MQOPEN call.
- You should ensure that there is no outstanding nowait PATHSEND I/O before calling MQCONN. MQCONN performs nowaited PATHSEND I/O and could intercept the completion of the application's outstanding I/O causing errors.
- FASTPATH-bound applications have special considerations if they are also multi-threaded TM/MP requesters (see the descriptions of MQOPEN, MQCLOSE and MQSET earlier).

## XA interface

No XA interface for unit of work (UOW) coordination is provided. All UOW coordination is performed by TM/MP.

## MQGMO_BROWSE_* with MQGMO_LOCK

MQGMO_BROWSE_* with MQGMO_LOCK is now supported. See the *MQSeries Application Programming Reference*.

## Triggered applications

Triggered MQSeries applications in the Compaq NSK environment receive user data through environment variables set up in the TACL process that is running. This is because there is a limit to the length of the argument list that can be passed to a Compaq C process.

In order to access this information, triggered applications should contain code similar to the following (see sample amqsinqa for more details):

```
MQTMC2  *trig;                      /* trigger message structure     */
MQTMC2  trigdata;               /* trigger message structure     */
char     *applId;
char     *envData;
char     *usrData;
char     *qmName;

/******************************************************************/
/*                                                                */
/*   Set the program argument into the trigger message            */
/*                                                                */
/******************************************************************/
trig = (MQTMC2*)argv[1];        /* -> trigger message   */

/* get the environment variables and load the rest of the trigger */
memcpy(&trigdata, trig, sizeof(trigdata));

memset(trigdata.ApplId,   ' ', sizeof(trigdata.ApplId));
memset(trigdata.EnvData,  ' ', sizeof(trigdata.EnvData));
memset(trigdata.UserData, ' ', sizeof(trigdata.UserData));
memset(trigdata.QMgrName, ' ', sizeof(trigdata.QMgrName));


if( (applId = getenv("TRIGAPPLID")) != 0)
{
  strncpy( trigdata.ApplId  ,applId, strlen(applId) );
}

if ( (envData = getenv("TRIGENVDATA")) != 0)
{
  strncpy( trigdata.EnvData , envData, strlen(envData) );
}

if ( (usrData = getenv("TRIGUSERDATA")) != 0)
{
  strncpy( trigdata.UserData, usrData, strlen(usrData) );
}


if ( (qmName = getenv("TRIGQMGRNAME")) != 0)
{
  strncpy( trigdata.QMgrName, qmName, strlen(qmName) );
}

trig = &trigdata;
```

## Supported languages and environments

MQSeries for Compaq NSK V5.1 supports the languages and environments described in Table 24. The table also describes whether the application can use FASTPATH or STANDARD bindings.

*Table 24. Summary of supported languages and environments*

| Language | Runs on Guardian?[1] | Runs on OSS? | Can use STANDARD binding? | Can use FASTPATH binding?[1] |
|---|---|---|---|---|
| C native | Yes | Yes | Yes | Yes |

*Table 24. Summary of supported languages and environments (continued)*

| Language | Runs on Guardian?[1] | Runs on OSS? | Can use STANDARD binding? | Can use FASTPATH binding?[1] |
|---|---|---|---|---|
| C non-native | Yes | No | Yes | No |
| COBOL native | Yes | Yes | Yes | Yes |
| COBOL non-native | Yes | No | Yes | No |
| C++ native | Yes | Yes | Yes | Yes |
| TAL non-native | Yes | No | Yes | No |
| NonStop Java[2] | No | Yes | Yes | No |

**Notes:**

1. The Guardian environment and FASTPATH-bound OSS applications cannot use threads. Only OSS STANDARD-bound and Java can use threads. For more information on using threads in your application, see "Considerations for creating applications with threads".
2. NonStop Java applications use the Java Transaction Services (JTS) for transactions.
3. A native application that uses FASTPATH binding may resolve the MQI only through MQSeries for Compaq NSK's Shared Resource Library. A native application that uses STANDARD bindings is able to resolve the MQI using either MQSeries for Compaq NSK's Shared Resource Library or a static Native MQI library. For more information about FASTPATH and STANDARD binding, see "FASTPATH versus STANDARD bindings" on page 328.

## Considerations for creating applications with threads

Guardian applications do not support threads. They may implement their own cooperative threading mechanism, but the rules for using the MQI from the Guardian environment must be obeyed for the process that is using it.

In the OSS environment, a thread emulation package based on POSIX threads is available. The emulation implements a cooperative scheduling mechanism where a thread must give up execution control before the code in any other thread can execute. Applications can use this threading package to organize processing into threads, but the following restrictions apply:

- The MQI does not support cooperative scheduling between threads. This means that when the MQI is called from an application thread, no other thread can obtain execution control, regardless of how long it takes. For example, if a thread calls MQGET with the 'wait indefinitely ' option, no other thread can execute in the application process until the MQGET returns.
- FASTPATH-bound applications running in an OSS environment cannot use threads.
- The queue manager does not support multi-threaded Local Queue Manager Agents (LQMA or MQLQMAG processes) or Message Channel Agents (MCAs).

## Compiling and binding applications

The MQSeries for Compaq NSK MQI is implemented using the Compaq wide memory model (the `int` datatype is 4 bytes) and the Common Runtime Environment (CRE). Applications must be compatible with this environment in order to work correctly. Refer to the sample build files for the correct options for each compiler in order to ensure compatibility.

In particular, TAL and COBOL applications must follow the rules that are required for compatibility with the CRE, documented in the Compaq books relating to the CRE.

Note that, for successful use of the MQGMO_SET_SIGNAL function of MQGET, you must set the HIGHREQUESTERS attribute to ON in object code for COBOL and TAL applications.

For an installation, three versions of the MQI library are delivered with MQSeries for Compaq NSK, contained in ZMQSLIB. You must ensure that you use the correct library, as shown in Table 25.

*Table 25. Using the correct version of the MQI library*

| Programming Language | Nonnative | Native/Static | Native/Dynamic |
|---|---|---|---|
| TAL | MQMLIB | Not applicable | Not applicable |
| COBOL | MQMLIB | MQMLIBN | MQSRLLIB |
| C | MQMLIB | MQMLIBN | MQSRLLIB |
| C++ | Not applicable | MQMLIBN | MQSRLLIB |

## FASTPATH versus STANDARD bindings

MQSeries for Compaq NSK V5.1 supports both FASTPATH and STANDARD bindings. Table 25 describes the languages and environments that support each type of binding.

### STANDARD bindings

Consider the following when using STANDARD bindings in an application:

- Non-native and native applications can use STANDARD bindings.
- A native application that uses STANDARD bindings can resolve the MQI using either:
  - MQSeries for Compaq NSK’s Shared Resource Library
  - A static Native MQI library. This provides support for applications that already use a shared resource library.

### FASTPATH bindings

Consider the following when using FASTPATH bindings in an application:

- Only native applications can use FASTPATH binding.
- A native application using a FASTPATH binding can resolve the MQI only through MQSeries for Compaq NSK’s Shared Resource Library.
- FASTPATH-bound applications running on OSS cannot use threads.
- FASTPATH-bound applications must run under the User ID in the Compaq NSK MQM Administrative User group that created the queue manager.

## Running applications

To be able to connect to a queue manager, the environment of an application program must be correctly defined:

- The PARAM MQDEFAULTPREFIX is mandatory in the environment of all applications.
- If you have chosen an alternative (nondefault) location for your MQSINI file, an application will not be able to connect to the queue manager if the PARAM MQMACHINIFILE is not set correctly.
- TAL and COBOL applications must have the PARAM SAVE-ENVIRONMENT ON defined in their environment, or they will not be able to connect to the queue manager.

An application may run as either low-pin or high-pin. MQSeries executables themselves are configured to run as high-pin.

MQSeries applications are supported in both the Guardian and OSS environments.

An MQSeries application may run under PATHWAY, from TACL, or as a child process of another process. Applications can even be added to the queue manager PATHWAY configuration itself, provided they behave correctly on queue manager shutdown.

# Appendix J. MQSeries Administration Interface (MQAI)

MQSeries for Compaq NSK V5.1 supports the MQAI interface.

The MQAI is a programming interface to MQSeries, using the C language. It performs administration tasks on an MQSeries queue manager using data bags. Data bags allow you to handle properties (or parameters) of objects in a way that is easier than using the other administration interface, Programmable Command Formats (PCFs).

The MQAI offers easier manipulation of PCFs than using the MQGET and MQPUT calls. You can use the MQAI to:

- Implement self-administering applications and administration tools.
- Simplify the use of PCF messages. The MQAI is an easy way to administer MQSeries; you do not have to write your own PCF messages and thus avoid the problems associated with complex data structures.
- Handle error conditions more easily. It is difficult to get return codes back from the MQSeries commands (MQSC), but the MQAI makes it easier for the program to handle error conditions.

**Note:** MQSeries for Compaq NSK V5.1 provides C header files only. It does not provide Visual Basic header files.

For more information about the MQAI, see the *MQSeries Administration Interface Programming Guide and Reference*

© Copyright IBM Corp. 1993, 2001

# Appendix K. MQSeries for Compaq NSK sample programs

The following C and COBOL sample programs are supplied with MQSeries for Compaq NSK V5.1:

| Description | C source | C executable | COBOL85 source | COBOL85 executable |
|---|---|---|---|---|
| Read and output message descriptor and context for each message on a queue | amqsbcg0 | amqsbcg | No sample | No sample |
| Echo a message from a message queue to the reply-to queue | amqsecha | amqsech | amq0ech0 | amq0ech |
| Write messages from a queue to stdout, leave messages on the queue (Browse) | amqsgbr0 | amqsgbr | amq0gbr0 | amq0gbr |
| Remove messages from the named queue and write to stdout | amqsget0 | amqsget | amq0get0 | amq0get |
| Read the triggered queue, respond with queue information | amqsinqa | amqsinq | No sample | No sample |
| Use a shared input queue | No sample | No sample | amq0inq0 | amq0inq |
| Copy stdin to a message and put the message on a specified queue | amqsput0 | amqsput | amq0put0 | amq0put |
| Put a request message on a specified queue and display the replies | amqsreq0 | amqsreq | amq0req0 | amq0req |
| (Trigger function) inhibit puts on a named queue and respond with a statement of the result | amqsseta | amqsset | amq0set0 | amq0set |
| Trigger monitor | amqstrg0 | amqstrg | No sample | No sample |
| Sample skeleton for data conversion exit | amqsvfcn | No sample | No sample | No sample |
| Sample skeleton for channel exit | amqsvchn | No sample | No sample | No sample |
| Sample skeleton for cluster workload exit | amqswlm0 | No sample | No sample | No sample |
| Sample skeleton for MQLOADEXIT | amqslxp0 | No sample | No sample | No sample |

The following TAL sample programs are supplied with MQSeries for Compaq NSK V5.1:

| Description | TAL source | TAL executable |
|---|---|---|
| Read *n* messages from a queue | zmqreadt | zmqread |
| Write *n* messages of *n* length to a queue | zmqwritt | zmqwrit |

© Copyright IBM Corp. 1993, 2001

# Building C sample programs

## Non-Native (using non-native static library MQMLIB)

The subvolume ZMQSSMPL contains the following TACL macro files to be used for building non-native sample C applications:

**CSAMP**

Usage: `CSAMP` *`source-code-file-name`*

This is a basic macro for compiling a C source file using the include files contained in subvolume ZMQSLIB. For example, to compile the sample AMQSBCG0, use `CSAMP AMQSBCG0`. If the compilation is successful, the macro produces an object file with the last character of the file name replaced by the letter O; for example, AMQSBCGO.

**BSAMP**

Usage: `BSAMP` *`exe-file-name`*

This is a basic macro used to bind an object file with the user library MQMLIB in ZMQSLIB. For example, to bind the compiled sample AMQSBCG0, use `BSAMP AMQSBCG`. The macro produces an executable file called *`exe-file-name`*`E`; for example, AMQSBCGE.

**COMPALL**

Usage: `COMPALL`

This TACL macro compiles each of the sample source code files using the CSAMP macro.

**BINDALL**

Usage: `BINDALL`

This TACL macro binds each of the sample object files into executables using the BSAMP macro.

**BUILDC**

Usage: `BUILDC`

This TACL macro compiles and binds all of the C sample files using the macros COMPALL and BINDALL.

## Native (using native static library MQMLIBN)

For a native install, the following TACL macro files are to be used for building sample MQI applications:

**NMCALL**

Usage: `NMCALL`

Macro to compile all samples native using NMCSAMP.

**NMCSAMP**

Usage: `NMCSAMP` *`source-code-file-name`*

This is a basic macro for compiling a C source file using the include files contained in subvolume ZMQSLIB. For example, to compile the sample AMQSBCG0, use `NMCSAMP AMQSBCG0`. If the compilation is successful, the macro produces an object file with the last character of the file name replaced by the letter O; for example, AMQSBCGO.

**NMLDSAMP**

Usage: `NMLDSAMP` *`exe-file-name`*

This basic macro links an object file with the static Native MQI library MQMLIBN in ZMQSLIB.

**NMLDALL**
Usage: `NMLDALL`

This TACL macro binds each of the sample object files into executables using the NMLDSAMP macro.

**NMBUILDC**
Usage: `NMBUILDC`

This TACL macro compiles and binds all of the Native C sample files using the macros NMCALL and NMLDALL.

## Native (using SRL MQSRLLIB)

**NMLDSSMP**
Usage: `NMLDSSMP` *`exe-file-name`*

This basic macro links an object file with the Native MQ SRL MQSRLLIB in ZMQSLIB.

**NMLDSALL**
Usage: `NMLDSALL`

This TACL macro binds each of the sample object files into executables using the NMLDSSMP macro.

**NMBULDSC**
Usage: `NMBULDSC`

This TACL macro compiles and binds all of the Native C sample files using the macros NMCALL and NMLDSALL.

**NMLDPSRL**
Usage: `NMLDPSRL` *`exe-file-name`*

This basic macro links an object file with the MQSeries private SRL in ZMQSLIB

**NMCPSRL**
Usage: `NMCPSRL` *`source-code-file-name`*

Macro to compile user code for inclusion in the MQSeries PSRL.

**NMLDUSRL**
Usage: `NMLDUSRL` *`object-input-file`*, where *`object-input-file`* is a file containing a list of objects to be linked.

This is a basic macro for linking user code into a relinkable library.

**Note:** Non-native applications can connect to native queue managers, and vice versa. All combinations of native and non-native operation are valid and supported.

# Building C++ sample programs

## Native (using native static library MQMLIBN)

**NMCPPALL**
Usage: `NMCPALL`

Macro to compile all samples native using NMCCPP.

### Building C++ samples

**NMCCPP**
Usage: `NMCCPP` *`source-code-file-name`*

This is a basic macro for compiling a C++ source file using the include files contained in subvolume ZMQSLIB. For example, to compile the sample IMQSGETP, use `NMCCPP IMQSGETP`. If the compilation is successful, the macro produces an object file with the last character of the file name replaced by the letter O; for example, IMQSGETO.

**NMLDCPP**
Usage: `NMLDCPP` *`exe-file-name`*

This basic macro links an object file with the Static Native MQI library MQMLIBN in ZMQSLIB.

**NMLDCPPA**
Usage: `NMLDCPPA`

This TACL macro binds each of the sample object files into executables using the NMLDSAMP macro.

**NMBLDCPP**
Usage: `NMBUILDC`

This TACL macro compiles and binds all of the Native C++ sample files using the macros NMCPPALL and NMLDCPPA.

## Native (using SRL MQSRLLIB)

**NMLDCPPS**
Usage: `NMLDCPPS` *`exe-file-name`*

This basic macro links an object file with the Native MQ SRL MQSRLLIB in ZMQSLIB.

**NMLDCPSA**
Usage: `NMLDCPSA`

This TACL macro binds each of the sample object files into executables using the NMLDCPPS macro.

**NMBLDSCP**
Usage: `NMBLDSCP`

This TACL macro compiles and binds all of the Native C sample files using the macros NMCPPALL and NMLDCPSA.

# Building COBOL sample programs

## Non-Native (using non-native static libary MQMLIB)

The subvolume ZMQSSMPL contains the following files to be used for building sample COBOL applications.

**COBSAMP**
Usage: `COBSAMP` *`source-code-file-name`*

This is a basic macro for compiling a COBOL source file using the definition files contained in subvolume ZMQSLIB. For example, to compile the program AMQ0GBR0, use `COBSAMP AMQ0GBR0`. If the compilation is successful, the macro produces an object file with the last character of the file name replaced by the letter O; for example AMQ0GBRO.

**BCOBSAMP**

Usage: `BCOBSAMP` *`exe-file-name`*

This is a basic macro used to bind an object with the user libraries in ZMQSLIB. For example to bind the compiled sample AMQ0GBRO, use `BCOBSAMP AMQ0GBR`. The macro produces an executable called *exe-file-name* AMQ0GBR.

**CCBSMPLS**

Usage: `CCBSMPLS`

This TACL macro compiles each of the COBOL sample source code files.

**BCBSMPLS**

Usage: `BIND /IN BCBSMPLS/`

This bind input file binds each of the COBOL sample object files into executables.

**BUILDCOB**

Usage: `BUILDCOB`

This TACL macro compiles and binds all of the COBOL sample files using the macros CCBSMPLS and BCBSMPLS.

## Native (using native static library MQMLIBN)

**NMCOBSMP**

Usage: `NMCOBSMP` *`source-code-file-name`*

This is a macro for compiling Native mode COBOL 'NMCOBOL' using the MQSeries Native Library MQMLIBN in ZMQSLIB. For example, to compile the program AMQ0GBR0, use `NMCOBSMP AMQ0GBR0`. If the compilation is successful, the macro produces an object file with the last character of the file name replaced by the letter O; for example AMQ0GBRO.

**NMLDCOB**

Usage: `NMLDCOB` *`exe-file-name`*

This macro binds object with the MQSeries Native library MQMLIBN in ZMQSLIB. For example to bind the compiled sample AMQ0GBRO, use `NMLDCOB AMQ0GBR`. The macro produces an executable called *exe-file-name* AMQ0GBR.

**NMCOBALL**

Usage: `NMCOBALL`

This TACL macro compiles each of the COBOL sample source code files using NMCOBSMP.

**NMLDACOB**

Usage: `NMLDACOB`

This bind input file binds each of the NMLDCOB sample object files into executables.

**NMBLDCOB**

Usage: `NMBLDCOB`

This TACL macro compiles and binds all of the COBOL sample files using the macros NMCOBALL and NMLDACOB.

## Native (using SRL MQSRLLIB)

**NMCBSSMP**

Usage: `NMCBSSMP` *`source-code-file-name`*

### Building COBOL samples

This is a macro for compiling Native mode COBOL ’NMCOBOL’ using the MQSeries SRL MQSRLLIB in ZMQSLIB. For example, to compile the program AMQ0GBR0, use `NMCBSSMP AMQ0GBR0`. If the compilation is successful, the macro produces an object file with the last character of the file name replaced by the letter O; for example AMQ0GBRO.

**NMLDSCOB**
Usage: `NMLDSCOB` *`exe-file-name`*

This macro binds object with the MQSeries SRL MQSRLLIB in ZMQSLIB. For example to bind the compiled sample AMQ0GBRO, use `NMLDSCOB AMQ0GBR`. The macro produces an executable called *exe-file-name* AMQ0GBR.

**NMCBSALL**
Usage: `NMCBSALL`

This TACL macro compiles each of the COBOL sample source code files using NMCBSSMP.

**NMLDSCOB**
Usage: `NMLDSCOB`

This bind input file binds each of the NMLDSCOB sample object files into executables.

**NMBLDSCB**
Usage: `NMBLDSCB`

This TACL macro compiles and binds all of the COBOL sample files using the macros NMCBSALL and NMLDSCOB.

## Building TAL sample programs

### Non-Native (using non-native static library MQMLIB)

The subvolume ZMQSSMPL contains the following files to be used for building sample TAL programs.

**TALSAMP**
Usage: `TALSAMP` *`source-code-file-name`* This is a basic macro for compiling a TAL source file using the definition files contained in subvolume `Appendix I. MQSeries for Compaq NSK sample programs 335.ZMQSLIB`. For example, to compile the program ZMQWRITT, use `TALSAMP ZMQWRITT`. If the compilation is successful, the macro produces an object file with the last character of the file name replaced by the letter O; for example ZMQWRITO.

**BTALSAMP**
Usage: `BTALSAMP` *`exe-file-name`*

This is a basic macro used to bind an object with the user libraries in ZMQSLIB. For example to bind the compiled sample ZMQWRITO, use `BTALSAMP ZMQWRIT`.

**CTLSMPLS**
Usage: `CTLSMPLS`

This TACL macro compiles each of the TAL sample source code files.

**BTLSMPLS**
Usage: `BIND /IN BTLSMPLS/`

This bind input file binds each of the TAL sample object files into executables.

**BUILDTAL**

Usage: `BUILDTAL`

This TACL macro compiles and binds all of the TAL sample files using the macros CTLSMPLS and BTLSMPLS.

## Building sample programs on OSS (Native mode only)

The directory - `/opt/mqm/samp` contains the sample programs for MQSeries and the make file MQMAKE.SMP. This MakeFile contains all the targets needed to build all the C, C++ and NMCOBOL samples

The directory - `/opt/mqm/inc` contains all the copylibs and header files needed to build programs on OSS.

**Note:** The MQSeries SRL file and the native MQI library—MQMLIBN exist only on the Guardian file system. You will need to edit your build scripts and make files to point to them if needed, for example, `/G/system/zmqslib/mqsrllib`.

## Building samples on OSS

# Appendix L. User exits

MQSeries for Compaq NSK V5.1 supports channel exit programs, data conversion exit programs and the Cluster Workload Management (CLWL) exit program. In addition, a Compaq NSK specific load program exit is supported. For information about channel exits, see the *MQSeries Intercommunication* book. For information about data conversion exits, see the *MQSeries Application Programming Guide* and the *MQSeries Application Programming Reference*. For information about Cluster Workload Management exits, see the *MQSeries Queue Manager Clusters* book.

This appendix provides information specific to the use of exit programs in MQSeries for Compaq NSK.

## Supported user exits

Table 26 lists the characteristics of each type of user exit supported for MQSeries for Compaq NSK.

*Table 26. User exits supported for MQSeries for Compaq NSK*

| User Exit Type | Exit Name Length Maximum | Exit Data Length Maximum | Where enabled | Chained? |
|---|---|---|---|---|
| Channel MSG Exit | 32 | 32 | DEFINE CHANNEL | Yes |
| Channel SEND Exit | 32 | 32 | DEFINE CHANNEL | Yes |
| Channel RECEIVE Exit | 32 | 32 | DEFINE CHANNEL | Yes |
| Channel SECURITY Exit | 32 | 32 | DEFINE CHANNEL | |
| Channel MSGRETRY Exit | 32 | 32 | DEFINE CHANNEL | |
| Channel Auto-Definition Exit | 32 | 32 | ALTER QMGR | |
| Cluster Workload Management Exit | 32 | 32 | ALTER QMGR | |
| Data Conversion Exit | 8 | Not applicable | Unknown FORMAT name | |
| MQ_LOAD_ENTRY _POINT_EXIT | Fixed Name | Not applicable | Called when any of the above exits are required or enabled. | |

MQ_LOAD_ENTRY_POINT_EXIT is the only user exit that is specific to Compaq NSK.

© Copyright IBM Corp. 1993, 2001

## Exit name format

Exit names (other than MQ_LOAD_ENTRY_POINT_EXIT) can be any alpha-numeric string up 32 characters long. For exits that support an associated data field, the data can be any string up 32 characters long.

## MQ_LOAD_ENTRY_POINT_EXIT - Loading User Exits

All user exit programs must be linked into the MQSeries Private SRL or static library. User exit programs must contain at least one external function (symbol) that can be called by MQSeries when required.

Before enabling any other MQSeries user exit, you must install an MQ_LOAD_ENTRY_POINT_EXIT program to map your exit names to entry-point addresses. Your MQ_LOAD_ENTRY_POINT_EXIT program must be linked into the MQSeries SRL or static library, and is called by MQSeries whenever one of the other user exits is enabled.

The MQ_LOAD_ENTRY_POINT_EXIT program's name is fixed, that is, its external function name must remain MQ_LOAD_ENTRY_POINT.

The MQ_LOAD_ENTRY_POINT_EXIT is free to map an exit name to any entry point address or to map many exit names to the same entry-point address.

MQSeries supplies a stub MQ_LOAD_ENTRY_POINT_EXIT function that always returns MQXCC_FAILED (Exit Name not found) when called. You must replace this stub exit with your own before enabling any of the other user exits.

```
MQ_LOAD_ENTRY_POINT_EXIT (ExitParms)
```

Parameters:

```
Exitparms (PMQLXP) – input/output
        LoadExit Parameter Block
```

**Usage notes:** The function performed by the MQ_LOAD_ENTRY_POINT_EXIT program is defined by the provider of the exit.

Figure 46 on page 345 contains a sample MQ_LOAD_ENTRY_POINT_EXIT that maps three exit names to entry point addresses.

### MQLXP - MQ_LOAD_ENTRY_POINT_EXIT parameter structure

The MQLXP structure describes the information that is passed to the load exit.

This structure is supported for Compaq NSK only.

#### Fields

**StrucId (MQCHAR4)**

Structure identifier.

The value is: `MQLXP_STRUC_ID`.

Identifier for load exit parameter structure.

For the C programming language, the constant MQLXP_STRUC_ID_ARRAY is also defined. This has the same value as MQLXP_STRUC_ID, but is an array of characters instead of a string.

This is an input field to the exit.

**Version (MQLONG)**

Structure version number.

The value is: `MQLXP_VERSION_1`

Version-1 load exit parameter structure.

The following constant specifies the version number of the current version: `MQLXP_CURRENT_VERSION`

Current version of load exit parameter structure.

This is an input field to the exit.

**QMgrName (MQCHAR48)**

Name of local queue manager.

This is the name of the queue manager that has invoked the load exit. The name is padded with blanks to the length of the field.

This is an input field to the exit.

**EntryPointName (MQCHAR32)**

Name of the requested Entry Point.

This is the name of the Entry Point that the load exit needs to resolve to a callable address. The name is padded with blanks to the length of the field.

This is an input field to the exit.

**EntryAddress (PMQFUNC)**

Returned Callable Address.

This is the address of the requested EntryPoint.

This is an output field from the exit.

**ExitResponse (MQLONG)**

Response from exit.

This is set by the exit to indicate whether resolving of the Entry Name to a callable address was successful. It must be one of the following:

```
MQXCC_OK
   Success.
```

This indicates that processing of the exit successfully resolved the EntryPointName supplied in the ExitParms to a callable adddress. The callable address is returned in the EntryAddress field in the MQLXP structure.

```
MQXCC_FAILED
 Failed.
```

This indicates that the exit was unable to resolve the EntryPointName supplied in the ExitParms to a callable adddress.

Any other value that is returned in the ExitResponse field has the same meaning as MQXCC_FAILED.

This is an output field from the exit.

### MQ_LOAD_ENTRY_POINT_EXIT example

Figure 46 on page 345 is an example of a working MQ_LOAD_ENTRY_POINT_EXIT program that maps three exit names (two channels exits and one data-conversion exit) to entry point addresses. The source code for the MQ_LOAD_ENTRY_POINT_EXIT sample program is provided in the samples subvolume (AMQSLXP0).

```
 /********************************************************************/
 /*                                                                  */
 /* Program name: AMQSLXP0              (Compaq NSK only)            */
 /*                                                                  */
 /* Description: Sample C skeleton of a Load Exit function           */
 /*                                                                  */
 /* Statement:     Licensed Materials - Property of IBM              */
 /*                                                                  */
 /*                (C) Copyright IBM Corp. 1993, 2001                */
 /*                                                                  */
 /********************************************************************/
 /*                                                                  */
 /* Function:                                                        */
 /*                                                                  */
 /*   AMQSLXP0 is a sample C skeleton of a Load Exit function        */
 /*                                                                  */
 /*   The function resolves EntryNames to callable addresses         */
 /*                                                                  */
 /*                                                                  */
 /*   Once complete the code should be compiled into a loadable      */
 /*   object, the name of the object should be the name of the       */
 /*   format to be converted. Instructions on how to do this are     */
 /*   contained in the README file in this directory.                */
 /*                                                                  */
 /********************************************************************/
 /*                                                                  */
 /*  AMQSLXP0 takes the parameters defined for a Load Exit           */
 /*  routine in the CMQXC.H header file.                             */
 /*                                                                  */
 /********************************************************************/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <limits.h>

#include <cmqc.h>
#include <cmqxc.h>


 /********************************************************************/
 /* Load Exit                                                        */
 /*                                                                  */
 /*                                                                  */
 /********************************************************************/
void
MQENTRY MQ_LOAD_ENTRY_POINT(
     PMQLXP  pExitParms             /* exit Parameter */
     )
 {

   /* No loadable entry points are defined */
   pExitParms->ExitResponse = MQXCC_FAILED;

   return;
 }
 /********************************************************************/
 /* End of AMQSLXP0                                                  */
 /********************************************************************/
```

*Figure 46. Sample MQLOADEXIT*

### Supported exit programs

With the above MQ_LOAD_ENTRY_POINT_EXIT program and the channel and data-conversion exits installed, you can enable your channel receive and send exit using the following MQSC command:

```
ALTER CHANNEL(CHAN) CHLTYPE(SDR) SENDEXIT(MY_CHANNEL_SEND_EXIT)
ALTER CHANNEL(CHAN) CHLTYPE(SDR) RCVEXIT(MY_CHANNEL_RCV_EXIT)
```

The Data Conversion exit will be called by MQSeries when an MQGET is done with conversion enabled (MQGMO_CONVERT) and the message format name is MY_FORMAT.

# Installing user exits

All user exits that you create need to be installed into the MQSeries private SRL. Additionally, some exit types need to installed in any MQSeries static libraries used by application programs.

| User Exit Type | Where Installed |
|---|---|
| Channel MSG Exit | Private SRL |
| Channel SEND Exit | Private SRL |
| Channel RECEIVE Exit | Private SRL |
| Channel SECURITY Exit | Private SRL |
| Channel MSGRETRY Exit | Private SRL |
| Channel Auto-Definition Exit | Private SRL |
| Cluster Workload Management Exit | Private SRL |
| Data Conversion Exit | Private SRL (also native static library and/or non-native static library if used by applications) |
| MQ_LOAD_ENTRY_POINT _EXIT | Private SRL (also native static library and/or non-native static library if used by applications) |

## Installing an exit in the MQSeries private SRL

To install an exit program into the MQSeries private SRL, create a new version of the MQSeries private SRL containing the new exit:

1. Compile the exit function. For example:

```
NMCPSRL  AMQSVFCN
```

2. Compile the MQLOAD entry point function. For example:

```
NMCPSRL  AMQSLXP0
```

3. Link the exit and entry point objects into a relinkable library for use in the SRL. For example:

```
NMLDUSRL  OBJECTS EXITS
```

4. Create a new version of the MQSeries private SRL called NEWMQSRL by linking this data conversion object with the relinkable version of the MQSeries private SRL. For example:

```
MAKEPSRL EXITS $VOL.ZMQSLIB NEWMQSRL
```

5. Stop all queue managers and applications accessing the current MQSeries private SRL.
6. Relink all MQSeries applications to the new PSRL. For example:

```
NMLDEXES $VOL.ZMQSLIB.NEWMQSRL $VOL.ZMQSEXE
```

7. Compile the get application. For example:

```
NMCSAMP AMQSGET0
```

8. Relink all user applications to the new PSRL. For example:

```
AMQSGET  NMLDPSRL AMQSGET
```

9. Restart MQSeries and all MQI applications.

Steps 6 and 8 are quite fast, but can be omitted if the new MQSeries PSRL is placed in ZMQSLIB and called MQSRLLIB. The steps can be repeated to link to a different MQSeries PSRL.

## Installing an exit in the MQSeries native static library

An exit can be linked with the chosen application and the MQI library by using the TACL macro NMLDEXIT. For example:

```
NMLDEXIT Object-File Exit-Object-File
```

## Installing an exit in the MQSeries non-native static library

An exit can be bound into the chosen executable (or library) using the TACL macro BEXITE.

**Note:** This procedure modifies the target executable; it is recommended you make a backup copy of the target executable or library before using the macro.

Exit functions, once compiled, must be bound directly into the target executable or library to be accessible by MQSeries. The TACL macro, BEXITE, is used for this purpose. For example:

## Supported exit programs

```
BEXITE Target-Executable-Or-Library Source-Exit-File-Or-Library
```

For example, to bind the sample data conversion exit into the sample MQSGETA, follow these steps:

1. Compile the exit function. For example:

```
CSAMP AMQSVFCN
```

2. Compile the MQLOAD entry point function. For example:

```
CSAMP AMQSLXP0
```

3. Compile the get application. For example:

```
CSAMP AMQSGET0
```

4. Bind the get application. For example:

```
BSAMP AMQSGET
```

5. Bind the exit function into the get application. For example:

```
BEXITE AMQSGET AMQSVFCO
```

6. Bind the entry point function into the get application. For example:

```
BEXITE AMQSGET AMQSLXPO
```

Alternatively, if all applications are to have this data conversion exit, the following steps would create both a user library and an application with the exit bound in:

1. Compile the exit function. For example:

```
CSAMP AMQSVFCN
```

2. Compile the MQLOAD entry point function. For example:

```
CSAMP AMQSLXP0
```

3. Compile the get application. For example:

```
CSAMP AMQSGET0
```

4. Bind the exit function into the user library. For example:

```
BEXITE ZMQSLIB.MQMLIB AMQSVFCO
```

5. Bind the exit function into the user library. For example:

```
BEXITE ZMQSLIB.MQMLIB AMQSLXPO
```

6. Bind the get application with the modified library. For example:

```
BSAMP AMQSGET
```

# Appendix M. Setting up communications

This appendix describes how to set up communications for MQSeries for Compaq NSK using the SNA and TCP/IP communications protocols. The following examples are provided:

- "SNAX communications example" on page 358
- "ICE communications example" on page 363
- "TCP/IP communications example" on page 367

## SNA channels

The following channel attributes are necessary for SNA channels in MQSeries for Compaq NSK V5.1:

**CONNAME**

The value of CONNAME depends on whether SNAX or ICE is used as the communications protocol:

*If SNAX is used*:

**CONNAME('$PPPP.LOCALLU.REMOTELU')**

Applies to sender, requester and fully qualified server channels, where:

**$PPPP** Is the process name of the SNAX/APC process.

**LOCALLU** Is the name of the Local LU.

**REMOTELU** Is the name of the partner LU on the remote machine.

For example:

```
CONNAME('$BP01.IYAHT080.IYCNVM03')
```

*If ICE is used*:

**CONNAME('$PPPP.#OPEN.LOCALLU.REMOTELU')**

Applies to sender, requester and fully qualified server channels, where:

**$PPPP** Is the process name of the ICE process.

**#OPEN** Is the ICE open name.

**LOCALLU** Is the name of the Local LU.

**REMOTELU** Is the name of the partner LU on the remote machine.

For example:

```
CONNAME('$ICE.#IYAHT0C.IYAHT0C0.IYCNVM03')+
```

**MODENAME**

Is the SNA mode name. For example, MODENAME(LU62PS).

**TPNAME('LOCALTP[.REMOTETP]')**

Is the Transaction Process (TP) name.

© Copyright IBM Corp. 1993, 2001

**LOCALTP** Is the name of the server class (usually MQSeries) in the PATHWAY used for SNA communication. In the case of SNAX, the server class must exist in the same PATHWAY as the SNAX dispatcher and the APC Process' server class. In the case of ICE, it must be in the PATHWAY declared in the NOF - ADD TP command.

**REMOTETP** Is the name of the TP on the remote machine. This value is optional. If it is not specified, and the channel is one that initiates a conversation (that is, a sender, requester, or fully qualified server channel) the LOCALTP name is used.

Both the LOCALTP and REMOTETP values can be up to 16 characters in length.

## LU 6.2 responder processes

In MQSeries for Compaq NSK V5.1, a SNA Listener process is needed to listen for incoming attach requests from remote queue manager channels.

MQSeries SNA listeners must be dispatched by the SNA product, when an incoming Attach arrives from a remote channel.

Using the SNAX APC Dispatcher allows SNAX to accept incoming Attach requests from partner Transaction Programs. To do this, the SNAX Dispatcher requires that:

- the SNAX Dispatcher is run in a different PATHWAY from the queue managers
- the APC process is run as a server class from that same PATHWAY

Using Insession's ICE requires:

- That a transaction program (server class) is defined in a PATHWAY
- That a transaction program is added in NOF that points to this server class.

There is no separate ICE Dispatcher, the ICE process itself handles incoming Attach requests.

### SNAX TP dispatching

The Compaq SNAX SNA product supports the starting of APPC transaction programs (TPs) when an APPC Attach arrives from a partner transaction program. A SNAX Dispatcher dispatches these requests to its associated SNAX $APC process via a local transaction program server class.

The SNAX Dispatcher for MQSeries has the following requirements:

- The SNAX Dispatcher must run in the same PATHWAY as the associated $APC process.
- Each incoming TPNAME must be defined as a server class (usually named MQSERIES) in the same PATHWAY as the Dispatcher and the $APC process.
- The server class program name is the **runmqlsr** program that exists in the MQSeries executables subvolume (usually ZMQSEXE).
- The Dispatcher process starts the server class and passes all relevant information ($APC Process, LUName, TPName) to this server class(TP) in a DISPATCH-TP IPC request.

### ICE TP dispatching

ICE Version 3.2 implements its SNA Attach Manager similarly to SNAX in that its TP is a PATHWAY server class. The ICE process accepts an Attach request and is itself the Dispatcher. However this ICE process does not need to run in the same PATHWAY as the TP (ServerClass). The environment in this case has the following requirements:

- An active Ice process must be running.
- A Dispatch TP must be added in NOF. For example,

```
ADD TP <tpname>, PROCESS <$process>, SERVERCLASS <Serverclass name>
```

  *Serverclass name* is usually MQSeries.
- Each incoming TPNAME be defined as a server class (usually named MQSERIES) in the PATHWAY<$process>.
- The server class program name is the RUNMQLSR program that exists in the MQSeries executables subvolume (usually ZMQSEXE)
- The ICE process starts the server class and passes all relevant information ($ICE Process, ApplName, TPName) to this server class(TP) in a DISPATCH-TP IPC request.

## Sample SNA environment setup

The following are examples of how to configure your SNA environments.

**Using SNAX APC:** If using SNAX APC:

- A PATHWAY must be created to be used exclusively for this listener
- An APC process serverclass needs to be run from this PATHWAY.

Enter the following at a TACL prompt:

```
TACL> Pathmon /name $PMAP, nowait, out $vhs, cpu 3/4
TACL> Pathcom $PMAP
= O LU62SCFG
```

where LU62SCFG is an edit file containing the following:

```
[ SET PATHMON BACKUPCPU 6
SET PATHWAY MAXTCPS 10
SET PATHWAY MAXTERMS 10
SET PATHWAY MAXPROGRAMS 10
SET PATHWAY MAXSERVERCLASSES 10
SET PATHWAY MAXSERVERPROCESSES 10
SET PATHWAY MAXSTARTUPS 10
SET PATHWAY MAXPATHCOMS 40
SET PATHWAY MAXASSIGNS 32
SET PATHWAY MAXPARAMS 32
START PATHWAY COLD!

SET TCP PROGRAM $ SYSTEM.SYSTEM.PATHTCP2
SET TCP CPUS 3:4
SET TCP MAXTERMS 5
SET TCP MAXSERVERCLASSES 010
SET TCP MAXSERVERPROCESSES 010
SET TCP MAXTERMDATA 08960
SET TCP MAXREPLY 20000  SET TCP NONSTOP 0
SET TCP TCLPROG $system.system.APCP
ADD TCP SNAXAPC-TCP
```

*Figure 47. Sample MQSeries SNAX setup file (Part 1 of 3)*

```
[Configure the SNAX/APC SERVER]
RESET SERVER
SET SERVER PARAM LOGFILE APCLOG
SET SERVER PARAM TRACEFILE APCTRC
SET SERVER PARAM BACKUPCPU -1
SET SERVER PARAM MAXINRUSIZE 4096
SET SERVER PARAM MAXOUTRUSIZE 4096
SET SERVER PARAM MAXAPPLIOSIZE 4096
SET SERVER PARAM DATAPAGES 100
SET SERVER PARAM TRACEPAGES 300
SET SERVER PARAM RMTATTACHDISP QUEUE
SET SERVER PARAM RMTATTACHTIMER -1
SET SERVER PARAM CONFIG APCCFG
SET SERVER PROGRAM $system.system.APCOBJ
SET SERVER OUT $VHS
SET SERVER HOMETERM $VHS
SET SERVER PROCESS $AP02
SET SERVER NUMSTATIC 1
SET SERVER MAXSERVERS 1
SET SERVER CREATEDELAY 0 SECS
SET SERVER DELETEDELAY 1 MINS
SET SERVER CPUS 3:4
ADD SERVER SNAXAPCSVR
```

*Figure 47. Sample MQSeries SNAX setup file (Part 2 of 3)*

```
[Add MQSeries SNAX Listener]
RESET SERVER
SET SERVER PROGRAM $DATA00.ZMQSEXE.RUNMQLSR
SET SERVER PROCESS $lrcv
SET SERVER NUMSTATIC 1
SET SERVER MAXSERVERS 1
SET SERVER CREATEDELAY 0 SECS
SET SERVER DELETEDELAY 1 MINS
SET SERVER STARTUP "-t LU62"
SET SERVER PARAM MQQUEMGRNAME "QMGR"
SET SERVER PARAM MQMACHINIFILE "$DATA03.QMGRD.UMQSINI"
SET SERVER PARAM MQDEFAULTPREFIX "$DATA00"
SET SERVER OUT $VHS
SET SERVER HOMETERM $VHS
SET SERVER CPUS 3:4
[ADD SERVER MQSERIES]
ADD SERVER MQSERIES
START TCP *

[Configure the DISPATCHER]
SET TERM FILE $s.#displog
SET TERM INITIAL SNAXAPC-DISPATCHER
SET TERM TYPE CONVERSATIONAL
SET TERM TCP SNAXAPC-TCP
ADD TERM SNAXAPCSVR01 [First 10 chars are the SNAX/APC server name]

start server MQSERIES
start server SNAXAPCSVR
start term SNAXAPCSVR01
```

*Figure 47. Sample MQSeries SNAX setup file (Part 3 of 3)*

**Note:** The Listener server class is identical to the MQS-TCPLIS00 server class in the queue managers own PATHWAY, except there is an addition startup parameter: SET SERVER STARTUP ″-t LU62″

## Using Insession ICE

If you are using Insession ICE, a PATHWAY should be created to be used exclusively for this listener. The ICE process is not run from this PATHWAY.

1. Add the TP in NOF as follows:

```
ADD TP <tpname>, PROCESS <process>, SERVERCLASS <server> [, <option> ...]
```

where:

**process** is the name of the PATHMON process that manages the TP

**server** is the name of the PATHMON SERVERCLASS to which the TP belongs

**option** can be
- [ATTACHTIMER n] -- the amount of time (in hundredths of a second) ICE will wait for an ATTACH after dispatching a new TP thread. The default is 6000 (60 seconds).
- [MAXDISPATCHTHREADS n] -- maximum number of simultaneous dispatched TP’s DEFAULT: 0 (no limit on the number of simultaneous DISPATCHED TP’s)
- [TIMEOUT n] -- determines how ICE will respond to an ATTACH if MAXDISPATCHTHREADS has been reached on a TP TIMEOUT -1 = ATTACH is queued indefinitely TIMEOUT >0 = ATTACH is queued for n/100 seconds DEFAULT: 0 ( ATTACH is rejected immediately)

2. It is still necessary to add the server class to the PATHWAY. At the TACL prompt, enter:

```
TACL> Pathmon /name $PMAP, nowait, out $vhs, cpu 3/4
TACL> Pathcom $PMAP
 = O LU62ICFG
```

where LU62ICFG is an edit file containing the following:

```
[
SET PATHMON BACKUPCPU 6
SET PATHWAY MAXTCPS 10
SET PATHWAY MAXTERMS 10
SET PATHWAY MAXPROGRAMS 10
SET PATHWAY MAXSERVERCLASSES 10
SET PATHWAY MAXSERVERPROCESSES 10
SET PATHWAY MAXSTARTUPS 10
SET PATHWAY MAXPATHCOMS 40
SET PATHWAY MAXASSIGNS 32
SET PATHWAY MAXPARAMS 32
START PATHWAY COLD!
SET TCP PROGRAM $SYSTEM.SYSTEM.PATHTCP2
SET TCP CPUS 3:4
SET TCP MAXTERMS 5
SET TCP MAXSERVERCLASSES 010
SET TCP MAXSERVERPROCESSES 010
SET TCP MAXTERMDATA 08960SET TCP MAXREPLY 20000
SET TCP NONSTOP 0
SET TCP TCLPROG $system.system.APCP
ADD TCP SNAXAPC-TCP
```

*Figure 48. Sample MQSeries SNA setup file for ICE (Part 1 of 2)*

```
[Add MQSeries ICE Listener]
RESET SERVER
SET SERVER PROGRAM $DATA00.ZMQSEXE.RUNMQLSR
SET SERVER PROCESS $lrcv
SET SERVER NUMSTATIC 1
SET SERVER MAXSERVERS 1
SET SERVER CREATEDELAY 0 SECS
SET SERVER DELETEDELAY 1 MINS
SET SERVER STARTUP "-t LU62"
SET SERVER PARAM MQQUEMGRNAME "QMGR"
SET SERVER PARAM MQMACHINIFILE "$DATA03.QMGRD.UMQSINI"
SET SERVER PARAM MQDEFAULTPREFIX "$DATA00"
SET SERVER OUT $VHS
SET SERVER HOMETERM $VHS
SET SERVER CPUS 3:4
[ADD SERVER MQSERIES]
ADD SERVER MQSERIES

START TCP *
start server MQSERIES
```

*Figure 48. Sample MQSeries SNA setup file for ICE (Part 2 of 2)*

**Note:** The Listener server class is identical to the MQS-TCPLIS00 server class in the queue managers own PATHWAY, except there is an addition startup parameter: SET SERVER STARTUP ″-t LU62″

## TCP/IP channels

MQSeries for Compaq NSK gives you the option of using multiple TCP/IP processes within a single MQSeries queue manager environment. This means you can select TCP/IP processes used within a queue manager by associating the required TCP/IP process with a given channel. Outbound Channels (Sender, Server, Requester) can specify the required TCP/IP Process Name in the CONNAME field of the channel definition.

Using **runmqsc**:

```
alter channel ... conname ('$ZTC1.123.456.789.012(1415)')
alter channel ... conname ('$ZTC1.dnshostname(1415)')
```

Using the MQMC panels:

```
TCPIP/SNA Process:  $ZTC1
```

Using PCF commands:

```
strncpy( pPCFString->String, '('$ZTC1.123.456.789.012(1415)', len );
```

To reconfigure DNS resolution for non-default resolver, add to all PATHWAY ECnn server classes the following:

```
DEFINE =TCPIP^RESOLVER^NAME, FILE filename
```

where `filename` is the location of the resolver file.

If using a hosts file, add to all PATHWAY ECnn server classes the following:

```
DEFINE =TCPIP^HOST^FILE, FILE filename
```

where `filename` is the location of the hosts file.

Inbound channels use environment variables to determine which TCP/IP process to use. The TCP/IP Listeners pass this process value to their respective ECs and onto their respective TCP/IP Responder processes via their agents. To set the inbound channel TCP/IP process:

Using TACL:

```
ADD DEFINE =TCPIP^PROCESS^NAME, FILE processname
```

where `processname` is the name of the TCP/IP process.

### Setting up communications

Using PATHWAY, for MQS-TCPLIS*nn* server classes, where *nn* is the Listener server class number:

```
DEFINE =TCPIP^PROCESS^NAME, FILE \HAWK.$ZTC1
PARAM MQLISTENPORTNUM "1415"
```

For information about using a nondefault TCP/IP process for communications via TCP/IP, see “Reconfiguring the MQS-TCPLISnn server class for a nondefault TCP/IP process and port” on page 50. For information about the TCP/IP ports a queue manager listens on, see “TCP/IP ports listened on by the queue manager” on page 49.

# Communications examples

This section provides communications setup examples for SNA (SNAX and ICE) and TCP/IP.

## SNAX communications example

This section provides:
- An example SCF configuration file for the SNA line
- Some example SYSGEN parameters to support the line
- An example SCF configuration file for the SNA process definition
- Some example MQSC channel definitions

### SCF SNA line configuration file

Here is an example SCF configuration file:

```
==
== SCF configuration file for defining SNA LINE, PUs and LUs to VTAM®
== Line is called $SNA02 and SYSGEN'd into the Compaq system
==

ALLOW ALL
ASSUME LINE $SNA02

ABORT, SUB LU
ABORT, SUB PU
ABORT

DELETE, SUB LU
DELETE, SUB PU
DELETE

==
== ADD $SNA02 LINE DEFINITION
==

ADD LINE $SNA02, STATION SECONDARY, MAXPUS 5, MAXLUS 1024, RECSIZE 2048, &
        CHARACTERSET ASCII, MAXLOCALLUS 256, &
        PUIDBLK %H05D, PUIDNUM %H312FB

==
== ADD REMOTE PU OBJECT, LOCAL IS IMPLICITLY DEFINED AS #ZNT21
==

ADD PU #PU2, ADDRESS 1, MAXLUS 16, RECSIZE 2046, TYPE (13,21), &
             TRRMTADDR 04400045121088, DYNAMIC ON, &
             ASSOCIATESUBDEV $CHAMB.#p2, &
             TRSSAP %H04, &
             CPNAME IYAQCDRM, SNANETID GBIBMIYA
```

```
==
== ADD LOCAL LU OBJECT
==

ADD LU #ZNTLU1, TYPE (14,21), RECSIZE 1024, &
          CHARACTERSET ASCII, PUNAME #ZNT21, SNANAME IYAHT080

==
== ADD PARTNER LU OBJECTS
==

== spinach (HP)

ADD LU #PU2LU1, TYPE(14,21), PUNAME #PU2, SNANAME IYABT0F0

== stingray (AIX)

ADD LU #PU2LU2, TYPE(14,21), PUNAME #PU2, SNANAME IYA3T995

== coop007 (OS/2)

ADD LU #PU2LU3, TYPE(14,21), PUNAME #PU2, SNANAME IYAFT170

== MVS CICS

ADD LU #PU2LU4, TYPE(14,21), PUNAME #PU2, SNANAME IYCMVM03

== MVS Non-CICS

ADD LU #PU2LU5, TYPE(14,21), PUNAME #PU2, SNANAME IYCNVM03

== finnr100 (NT)

ADD LU #PU2LU6, TYPE(14,21), PUNAME #PU2, SNANAME IYAFT080

== winas18 (AS400)

ADD LU #PU2LU7, TYPE(14,21), PUNAME #PU2, SNANAME IYAFT110

== MQ-Portugese (OS/2)

ADD LU #PU2LU8, TYPE(14,21), PUNAME #PU2, SNANAME IYAHT090

== VSE

ADD LU #PU2LU10, TYPE(14,21), PUNAME #PU2, SNANAME IYZMZSI2

== START UP TOKEN RING ASSOCIATE SUB DEVICE $CHAMB.#P2
== then start the line, pu's and lu's

START LINE $CHAMB, SUB ALL

START
START, SUB PU

STATUS
STATUS, SUB PU
STATUS, SUB LU
```

## SYSGEN parameters

The following are CONFTEXT file entries for a SYSGEN to support the SNA and token ring lines:

```
!***************************************************************************
!                       LAN MACRO
!***************************************************************************
!  This macro is used for all 361x LAN controllers
!  REQUIRES T9375 SOFTWARE PACKAGE

   C3613^MLAM         = MLAM
                        TYPE 56,             SUBTYPE 0,
                        PROGRAM              C9376P00,
                        INTERRUPT            IOP^INTERRUPT^HANDLER,
                        MAXREQUESTSIZE       32000,
                        RSIZE                32000,
                        BURSTSIZE            16,
                        LINEBUFFERSIZE       32,
                        STARTDOWN  #;
!***************************************************************************
!                       SNAX macro for Token ring lines
!***************************************************************************
TOKEN^RING^SNAX^MACRO = SNATS
                        TYPE 58,
                        SUBTYPE 4,
                        RSIZE 1024,
                        SUBTYPE 4,
                        FRAMESIZE 1036 # ;

!***************************************************************************
!                       SNAX MANAGER
!***************************************************************************
  SSCP^MACRO          = SNASVM
                        TYPE 13,         SUBTYPE 5,
                        RSIZE                256  #;

!***************************************************************************
!                       LAN CONTROLLER
!***************************************************************************
LAN1      3616    0,1     %130    ;

!***********   Service manager
SNAX     6999    0,1     %370    ;

!***********   SNAX/Token Ring Pseudocontroller
RING      6997    0,1     %360    ;

!*********** Token Ring Line
$CHAMB     LAN1.0, LAN1.1       C3613^MLAM, NAME #LAN1;

!***********   Configure the SSCP
 $SSCP     SNAX.0, SNAX.1 SSCP^MACRO;

!***********   Sna lines for Dummy Controller over Token Ring
 $SNA01    RING.0, RING.1 TOKEN^RING^SNAX^MACRO;
 $SNA02    RING.2, RING.3 TOKEN^RING^SNAX^MACRO;
```

## SNAX/APC process configuration

The following definitions configure the example APC process (process name $BP01) via SCF for the SNA line.

**Note:** The APC process $BP01 is defined as a server class process running in the same PATHWAY as the SNAX APC Dispatcher.

```
==
== SCF Configuration file for SNAX/APC Lus
==

ALLOW ERRORS
```

```
ASSUME PROCESS $BP01

ABORT  SESSION *
ABORT  TPN *
ABORT  PTNR-MODE *
ABORT  PTNR-LU *
ABORT  LU *

DELETE TPN *
DELETE PTNR-MODE *
DELETE PTNR-LU *
DELETE LU *

==
== ADD LOCAL LU
==
ADD LU IYAHT080, SNANAME GBIBMIYA.IYAHT080, SNAXFILENAME $SNA02.#ZNTLU1, &
                 MAXSESSION 256, AUTOSTART YES

== TPnames for MQSeries

ADD TPN IYAHT080.MQSeries

=== Spinach (HP) Partner LU

ADD PTNR-LU   IYAHT080.IYABT0F0, SNANAME GBIBMIYA.IYABT0F0, &
              PERIPHERAL-NODE NO, PARALLEL-SESSION-LU YES

ADD PTNR-MODE IYAHT080.IYABT0F0.LU62PS, MODENAME LU62PS, &
              DEFAULTMAXSESSION 8, DEFAULTMINCONWINNER 4, &
              DEFAULTMINCONLOSER 3, MAXAUTOACT 1, RCVWINDOW 4, &
              DEFAULTMAXINRUSIZE 1024, DEFAULTMAXOUTRUSIZE 1024, &
              SENDWINDOW 4
==
==  Winas18 (AS400) Partner LU
==

ADD PTNR-LU   IYAHT080.IYAFT110, SNANAME GBIBMIYA.IYAFT110, &
              PERIPHERAL-NODE NO, PARALLEL-SESSION-LU YES

ADD PTNR-MODE IYAHT080.IYAFT110.LU62PS, MODENAME LU62PS, &
              DEFAULTMAXSESSION 8, DEFAULTMINCONWINNER 4, &
              DEFAULTMINCONLOSER 3, MAXAUTOACT 1, RCVWINDOW 4, &
              DEFAULTMAXINRUSIZE 1024, DEFAULTMAXOUTRUSIZE 1024, &
              SENDWINDOW 4
==
==  Stingray (AIX) Partner LU
==

ADD PTNR-LU   IYAHT080.IYA3T995, SNANAME GBIBMIYA.IYA3T995, &
              PERIPHERAL-NODE NO, PARALLEL-SESSION-LU YES

ADD PTNR-MODE IYAHT080.IYA3T995.LU62PS, MODENAME LU62PS, &
              DEFAULTMAXSESSION 8, DEFAULTMINCONWINNER 4, &
              DEFAULTMINCONLOSER 3, MAXAUTOACT 1, RCVWINDOW 4, &
              DEFAULTMAXINRUSIZE 1024, DEFAULTMAXOUTRUSIZE 1024, &
              SENDWINDOW 4
==
== coop007 (OS/2) Partner LU
==

ADD PTNR-LU   IYAHT080.IYAFT170, SNANAME GBIBMIYA.IYAFT170, &
              PERIPHERAL-NODE NO, PARALLEL-SESSION-LU YES

ADD PTNR-MODE IYAHT080.IYAFT170.LU62PS, MODENAME LU62PS, &
              DEFAULTMAXSESSION 8, DEFAULTMINCONWINNER 4, &
              DEFAULTMINCONLOSER 3, MAXAUTOACT 1, RCVWINDOW 4, &
              DEFAULTMAXINRUSIZE 1024, DEFAULTMAXOUTRUSIZE 1024, &
              SENDWINDOW 4
==
== MQ-Portugese (OS/2) Partner LU
==

ADD PTNR-LU   IYAHT080.IYAHT090, SNANAME GBIBMIYA.IYAHT090, &
              PERIPHERAL-NODE NO, PARALLEL-SESSION-LU YES

ADD PTNR-MODE IYAHT080.IYAHT090.LU62PS, MODENAME LU62PS, &
              DEFAULTMAXSESSION 8, DEFAULTMINCONWINNER 4, &
              DEFAULTMINCONLOSER 3, MAXAUTOACT 1, RCVWINDOW 4, &
              DEFAULTMAXINRUSIZE 1024, DEFAULTMAXOUTRUSIZE 1024, &
              SENDWINDOW 4
==
== finnr100 (NT)  Partner LU
==

ADD PTNR-LU   IYAHT080.IYAFT080, SNANAME GBIBMIYA.IYAFT080, &
              PERIPHERAL-NODE NO, PARALLEL-SESSION-LU YES

ADD PTNR-MODE IYAHT080.IYAFT080.LU62PS, MODENAME LU62PS, &
              DEFAULTMAXSESSION 8, DEFAULTMINCONWINNER 4, &
              DEFAULTMINCONLOSER 3, MAXAUTOACT 1, RCVWINDOW 4, &
              DEFAULTMAXINRUSIZE 1024, DEFAULTMAXOUTRUSIZE 1024, &
              SENDWINDOW 4
==
== MVS CICS       Partner LU
==

ADD PTNR-LU   IYAHT080.IYCMVM03, SNANAME GBIBMIYA.IYCMVM03, &
              PERIPHERAL-NODE NO, PARALLEL-SESSION-LU YES

ADD PTNR-MODE IYAHT080.IYCMVM03.LU62PS, MODENAME LU62PS, &
              DEFAULTMAXSESSION 8, DEFAULTMINCONWINNER 4, &
              DEFAULTMINCONLOSER 3, MAXAUTOACT 1, RCVWINDOW 4, &
              DEFAULTMAXINRUSIZE 1024, DEFAULTMAXOUTRUSIZE 1024, &
              SENDWINDOW 4
==
== MVS Non CICS Partner LU
==

ADD PTNR-LU   IYAHT080.IYCNVM03, SNANAME GBIBMIYA.IYCNVM03, &
              PERIPHERAL-NODE NO, PARALLEL-SESSION-LU YES

ADD PTNR-MODE IYAHT080.IYCNVM03.LU62PS, MODENAME LU62PS, &
              DEFAULTMAXSESSION 8, DEFAULTMINCONWINNER 4, &
              DEFAULTMINCONLOSER 3, MAXAUTOACT 1, RCVWINDOW 4, &
              DEFAULTMAXINRUSIZE 1024, DEFAULTMAXOUTRUSIZE 1024, &
              SENDWINDOW 4
==
== VSE Partner LU
==

ADD PTNR-LU   IYAHT080.IYZMZSI2, SNANAME GBIBMIYA.IYZMZSI2, &
              PERIPHERAL-NODE NO, PARALLEL-SESSION-LU YES

ADD PTNR-MODE IYAHT080.IYZMZSI2.LU62PS, MODENAME LU62PS, &
              DEFAULTMAXSESSION 8, DEFAULTMINCONWINNER 4, &
              DEFAULTMINCONLOSER 3, MAXAUTOACT 1, RCVWINDOW 4, &
              DEFAULTMAXINRUSIZE 1024, DEFAULTMAXOUTRUSIZE 1024, &
              SENDWINDOW 4

==
== Start the LUs
==
START LU IYAHT080, SUB ALL
START TPN *
```

MQSeries applications require the `Maxapplio` value, which controls the maximum size of interprocess data transfers between MQSeries and the communications server process, to be set to 32000, which is larger than the default.

### Channel definitions

Here are some example MQSeries channel definitions that support the SNAX configuration:

- A sender channel to MQSeries on MVS/ESA (non-CICS mover):

```
DEFINE CHANNEL(MT01.VM03.SDRC.0002) CHLTYPE(SDR) +
        TRPTYPE(LU62)  +
        SEQWRAP(9999999) MAXMSGL(2048) +
        XMITQ('VM03NCM.TQ.SDRC.0001') +
        CONNAME('$BP01.IYAHT080.IYCNVM03') +
        MODENAME('LU62PS') TPNAME(MQSERIES)
```

- A receiver channel from MQSeries on MVS/ESA:

```
DEFINE CHANNEL(VM03.MT01.SDRC.0002) CHLTYPE(RCVR) +
        TRPTYPE(LU62) REPLACE DESCR('Receiver channel from VM03NCM') +
        SEQWRAP(9999999) +
        MAXMSGL(2048)
```

- A server channel to MQSeries on MVS/ESA which is capable of initiating a conversation, or being initiated by a remote requester channel:

```
DEFINE CHANNEL(MT01.VM03.RQSV.0002) CHLTYPE(SVR) +
         TRPTYPE(LU62)  +
         SEQWRAP(9999999) MAXMSGL(2048) +
         XMITQ('VM03NCM.TQ.RQSV.0001') +
         CONNAME('$BP01.IYAHT080.IYCNVM03') +
         MODENAME('LU62PS') TPNAME(MQSERIES)
```

where `MQSeries` is the TPNAME the MVS™ queue manager is listening on.

## ICE communications example

There are two stages in configuring ICE for MQSeries:

1. The ICE process itself must be configured.
2. Line ($ICE01, in the following example) and SNA information must be input to the ICE process.

### Configuring the ICE process

Here is an example ICE process configuration. This configuration is located by default in a file called GOICE:

```
?tacl macro
clear all
param backupcpu 1
param cinittimer 120
param collector $0
param config icectl
param idblk 05d
param idnum 312FF
param cpname IYAHR00C
param datapages 64
param dynamicrlu yes
param genesis $gen
param maxrcv 32000
param loglevel info
param netname GBIBMIYA
param password xxxxxxxxxxxxxxxxxxxx
param retrys1 5
param secuserid super.super
param startup %1%
param timer1 20
param timer2 300
param usstable default
run $system.ice.ice/name $ICE,nowait,cpu 0,pri 180,highpin off/
```

**Notes:**

1. The password PARAM has been replaced by xxxxxxxxxxxxxxxxxxxxx.
2. MQSeries applications require the `maxrcv` PARAM, which controls the maximum size of interprocess data transfers between MQSeries and the communications server process, to be set to 32000, which is larger than the default.

## Defining the line and APC information

Once the ICE process has been started with this configuration, the following information is input to the ICE process using the Node Operator Facility (NOF**). This example defines a line called $ICE01 running on the token ring port $CHAMB.#ICE:

```
==
== ICE definitions for PU IYAHR00C.
== Local LU for this PU is IYAHT0C0.
==

ALLOW ERRORS

OPEN $ICE

ABORT LINE $ICE01, SUB ALL

DELETE LINE $ICE01, SUB ALL

==
==  ADD TOKEN RING LINE
==

ADD LINE $ICE01, TNDM $CHAMB.#ICE, &
      IDBLK %H05D, &
      PROTOCOL TOKENRING, WRITEBUFFERSIZE 8192

==
== ADD PU OBJECT
==
ADD PU IYAHR00C, LINE $ICE01, MULTIROUTE YES, &
             DMAC 400045121088, DSAP %H04, &
             NETNAME GBIBMIYA, IDNUM %H312FF, IDBLK %H05D, &
             RCPNAME GBIBMIYA.IYAQCDRM, SSAP %H08

==
== Add Local APPL Object
==

DELETE APPL IYAHT0C0
ADD APPL IYAHT0C0, ALIAS IYAHT0C0, PROTOCOL CPIC, &
         OPENNAME #IYAHT0C

==
== Add Mode LU62PS
==

DELETE MODE LU62PS
ADD MODE LU62PS, MAXSESS 8, MINCONWIN 4, MINCONLOS 3

==
== Add Partner LU Objects
==

== spinach (HP)

ABORT RLU IYABT0F0
DELETE RLU IYABT0F0
ADD RLU IYABT0F0, MODE LU62PS, PARSESS YES

== stingray (AIX)

ABORT RLU IYA3T995
DELETE RLU IYA3T995
ADD RLU IYA3T995, MODE LU62PS, PARSESS YES

== coop007 (OS/2)

ABORT RLU IYAFT170
DELETE RLU IYAFT170
ADD RLU IYAFT170, MODE LU62PS, PARSESS YES

== MVS CICS

ABORT RLU IYCMVM03
DELETE RLU IYCMVM03
ADD RLU IYCMVM03, MODE LU62PS, PARSESS YES

== MVS Non-CICS

ABORT RLU IYCNVM03
DELETE RLU IYCNVM03
ADD RLU IYCNVM03, MODE LU62PS, PARSESS YES

== finnr100 (NT)

ABORT RLU IYAFT080
DELETE RLU IYAFT080
ADD RLU IYAFT080, MODE LU62PS, PARSESS YES

== winas18 (AS400)

ABORT RLU IYAFT110
DELETE RLU IYAFT110
ADD RLU IYAFT110, MODE LU62PS, PARSESS YES

ABORT RLU IYAHT080
```

```
DELETE RLU IYAHT080
ADD RLU IYAHT080, MODE LU62PS, PARSESS YES

==
== START UP ICE LINE $ICE01 AND SUB DEVICE
==

START LINE $ICE01, SUB ALL
```

**Note:** For this configuration to work, the port #ICE must have been defined to the token ring line.

For example, these commands could be entered into SCF:

```
    add port $chamb.#ice, type tr8025, address %H08
    start port $chamb.#ice
```

where $chamb is a token-ring controller, and the SAP of the port is %08.

## Channel definitions for ICE

Here are some MQSeries channel definitions that would support this ICE configuration:

- A sender channel to MQSeries on MVS/ESA (non-CICS mover):

```
     DEFINE CHANNEL(MT01.VM03.SDRC.ICE) CHLTYPE(SDR) +
             TRPTYPE(LU62)  +
             SEQWRAP(9999999) MAXMSGL(2048) +
             XMITQ('VM03NCM.TQ.SDRC.ICE') +
             CONNAME('$ICE.#IYAHT0C.IYAHT0C0.IYCNVM03')+
             MODENAME('LU62PS') TPNAME(MQSERIES)
```

- A receiver channel from MQSeries on MVS/ESA:

```
     DEFINE CHANNEL(VM03.MT01.SDRC.ICE) CHLTYPE(RCVR) +
             TRPTYPE(LU62) REPLACE DESCR('Receiver channel from VM03NCM') +
             SEQWRAP(9999999) +
             MAXMSGL(2048) +
            TPNAME(VM03NCMSDRCRCVR)
```

- A server channel to MQSeries on MVS/ESA that is capable of initiating a conversation, or being initiated by a remote requester channel:

```
    DEFINE CHANNEL(MT01.VM03.RQSV.ICE) CHLTYPE(SVR) +
             TRPTYPE(LU62)  +
             SEQWRAP(9999999) MAXMSGL(2048) +
             XMITQ('VM03NCM.TQ.RQSV.ICE') +
             CONNAME('$ICE.#IYAHT0C.IYAHT0C0.IYCNVM03')+
             MODENAME('LU62PS') TPNAME(MQSERIES) +
```

where `MQSeries` is the TPNAME the MVS queue manager is listening on.

# TCP/IP communications example

This example shows how to establish communications with a remote MQSeries system over TCP/IP.

## TCPConfig stanza in QMINI

The QMINI file must contain an appropriate `TCPConfig` stanza. For example:

```
TCPConfig:
   TCPPort=1414
   TCPNumListenerPorts=1
   TCPListenerPort=1996
   TCPKeepAlive=1
```

The `TCPPort` value is the default outbound port for channels without a port value in the CONNAME field. `TCPListenerPort` identifies the default port that is used if the –p option is not supplied when using **runmqlsr** on the command line.

## Defining a TCP/IP sender channel

A TCP/IP sender channel must be defined. In this example, the queue manager is MH01 on a host called SPINACH:

```
    DEFINE CHANNEL(MT01_MH01_SDRC_0001) CHLTYPE(SDR) +
            TRPTYPE(TCP) +
            SEQWRAP(9999999) MAXMSGL(4194304) +
            XMITQ('MH01_TQ_SDRC_0001') +
            CONNAME('SPINACH.HURSLEY.IBM.COM(2000)')
```

This channel would try to attach to a TCP/IP port number 2000 on the host SPINACH.

The following example shows a TCP/IP sender channel definition for a queue manager MH01 on the host SPINACH using the *default* outbound TCP/IP port:

```
    DEFINE CHANNEL(MT01_MH01_SDRC_0001) CHLTYPE(SDR) +
            TRPTYPE(TCP) +
            SEQWRAP(9999999) MAXMSGL(4194304) +
            XMITQ('MH01_TQ_SDRC_0001') +
            CONNAME('SPINACH.HURSLEY.IBM.COM')
```

No port number is specified in the CONNAME. Therefore, the value specified on the `TCPPort` entry in the QMINI file (1414) is used.

## Defining a TCP/IP receiver channel

An example TCP/IP receiver channel:

```
    DEFINE CHANNEL(MH01_MT01_SDRC_0001) CHLTYPE(RCVR) +
            TRPTYPE(TCP)
```

A TCP/IP receiver channel requires no CONNAME value, but a TCP/IP listener must be running. There are two ways of starting a TCP/IP listener. Either:

1. Go into the queue manager’s PATHWAY using PATHCOM, and enter:

```
start server mqs-tcplis00
```

or

From the TACL prompt, enter:

```
runmqlsr -m QMgrName
```

A TCP/IP listener, which will listen on the port defined in the QMINI file (in this example, 1996), is started.

**Note:** This port number can be overridden by the -p *Port* flag on **runmqlsr**.

## Defining a TCP/IP sender channel on the remote system

The sender channel definition on the remote system to connect to this receiver channel could look like:

```
   DEFINE CHANNEL(MH01_MT01_SDRC_0001) CHLTYPE(SDR) +
            TRPTYPE(TCP) +
            XMITQ('MT01_TQ_SDRC_0001') +
            CONNAME('Compaq.ISC.UK.IBM.COM(1996)')
```

## Configuring QMINI to support multiple TCP/IP listeners

To enable a queue manager to support multiple TCP/IP listeners, you must create a new PATHWAY server class for each additional listener, based on MQS-TCPLIS00.

In addition, each TCP/IP listener must have its own listener port entry in the `TCPConfig` stanza of the QMINI file.

For example:

```
TCPConfig:
 TCPPort=1414
 TCPNumListenerPorts=3
 TCPListenerPort=1996
 TCPListenerPort=1997
 TCPListenerPort=1998
 TCPKeepAlive=1
```

`TCPNumListenerPorts` must match the number of `TCPListenerPort` entries (three in this example). This QMINI file is capable of supporting three TCP/IP listeners listening on ports 1996, 1997 and 1998. Typically, the server classes to support these three ports would be named MQS-TCPLIS00, MQS-TCPLIS01, and MQS-TCPLIS02.

For more information about adding server classes, see “TS/MP (PATHWAY) administration” on page 29.

# Appendix N. MQSeries clients

An MQSeries client is an MQSeries system that does not include a queue manager. The MQSeries client code directs MQI calls from applications running on the client system to a queue manager on an MQSeries server system to which it is connected.

This appendix provides information about MQSeries clients that is specific to MQSeries for Compaq NSK V5.1. It should be used in conjunction with the *MQSeries Clients* book.

## Client support

MQSeries for Compaq NSK can function as an MQSeries server system to all MQSeries clients that can connect to the server using TCP/IP or SNA LU 6.2 protocols. However, there is no MQSeries for Compaq NSK client.

When an MQSeries client connects to a queue manager on MQSeries for Compaq NSK:

- Any MQGET, MQPUT, or MQPUT1 with an MQ*_SYNCPOINT option initiates a Compaq transaction, if one has not already been associated with the connection handle.
- Any MQGET, MQPUT, or MQPUT1 with neither an MQ*_SYNCPOINT nor an MQ*_NO_SYNCPOINT option initiates a Compaq transaction, if one has not already been associated with the connection handle.
- The MQCMIT call commits a Compaq transaction, if one is associated with the connection handle. The MQBACK call cancels the Compaq transaction, if one is associated with the connection handle.

In all cases, if the Compaq BEGINTRANSACTION fails, a *CompCode* of MQCC_FAILED, and a *Reason* of MQRC_SYNCPOINT_NOT_AVAILABLE are returned to the caller.

### Security considerations

MQSeries for Compaq NSK supports the use of channel security exits for the validation of clients, as follows:

- After a connection is established between the MQSeries client and the server, the client invokes the security exit on the server prior to returning from the MQCONN call.
- The server security exit can return information to the client security exit.

This dialog allows, for example, the communication of confidential data between the server and client. If the client has not defined a security exit, the values of the local environment variables MQ_USER_ID and MQ_PASSWORD are passed to the server via channel attributes. These attributes are available to the server security exit for validation.

© Copyright IBM Corp. 1993, 2001

# Appendix O. Programmable System Management

MQSeries for Compaq NSK supports these system-management functions of MQSeries:
- Instrumentation events
- Programmable Command Formats (PCFs)
- Installable services

This appendix provides a summary of these functions in MQSeries for Compaq NSK. For detailed descriptions, see the *MQSeries Programmable System Management* book.

## Instrumentation events

MQSeries for Compaq NSK supports the standard MQSeries instrumentation events, which result in the generation of an event message on an event queue.

You enable and disable events by specifying appropriate values for queue and queue manager attributes using:
- MQSC, as described in the *MQSeries MQSC Command Reference* book
- PCF commands, as described in the *MQSeries Programmable System Management* book
- Message Queue Management (MQM), as described in "Chapter 4. Managing queue managers" on page 41

### Event types supported by MQSeries for Compaq NSK

MQSeries for Compaq NSK supports the following event types:

*Table 27. Event types supported by MQSeries for Compaq NSK*

| Event type | Event name |
|---|---|
| Authority events | Not Authorized (type 1) |
| Channel events | Channel Activated<br>Channel Conversion Error<br>Channel Not Activated<br>Channel Started<br>Channel Stopped |
| Inhibit events | Get Inhibited<br>Put Inhibited |
| Local events | Alias Base Queue Type Error<br>Queue Type Error<br>Unknown Alias Base Queue<br>Unknown Object Name |
| Performance events | Queue Depth High<br>Queue Depth Low<br>Queue Full<br>Queue Service Interval High<br>Queue Service Interval OK |

© Copyright IBM Corp. 1993, 2001

Table 27. Event types supported by MQSeries for Compaq NSK (continued)

| Event type | Event name |
|---|---|
| Remote events | Default Transmission Queue Type Error<br>Default Transmission Queue Usage Error<br>Queue Type Error<br>Remote Queue Name Error<br>Transmission Queue Usage Error<br>Unknown Default Transmission Queue<br>Unknown Remote Queue Manager<br>Unknown Transmission Queue |
| Start and stop events | Queue Manager Active<br>Queue Manager Not Active |

## Event-message format

MQSeries for Compaq NSK supports the standard MQSeries event-message format. That is, the event message has two parts, the *message descriptor* (MQMD) and the *message data*. The message data comprises an event header and some data that is specific to the type of event.

The MQMD structure of an event message is summarized in “MQMD – Message Descriptor” on page 317. The event header structure (MQCFH) is summarized in Table 29 on page 373.

Table 28. MQMD structure of an event message

| Parameter | Type | Values |
|---|---|---|
| *StrucId* | MQCHAR4 | MQMD_STRUC_ID |
| *Version* | MQLONG | MQMD_VERSION_1 |
| *Report* | MQLONG | MQRO_NONE |
| *MsgType* | MQLONG | MQMT_DATAGRAM |
| *Expiry* | MQLONG | MQEI_UNLIMITED |
| *Feedback* | MQLONG | MQFB_NONE |
| *Encoding* | MQLONG | Encoding of the queue manager generating the event. |
| *CodedCharSetId* | MQLONG | Coded character set ID (CCSID) of the queue manager generating the event. |
| *Format* | MQCHAR8 | MQFMT_EVENT |
| *Priority* | MQLONG | Default priority of the event queue, if it is a local queue, or its local definition at the queue manager generating the event. |
| *Persistence* | MQLONG | Default persistence of the event queue, if it is a local queue, or its local definition at the queue manager generating the event. |
| *MsgId* | MQBYTE24 | The value is uniquely generated by the queue manager. |
| *CorrelId* | MQBYTE24 | MQCI_NONE |
| *BackoutCount* | MQLONG | The value is always 0. |
| *ReplyToQ* | MQCHAR48 | Always blank. |
| *ReplyToQMgr* | MQCHAR48 | The queue manager name at the originating system. |

*Table 28. MQMD structure of an event message (continued)*

| Parameter | Type | Values |
|---|---|---|
| *UserIdentifier* | MQCHAR12 | Always blank. |
| *AccountingToken* | MQBYTE32 | MQACT_NONE |
| *ApplIdentityData* | MQCHAR32 | Always blank. |
| *PutApplType* | MQLONG | Type of application that put the message. |
| *PutApplName* | MQCHAR28 | Name of the application that put the message. |
| *PutDate* | MQCHAR8 | Date when the message was put, generated by the queue manager. |
| *PutTime* | MQCHAR8 | Time when message was put, generated by the queue manager. |
| *ApplOriginData* | MQCHAR4 | Always blank. |

*Table 29. Event header structure (MQCFH)*

| Parameter | Type | Values |
|---|---|---|
| *Type* | MQLONG | MQCFT_EVENT |
| *StrucLength* | MQLONG | MQCFH_STRUC_LENGTH |
| *Version* | MQLONG | MQCFH_VERSION_1 |
| *Command* | MQLONG | MQCMD_Q_MGR_EVENT<br>MQCMD_PERFM_EVENT<br>MQCMD_CHANNEL_EVENT |
| *MsqSeqNumber* | MQLONG | Always 1. |
| *Control* | MQLONG | MQCFC_LAST |
| *CompCode* | MQLONG | MQCC_OK MQCC_WARNING |
| *Reason* | MQLONG | Reason code identifying event. |
| *ParameterCount* | MQLONG | The number of parameter structures that follow the MQCFH structure. |

# Programmable command formats (PCFs)

MQSeries for Compaq NSK supports the standard Programmable Command Format (PCF) functions, as described in the *MQSeries Programmable System Management* book. PCF messages are made up of two parts, the *message descriptor* (MQMD) and the *message data*. The message data comprises a PCF header (MQCFH) and some PCF parameters defined by the structures MQCFIN, MQCFIL, MQCFST, and MQCFSL.

The PCF message descriptor (MQMD) is summarized in Table 30 on page 374. The PCF header structure (MQCFH) is summarized in Table 31 on page 374. The PCF parameter structures are summarized in Table 32 on page 374 through Table 35 on page 375.

## PCF message descriptor

For MQSeries for Compaq NSK, the standard PCF message descriptor applies. That is, the message descriptor contains these fields:

Table 30. PCF message descriptor

| Field | Values |
|---|---|
| *Report* | Any valid value |
| *MsgType* | MQMT_REQUEST |
| *Expiry* | Any valid value |
| *Feedback* | MQFB_NONE |
| *Encoding* | Encoding used for the message data; conversion is performed if necessary. |
| *CodedCharSetId* | CCSID used for the message data; conversion is performed if necessary. |
| *Format* | MQFMT_ADMIN MQFMT_PCF (for user data) |
| *Priority* | Any valid value |
| *Persistence* | Any valid value |
| *MsgId* | Any valid value, including MQMI_NONE |
| *CorrelId* | Any valid value, including MQMI_NONE |
| *ReplyToQ* | Queue name |
| *ReplyToQMgr* | Queue manager name |
| Message context fields | Any valid value, including MQPMO_DEFAULT_CONTEXT |

## PCF header (MQCFH)

For MQSeries for Compaq NSK, the standard PCF header applies. That is, the PCF header structure contains these fields:

Table 31. PCF header

| Field | Type | Values |
|---|---|---|
| *Type* | MQLONG | MQCFT_COMMAND MQCFT_RESPONSE MQCFT_EVENT |
| *StrucLength* | MQLONG | MQCFH_STRUC_LENGTH |
| *Version* | MQLONG | MQCFH_VERSION_1 |
| *Command* | MQLONG | Valid command identifier. |
| *MsgSeqNumber* | MQLONG | Sequence number of the message. |
| *Control* | MQLONG | MQCFC_LAST MQCFC_NOT_LAST |
| *CompCode* | MQLONG | MQCC_OK MQCC_WARNING MQCC_FAILED MQCC_UNKNOWN |
| *Reason* | MQLONG | Reason code qualifying the completion code. |
| *ParameterCount* | MQLONG | Count of parameter structures. |

## PCF string parameter (MQCFST)

For MQSeries for Compaq NSK, the standard PCF string parameter structure (MQCFST) applies. That is, the PCF string parameter structure contains these fields:

Table 32. PCF string parameter

| Field | Type | Value |
|---|---|---|
| *Type* | MQLONG | MQCFT_STRING |

Table 32. PCF string parameter (continued)

| Field | Type | Value |
|---|---|---|
| *StrucLength* | MQLONG | Length in bytes of the MQCFST structure. |
| *Parameter* | MQLONG | Parameter identifier. |
| *CodedCharSetId* | MQLONG | Coded character set identifier (CCSID). |
| *StringLength* | MQLONG | Length in bytes of the data in the *String* field. |
| *String* | MQCHAR × *StringLength* | String value. |

## PCF integer list parameter (MQCFIL)

For MQSeries for Compaq NSK, the standard PCF integer list parameter structure (MQCFIL) applies. That is, the PCF integer list parameter structure contains these fields:

Table 33. PCF integer list

| Field | Type | Value |
|---|---|---|
| *Type* | MQLONG | MQCFT_INTEGER_LIST |
| *StrucLength* | MQLONG | Length in bytes of the MQCFIL structure. |
| *Parameter* | MQLONG | Parameter identifier. |
| *Count* | MQLONG | Number of elements in the *Values* array. |
| *Values* | MQLONG × *Count* | Parameter values. |

## PCF integer (MQCFIN)

For MQSeries for Compaq NSK, the standard PCF integer structure (MQCFIN) applies. That is, the PCF integer structure contains these fields:

Table 34. PCF integer

| Field | Type | Value |
|---|---|---|
| *Type* | MQLONG | MQCFT_INTEGER |
| *StrucLength* | MQLONG | MQCFIN_STRUC_LENGTH |
| *Parameter* | MQLONG | Parameter identifier |
| *Value* | MQLONG | Parameter value |

## PCF string list (MQCFSL)

For MQSeries for Compaq NSK, the standard PCF string list structure (MQCFSL) applies. That is, the PCF string list structure contains these fields:

Table 35. PCF string list

| Field | Type | Value |
|---|---|---|
| *Type* | MQLONG | MQCFT_STRING_LIST |
| *StrucLength* | MQLONG | Length in bytes of the MQCFSL structure |
| *Parameter* | MQLONG | Parameter identifier |
| *CodedCharSetId* | MQLONG | CCSID of the data in the *Strings* field. |

Table 35. PCF string list (continued)

| Field | Type | Value |
|---|---|---|
| *Count* | MQLONG | Number of strings in the *Strings* field. |
| *StringLength* | MQLONG | Length in bytes of each string in the *Strings* field. |
| *Strings* | MQCHAR × *StringLength* × *Count* | Set of string values for the parameter identified by the *Parameter* field. |

## PCF commands supported by MQSeries for Compaq NSK

The following MQSeries PCF commands are supported by MQSeries for Compaq NSK. For a complete description of these commands, see the *MQSeries Programmable System Management* book.

Table 36. PCF commands supported by MQSeries for Compaq NSK

| Command | Command identifier |
|---|---|
| Change Channel | MQCMD_CHANGE_CHANNEL |
| Change Namelist | MQCMD_CHANGE_NAMELIST |
| Change Process | MQCMD_CHANGE_PROCESS |
| Change Queue | MQCMD_CHANGE_Q |
| Change Queue Manager | MQCMD_CHANGE_Q_MGR |
| Clear Queue | MQCMD_CLEAR_Q |
| Copy Channel | MQCMD_COPY_CHANNEL |
| Copy Namelist | MQCMD_COPY_NAMELIST |
| Copy Process | MQCQ_COPY_PROCESS |
| Copy Queue | MQCMD_COPY_Q |
| Create Channel | MQCMD_CREATE_CHANNEL |
| Create Namelist | MQCMD_CREATE_NAMELIST |
| Create Process | MQCMD_CREATE_PROCESS |
| Create Queue | MQCMD_CREATE_Q |
| Delete Channel | MQCMD_DELETE_CHANNEL |
| Delete Namelist | MQCMD_DELETE_NAMELIST |
| Delete Process | MQCMD_DELETE_PROCESS |
| Delete Queue | MQCMD_DELETE_Q |
| Escape | MQCMD_ESCAPE |
| Inquire Channel | MQCMD_INQUIRE_CHANNEL |
| Inquire Channel Names | MQCMD_INQUIRE_CHANNEL_NAMES |
| Inquire Channel Status | MQCMD_INQUIRE_CHANNEL_STATUS |
| Inquire Cluster Queue Manager | MQCMD_INQUIRE_CLUSTER_Q_MGR |
| Inquire Namelist | MQCMD_INQUIRE_NAMELIST |
| Inquire Process | MQCMD_INQUIRE_PROCESS |
| Inquire Process Names | MQCMD_INQUIRE_PROCESS_NAMES |
| Inquire Queue | MQCMD_INQUIRE_Q |
| Inquire Queue Manager | MQCMD_INQUIRE_Q_MGR |

*Table 36. PCF commands supported by MQSeries for Compaq NSK (continued)*

| Command | Command identifier |
|---|---|
| Inquire Queue Names | MQCMD_INQUIRE_Q_NAMES |
| Ping Channel | MQCMD_PING_CHANNEL |
| Ping Queue Manager | MQCMD_PING_Q_MGR |
| Refresh Cluster | MQCMD_REFRESH_CLUSTER |
| Reset Channel | MQCMD_RESET_CHANNEL |
| Reset Queue Statistics | MQCMD_RESET_Q_STATS |
| Resolve Channel | MQCMD_RESOLVE_CHANNEL |
| Resume Queue Manager Cluster | MQCMD_RESUME_Q_MGR_CLUSTER |
| Start Channel | MQCMD_START_CHANNEL |
| Stop Channel | MQCMD_STOP_CHANNEL |
| Suspend Queue Manager Cluster | MQCMD_SUSPEND_Q_MGR_CLUSTER |

**Note:** MQSeries for Compaq NSK does not support the Start Channel Initiator and Start Channel Listener commands.

## PCF command responses

In MQSeries for Compaq NSK, the command server generates standard response messages to each PCF command. There are three types of response:

- OK response
- Error response
- Data response

For more information, see the *MQSeries Programmable System Management* book.

# Installable services

MQSeries for Compaq NSK supports the authorization service and the name service.

## Authorization service interface

The authorization service enables queue managers to invoke authorization facilities. For example, a queue manager can check that a particular user ID is authorized to open a queue using the authorization service.

An authorization service component is supplied with MQSeries for Compaq NSK. This component is called the Object Authority Manager (OAM). By default, the OAM is active and works with the control commands **dspmqaut** (display authority) and **setmqaut** (set authority).

You can augment or replace the OAM with your own authorization service component, as described in the *MQSeries Programmable System Management* book.

## Name service interface

The name service provides support to the queue manager for resolving the name of the queue manager that owns a queue.

## Installable services

The standard name service interface, as described in the *MQSeries Programmable System Management* book, is supported by MQSeries for Compaq NSK.

# Appendix P. EMS event template used by MQSeries for Compaq NSK

The EMS template file (SMQSTMPL) contains the source code for the definitions of MQSeries EMS events. These definitions control how the information in the EMS event messages is displayed, and also show the type and meaning of the data contained in each EMS Event message.

The following types of event are generated:

**ZMQS-VAL-EVT-ERROR**
: An FFST (a system resource problem, a software problem, or a hardware problem).

**ZMQS-VAL-EVT-ERR**
: An error with MQSeries, referencing an FFST event and logged data on disk.

**ZMQS-VAL-EVT-MSG**
: An MQSeries message, such as the starting of a queue manager or channel. All of these events correspond to an MQSeries AMQxxxxx log message and contain the same information and text. The variable data in each message is contained in individual tokens within the event message. For more information about the AMQxxxxx messages, see the *MQSeries Messages* book.

**ZMQS-VAL-EVT-QMGR**
: A queue manager event for authority, inhibit, local, remote, start, and stop events. These EMS events have effectively the same information content as their corresponding PCF event messages, which are described in the *MQSeries Programmable System Management* book. Individual tokens in the event message contain the variable data in each event message.

**ZMQS-VAL-EVT-PERF**
: A performance event, corresponding with the standard MQSeries performance events. These events report statistical data about queues within a queue manager. The variable data in performance events is contained in individual tokens within the event message.

**ZMQS-VAL-EVT-CHNL**
: A channel event, corresponding with the standard MQSeries channel events. Channel events report changes in status of channels, or problems in communication between queue managers. As with the other event message types, the variable data in channel events is contained in individual tokens within the event message.

Here is an extract from the definitions of the EMS templates:

```
VERSION: "IBM.MQS - 10JAN97"
SSID: ZMQS-VAL-SSID
SSNAME: "MQSeries", "MQS"

==
== This is an EMS FFST  message
==
MSG: ZEMS-TKN-EVENTNUMBER, ZMQS-VAL-EVT-ERROR
     OVERRIDE ZEMS-TKN-EMPHASIS ZSPI-VAL-TRUE
     "MQSeries FFST from component COMP_<1>                 "
```

© Copyright IBM Corp. 1993, 2001

```
     "<*CR>   Error Code : <2>                               "
     "<*CR>   Severity : <3>                                 "
     "<*CR>   Module Name : <4>                              "
     "<*CR>   Probe ID : <5>                                 "
     "<*CR>   Error Text :                                   "
     "<*CR>   <6>"
     1: ZMQS-TKN-COMPONENT
     2: ZMQS-TKN-ERROR-CODE

     3: ZMQS-TKN-SEVERITY
     4: ZMQS-TKN-MODULE-NAME
     5: ZMQS-TKN-PROBE-ID
     6: ZMQS-TKN-ERROR-TEXT


==
== This is an EMS Display Message Event
==
MSG: ZEMS-TKN-EVENTNUMBER, ZMQS-VAL-EVT-MSG
     "MQSeries message: <1>                                  "
     "                                                       "
     "<*CR>   EXPLANATION :                                  "
         "<*CR>   <2>                                            "
     "                                                       "
     "<*CR>   ACTION :                                       "
         "<*CR>   <3>"
     1: ZMQS-TKN-ERROR-TEXT
     2: ZMQS-TKN-ERROR-TEXT-2
     3: ZMQS-TKN-ERROR-TEXT-3


==
== This is an EMS Report Error Event
==
MSG: ZEMS-TKN-EVENTNUMBER, ZMQS-VAL-EVT-ERR
     OVERRIDE ZEMS-TKN-EMPHASIS ZSPI-VAL-TRUE
     "MQSeries Error                                         "
     "<*CR>   Error Code : <1>                               "
     "<*CR>   Function : <2>                                 "
     "<*CR>   Probe ID : <3>                                 "
     "<*CR>   FFST File : <4>                                "
     1: ZMQS-TKN-ERROR-CODE
     2: ZMQS-TKN-MODULE-NAME
     3: ZMQS-TKN-PROBE-ID
     4: ZMQS-TKN-FILE-NAME


==
== This is an EMS copy of PCF Queue Manager event message
== for authority, inhibit, local, remote, start_and_stop events
==
MSG: ZEMS-TKN-EVENTNUMBER, ZMQS-VAL-EVT-QMGR
     "MQSeries QMgr Event from  <1>                          "
     "<*CR>  Reason : <2>                                    "
     "<*IF 3><*CR>  Reason Qualifier : <4>                    <*ENDIF>"
     "<*IF 5><*CR>  User ID : <6>                             <*ENDIF>"
     "<*IF 13><*CR>  Object QMgr : <14>                        <*ENDIF>"
     "<*IF 9><*CR>  Options : <10>                             <*ENDIF>"
     "<*IF 11><*CR>  Command : <12>                            <*ENDIF>"
     "<*IF 15><*CR>  Queue Name : <16>                         <*ENDIF>"
     "<*IF 17><*CR>  Queue Type : <18>                         <*ENDIF>"
     "<*IF 19><*CR>  Base Queue Name : <20>                    <*ENDIF>"
     "<*IF 21><*CR>  XMit Queue Name : <22>                    <*ENDIF>"
     "<*IF 30><*CR>  Application Type : <31>                   <*ENDIF>"
     "<*IF 32><*CR>  Application Name : <33>                   <*ENDIF>"
     1: ZMQS-TKN-QMGR
     2: ZMQS-TKN-REASON
     3: TOKENPRESENT(ZMQS-TKN-REASON-QUALIFIER)
     4: ZMQS-TKN-REASON-QUALIFIER
     5: TOKENPRESENT(ZMQS-TKN-USER-ID)
     6: ZMQS-TKN-USER-ID
     9: TOKENPRESENT(ZMQS-TKN-OPTIONS)
    10: ZMQS-TKN-OPTIONS
    11: TOKENPRESENT(ZMQS-TKN-COMMAND)
    12: ZMQS-TKN-COMMAND
    13: TOKENPRESENT(ZMQS-TKN-OBJ-QMGR)
    14: ZMQS-TKN-OBJ-QMGR
    15: TOKENPRESENT(ZMQS-TKN-Q-NAME)
    16: ZMQS-TKN-Q-NAME
    17: TOKENPRESENT(ZMQS-TKN-Q-TYPE)
    18: ZMQS-TKN-Q-TYPE
    19: TOKENPRESENT(ZMQS-TKN-BASE-Q-NAME)
    20: ZMQS-TKN-BASE-Q-NAME
    21: TOKENPRESENT(ZMQS-TKN-XMIT-Q-NAME)
    22: ZMQS-TKN-XMIT-Q-NAME
    30: TOKENPRESENT(ZMQS-TKN-APPL-TYPE)
    31: ZMQS-TKN-APPL-TYPE
    32: TOKENPRESENT(ZMQS-TKN-APPL-NAME)
    33: ZMQS-TKN-APPL-NAME


==
== This is an EMS copy of PCF Performance event message
==
MSG: ZEMS-TKN-EVENTNUMBER, ZMQS-VAL-EVT-PERF
     "MQSeries Performance Event from <1>               "
     "<*CR>  Reason : <2>                                    "
     "<*CR>  Queue Name : <3>                                "
     "<*CR>  Time Since Last Reset  : <4>                    "
     "<*CR>  Highest Queue Depth    : <5>                    "
     "<*CR>  # Of Messages Enqueued : <6>                    "
     "<*CR>  # Of Messages Dequeued : <7>                    "
     1: ZMQS-TKN-QMGR
     2: ZMQS-TKN-REASON
     3: ZMQS-TKN-Q-NAME
     4: ZMQS-TKN-TIME-SINCE-RESET
     5: ZMQS-TKN-HIGH-Q-DEPTH
     6: ZMQS-TKN-MSG-ENQ-COUNT
     7: ZMQS-TKN-MSG-DEQ-COUNT


==
== This is an EMS copy of PCF Channel event message
==
MSG: ZEMS-TKN-EVENTNUMBER, ZMQS-VAL-EVT-CHNL
     "MQSeries Channel Event from <1>                   "
     "<*CR>  Reason : <2>                                    "
     "<*CR>  Channel Name : <3>                              "
     "<*CR>  XMit Queue Name : <5>                           "
     "<*CR>  Connection Name : <7>                           "
     "<*CR>  Reason Qualifier : <9>                          "
     "<*CR>  Format : <11>                                   "
     "<*CR>  Return Code : <13>                              "
     "<*CR>  Auxiliary rc 1 : <15>                           "
     "<*CR>  Auxiliary rc 2 : <17>                           "
     "<*CR>  CCSID 1 : <19>                                  "
     "<*CR>  Auxiliary string 1 : <21>                       "
     "<*CR>  CCSID 2 : <23>                                  "
     "<*CR>  Auxiliary string 2 : <25>                       "
     "<*CR>  CCSID 3 : <27>                                  "
     "<*CR>  Auxiliary string 3 : <29>                       "
     1: ZMQS-TKN-QMGR
     2: ZMQS-TKN-REASON
     3: ZMQS-TKN-CHANNEL-NAME
```

```
 5: ZMQS-TKN-XMIT-Q-NAME
 7: ZMQS-TKN-CONN-NAME
 9: ZMQS-TKN-REASON-QUALIFIER
11: ZMQS-TKN-FORMAT
13: ZMQS-TKN-RETURN-CODE
15: ZMQS-TKN-RETURN-CODE-2
17: ZMQS-TKN-RETURN-CODE-3
19: ZMQS-TKN-CCSID
21: ZMQS-TKN-ERROR-TEXT
23: ZMQS-TKN-CCSID-2
25: ZMQS-TKN-ERROR-TEXT-2
27: ZMQS-TKN-CCSID-3
29: ZMQS-TKN-ERROR-TEXT-3
```

# Appendix Q. 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.

© Copyright IBM Corp. 1993, 2001

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.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

## Trademarks

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

AIX
MQSeries
MVS/ESA
CICS
First Failure Support Technology
OS/390
IBMLink
SupportPac

IBM
AS/400
FFST
OS/2
VSE/ESA
BookManager
MVS
VTAM

Compaq and NonStop are trademarks of Compaq Computer Corporation.

Intel is a registered trademark of Intel Corporation in the United States, other countries, or both.

Microsoft, Windows and Windows NT are trademarks of Microsoft Corporation.

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.

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

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

# Bibliography

This section describes the documentation available for all current MQSeries products.

## MQSeries cross-platform publications

Most of these publications, which are sometimes referred to as the MQSeries "family" books, apply to all MQSeries products. The latest MQSeries products are:

- MQSeries for AIX, V5.2
- MQSeries for AS/400, V5.2
- MQSeries for AT&T GIS UNIX, V2.2
- MQSeries for Compaq OpenVMS Alpha, V5.1
- MQSeries for Compaq Tru64 UNIX, V5.1
- MQSeries for HP-UX, V5.2
- MQSeries for Linux, V5.2
- MQSeries for OS/2 Warp, V5.1
- MQSeries for OS/390, V5.2
- MQSeries for SINIX and DC/OSx, V2.2
- MQSeries for Sun Solaris, V5.2
- MQSeries for Sun Solaris, Intel Platform Edition, V5.1
- MQSeries for Compaq NSK, V5.1
- MQSeries for VSE/ESA, V2.1.1
- MQSeries for Windows, V2.0
- MQSeries for Windows, V2.1
- MQSeries for Windows NT and Windows 2000, V5.2

The MQSeries cross-platform publications are:

- *MQSeries Brochure*, G511-1908
- *An Introduction to Messaging and Queuing*, GC33-0805
- *MQSeries Intercommunication*, SC33-1872
- *MQSeries Queue Manager Clusters*, SC34-5349
- *MQSeries Clients*, GC33-1632
- *MQSeries System Administration*, SC33-1873
- *MQSeries MQSC Command Reference*, SC33-1369
- *MQSeries Event Monitoring*, SC34-5760
- *MQSeries Programmable System Management*, SC33-1482
- *MQSeries Administration Interface Programming Guide and Reference*, SC34-5390
- *MQSeries Messages*, GC33-1876
- *MQSeries Application Programming Guide*, SC33-0807
- *MQSeries Application Programming Reference*, SC33-1673
- *MQSeries Programming Interfaces Reference Summary*, SX33-6095
- *MQSeries Using C++*, SC33-1877
- *MQSeries Using Java*, SC34-5456
- *MQSeries Application Messaging Interface*, SC34-5604

## MQSeries platform-specific publications

Each MQSeries product is documented in at least one platform-specific publication, in addition to the MQSeries family books.

**MQSeries for AIX, V5.2**

*MQSeries for AIX Quick Beginnings*, GC33-1867

**MQSeries for AS/400, V5.2**

*MQSeries for AS/400® Quick Beginnings*, GC34-5557

*MQSeries for AS/400 System Administration*, SC34-5558

*MQSeries for AS/400 Application Programming Reference (ILE RPG)*, SC34-5559

**MQSeries for AT&T GIS UNIX, V2.2**

*MQSeries for AT&T GIS UNIX® System Management Guide*, SC33-1642

**MQSeries for Compaq OpenVMS Alpha, V5.1**

*MQSeries for Compaq OpenVMS Alpha Quick Beginnings*, GC34-5885

*MQSeries for Compaq OpenVMS Alpha System Administration Guide*, SC34-5884

**MQSeries for Compaq NSK, V5.1**

*MQSeries for Compaq NSK Quick Beginnings*, GC34-5887

*MQSeries for Compaq NSK System Administration Guide*, SC34-5886

**MQSeries for Compaq Tru64 UNIX, V5.1**

*MQSeries for Compaq Tru64 UNIX Quick Beginnings*, GC34-5684

**MQSeries for HP-UX, V5.2**

© Copyright IBM Corp. 1993, 2001

*MQSeries for HP-UX Quick Beginnings*, GC33-1869

**MQSeries for Linux, V5.2**

*MQSeries for Linux Quick Beginnings*, GC34-5691

**MQSeries for OS/2 Warp, V5.1**

*MQSeries for OS/2 Warp Quick Beginnings*, GC33-1868

**MQSeries for OS/390, V5.2**

*MQSeries for OS/390® Concepts and Planning Guide*, GC34-5650

*MQSeries for OS/390 System Setup Guide*, SC34-5651

*MQSeries for OS/390 System Administration Guide*, SC34-5652

*MQSeries for OS/390 System Administration Guide*, GC34-5892

*MQSeries for OS/390 Messages and Codes*, GC34-5891

*MQSeries for OS/390 Licensed Program Specifications*, GC34-5893

*MQSeries for OS/390 Program Directory*

**MQSeries link for R/3, Version 1.2**

*MQSeries link for R/3 User's Guide*, GC33-1934

**MQSeries for SINIX and DC/OSx, V2.2**

*MQSeries for SINIX and DC/OSx System Management Guide*, GC33-1768

**MQSeries for Sun Solaris, V5.2**

*MQSeries for Sun Solaris Quick Beginnings*, GC33-1870

**MQSeries for Sun Solaris, Intel Platform Edition, V5.1**

*MQSeries for Sun Solaris, Intel Platform Edition Quick Beginnings*, GC34-5851

**MQSeries for VSE/ESA, V2.1.1**

*MQSeries for VSE/ESA™ Licensed Program Specifications*, GC34-5365

*MQSeries for VSE/ESA System Management Guide*, GC34-5364

**MQSeries for Windows, V2.0**

*MQSeries for Windows® User's Guide*, GC33-1822

**MQSeries for Windows, V2.1**

*MQSeries for Windows User's Guide*, GC33-1965

**MQSeries for Windows NT and Windows 2000, V5.2**

*MQSeries for Windows NT and Windows 2000 Quick Beginnings*, GC34-5389

*MQSeries for Windows NT® Using the Component Object Model Interface*, SC34-5387

*MQSeries LotusScript Extension*, SC34-5404

## Softcopy books

Most of the MQSeries books are supplied in both hardcopy and softcopy formats.

### HTML format

Relevant MQSeries documentation is provided in HTML format with these MQSeries products:
- MQSeries for AIX, V5.2
- MQSeries for AS/400, V5.2
- MQSeries for Compaq OpenVMS Alpha, V5.1
- MQSeries for Compaq Tru64 UNIX, V5.1
- MQSeries for HP-UX, V5.2
- MQSeries for Linux, V5.2
- MQSeries for OS/2 Warp, V5.1
- MQSeries for OS/390, V5.2
- MQSeries for Sun Solaris, V5.2
- MQSeries for Sun Solaris, Intel Platform Edition, V5.1
- MQSeries for Windows NT and Windows 2000, V5.2 (compiled HTML)
- MQSeries link for R/3, V1.2

The MQSeries books are also available in HTML format from the MQSeries product family Web site at:

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

### Portable Document Format (PDF)

PDF files can be viewed and printed using the Adobe Acrobat Reader.

If you need to obtain the Adobe Acrobat Reader, or would like up-to-date information about the platforms on which the Acrobat Reader is supported, visit the Adobe Systems Inc. Web site at:

```
http://www.adobe.com/
```

PDF versions of relevant MQSeries books are supplied with these MQSeries products:
- MQSeries for AIX, V5.2
- MQSeries for AS/400, V5.2
- MQSeries for Compaq OpenVMS Alpha, V5.1

- MQSeries for Compaq Tru64 UNIX, V5.1
- MQSeries for HP-UX, V5.2
- MQSeries for Linux, V5.2
- MQSeries for OS/2 Warp, V5.1
- MQSeries for OS/390, V5.2
- MQSeries for Sun Solaris, V5.2
- MQSeries for Sun Solaris, Intel Platform Edition, V5.1
- MQSeries for Windows NT and Windows 2000, V5.2
- MQSeries link for R/3, V1.2

PDF versions of all current MQSeries books are also available from the MQSeries product family Web site at:

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

## BookManager® format

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

## PostScript format

The MQSeries library is provided in PostScript (.PS) format with many MQSeries Version 2 products. Books in PostScript format can be printed on a PostScript printer or viewed with a suitable viewer.

## Windows Help format

The *MQSeries for Windows User's Guide* is provided in Windows Help format with MQSeries for Windows, Version 2.0 and MQSeries for Windows, Version 2.1.

## MQSeries information available on the Internet

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 an MQSeries SupportPac™.

## Related publications

- *SNAX/APC Planning and Configuration Manual*, (Compaq Part No. 098289)

  SNAX/APC provides LU 6.2 support for the Compaq implementation of SNA. This guide explains how to install and configure SNAX/APC.
- *SCF Reference Manual for SNAX/APC*, (Compaq Part No. 064525)

  SNAX/APC provides LU 6.2 support for the Compaq implementation of SNA. This guide explains the Subsystem Control Facility (SCF) interactive interface that lets operators and network managers configure and control SNAX/APC.
- *Pathway System Management Guide*, (Compaq Part No. 096881)

  This guide presents guidelines for configuring and controlling Pathway transaction processing systems.
- *Introduction to NonStop Transaction Manager/MP (TM/MP)*, (Compaq Part No. 085812)

  This guide describes how to use the TMF subsystem to protect your business transactions and the integrity of your databases.
- *Introduction to Compaq Networking and Data Communications*, (Compaq Part No. 093148)

  This guide provides an overview of Compaq networking and data communications concepts, tasks, products, and manuals.
- *Intersystem Communications Environment (ICE) Installation Guide*, (Version 3 Release 2, or later edition)

  This guide describes how to install ICE and configure the ICE start-up parameters. (ICE provides LU 6.2 support for Insessions's implementation of SNA.)
- *Intersystem Communications Environment (ICE) Administrator's Guide*, (Version 3 Release 2, or later edition)

  This guide describes how to configure and operate ICE, its interfaces, and its utilities.

## Related publications

# Glossary of terms and abbreviations

This glossary defines MQSeries terms and abbreviations used in this book. If you cannot find a particular term, see the Index or the *IBM Dictionary of Computing*, New York: McGraw-Hill, 1994.

This glossary includes terms and definitions from the *American National Dictionary for Information Systems*, ANSI X3.172-1990, copyright 1990 by the American National Standards Institute (ANSI). Copies can be purchased from the American National Standards Institute, 11 West 42 Street, New York, New York 10036. Definitions are identified by the symbol (A) after the definition.

## A

**administration bag.** In the MQAI, a type of data bag that is created for administering MQSeries by implying that it can change the order of data items, create lists, and check selectors within a message.

**administration commands.** MQSeries commands used to manage MQSeries objects, such as queues, processes, and namelists.

**Advanced Program-to-Program Communication (APPC) .** The general facility characterizing the LU 6.2 architecture and its various implementations in products.

**alert.** A message sent to a management services focal point in a network to identify a problem or an impending problem.

**alias queue object.** An MQSeries object, the name of which is an alias for a base queue defined to the local queue manager. When an application or a queue manager uses an alias queue, the alias name is resolved and the requested operation is performed on the associated base queue.

**alternate user security.** A security feature in which the authority of one user ID can be used by another user ID, for example, to open an MQSeries object.

**APAR.** Authorized Program Analysis Report.

**APPC.** Advanced Program to Program Communication.

**application queue.** A queue used by an application.

**asynchronous messaging.** A method of communication between programs in which programs place messages on message queues. With asynchronous messaging, the sending program proceeds with its own processing without waiting for a reply to its message. Contrast with synchronous messaging.

**attribute.** One of a set of properties that defines the characteristics of an MQSeries object.

**authorization checks.** Security checks that are performed when you attempt to open an MQSeries object.

**authorization checks.** Security checks that are performed when a user tries to issue administration commands against an object, for example to open a queue or connect to a queue manager.

**authorization file.** A file that provides security definitions for an object, a class of objects, or all classes of object.

**authorized program analysis report (APAR).** A report of a problem caused by a suspected defect in a current, unaltered release of a program.

## B

**back out.** An operation that reverses all the changes made during the current unit of recovery or unit of work. After the operation is complete, a new unit of recovery or unit of work begins.

**bag.** See *data bag*.

**basic mapping support (BMS).** An interface between CICS and application programs that formats input and output display data and routes multiple-page output messages without regard for control characters used by various terminals.

**BMS.** Basic Mapping Support.

**browse.** In message queuing, to use the MQGET call to copy a message without removing it from the queue. See also *get*.

**browse cursor.** In message queuing, an indicator used when browsing a queue to identify the message that is next in sequence.

## C

**call back.** A requester message channel initiates a transfer from a sender channel by first calling the sender, then closing down and awaiting a call back.

**CCF.** Channel control function.

© Copyright IBM Corp. 1993, 2001

**CCSID.** Coded character set identifier.

**CDF.** Channel definition file.

**channel.** See *message channel*.

**channel control function (CCF).** A program to move messages from a transmission queue to a communication link, and from a communication link to a local queue, together with an operator panel interface to allow the setup and control of channels.

**channel definition file (CDF).** In MQSeries, a file containing communication channel definitions that associate transmission queues with communication links.

**channel event.** An event that indicates that a channel instance has become available or unavailable. Channel events are generated on the queue managers at both ends of the channel.

**checkpoint.** A time when significant information is written on the log. Contrast with *syncpoint*. In MQSeries on UNIX systems, the point in time when a data record described in the log is the same as the data record in the queue. Checkpoints are generated automatically and are used during the system restart process.

**CICS.** Customer Information Control System.

**client.** A run-time component that provides access to queuing services on a server for local user applications. The queues used by the applications reside on the server. See also *MQI client*.

**client application.** An application running on a workstation and linked to a client that gives the application access to queuing services on a server.

**client connection channel type.** The type of MQI channel definition associated with an MQI client. See also *server connection channel type*.

**cluster.** A network of queue managers that are logically associated in some way.

**coded character set identifier (CCSID).** The name of a coded set of characters and their code point assignments.

**command.** In MQSeries, an instruction that can be carried out by the queue manager.

**command bag.** In the MQAI, a type of bag that is created for administering MQSeries objects, but cannot change the order of data items nor create lists within a message.

**command processor.** The MQSeries component that processes commands.

**command server.** The MQSeries component that reads commands from the system-command input queue, verifies them, and passes valid commands to the command processor.

**commit.** The act of completing a transaction so that changes to the database a recorded and stable. Protected resources are released after the transaction is committed.

**Common Run-Time Environment (CRE).** A set of services that enable system and application programmers to write mixed-language programs. These shared, run-time services can be used by C, COBOL85, FORTRAN, Pascal, and TAL programs.

**completion code.** A return code indicating how an MQI call has ended.

**configuration file (also known as ini file).** A file that contains configuration information related to logs, communications, or installable services. See also *stanza*.

**connect.** To provide a queue manager connection handle, which an application uses on subsequent MQI calls. The connection is made either by the MQCONN call or automatically by the MQOPEN call.

**connection handle.** The identifier, or token, by which a program accesses the queue manager to which it is connected.

**context.** Information about the origin of a message.

**context security.** A method of allowing security to be handled such that messages are obliged to carry details of their origins in the message descriptor.

**controlled shutdown.** See *quiesced shutdown*.

**CRE.** Common Run-Time Environment.

**Customer Information Control System (CICS).** An IBM transaction management system that provides concurrent online access to data files by means of user-written application programs. CICS also includes facilities for building, using, and maintaining databases.

# D

**data bag.** In the MQAI, a bag that allows you to handle properties (or parameters) of objects.

**data item.** In the MQAI, an item contained within a data bag. This can be an integer item or a character-string item, and a user item or a system item.

**data conversion interface (DCI).** The MQSeries interface to which customer- or vendor-written programs that convert application data between different machine encodings and CCSIDs must conform. A part of the MQSeries Framework.

**datagram.** The simplest message that MQSeries supports. This type of message does not require a reply.

**DCE.** Distributed Computing Environment.

**DCI.** Data conversion interface.

**dead-letter queue (DLQ).** A queue to which a queue manager or application sends messages that it cannot deliver to their correct destination.

**dead-letter queue handler.** An MQSeries-supplied utility that monitors a dead-letter queue (DLQ) and processes messages on the queue in accordance with a user-written rules table.

**default object.** A definition of an object (for example, a queue) with all attributes defined. If a user defines an object but does not specify all possible attributes for that object, the queue manager uses default attributes in place of any that were not specified.

**distributed application.** In message queuing, a set of application programs that can each be connected to a different queue manager, but that collectively constitute a single application.

**Distributed Computing Environment (DCE).** Middleware that provides basic services, making the development of distributed applications easier. DCE is defined by the Open Software Foundation (OSF).

**distributed queue management.** In message queuing, the setup and control of message channels to queue managers on other systems.

**DLQ (dead-letter queue).** A queue to which a queue manager or application sends messages that it cannot deliver to their correct destination.

**dynamic queue.** A local queue that is created when a program opens a model queue object. See also *permanent dynamic queue* and *temporary dynamic queue*.

# E

**EC.** EC is a subsidiary controlling process in the queue manager, responsible for a set of agents.

**EC Boss.** The Execution Controller Boss is the main controlling process in the queue manager.

**EMS.** Event Monitoring System.

**event.** See *channel event*, *instrumentation event*, *performance event*, and *queue manager event*.

**event data.** In an event message, the part of the message data that contains information about the event (such as the queue manager name, and the application that gave rise to the event). See also *event header*.

**event header.** In an event message, the part of the message data that identifies the event type of the reason code for the event.

**event message.** Contains information (such as the category of event, the name of the application that caused the event, and queue manager statistics) relating to the origin of an instrumentation event in a network of MQSeries systems.

**event queue.** The queue onto which the queue manager puts an event message after it detects an event. Each category of event (queue manager, performance, or channel event) has its own event queue.

# F

**FFST.** First Failure Support Technology.

**FIFO (first-in-first-out).** A queuing technique in which the next item to be retrieved is the item that has been in the queue for the longest time.

**First Failure Support Technology (FFST).** Used by MQSeries on UNIX systems, MQSeries for OS/2 Warp, MQSeries for Windows NT and Windows 2000, and MQSeries for AS/400 to detect and report software problems.

**first-in-first-out (FIFO).** A queuing technique in which the next item to be retrieved is the item that has been in the queue for the longest time.

**Framework.** In MQSeries, a collection of programming interfaces that allow customers or vendors to write programs that extend or replace certain functions provided in MQSeries products. The interfaces are:
- MQSeries data conversion interface (DCI)
- MQSeries message channel interface (MCI)
- MQSeries name service interface (NSI)
- MQSeries security enabling interface (SEI)
- MQSeries trigger monitor interface (TMI)

# G

**get.** In message queuing, to use the MQGET call to remove a message from a queue. See also *browse*.

# H

**handle.** See *connection handle* and *object handle*.

# I

**ICE.** Intersystem Communications Environment is a family of Compaq-based software products that enables you to access a variety of applications on Compaq computers.

**immediate shutdown.** In MQSeries, a shutdown of a queue manager that does not wait for applications to disconnect. Current MQI calls are allowed to complete, but new MQI calls fail after an immediate shutdown has been requested. Contrast with *quiesced shutdown* and *preemptive shutdown*.

**ini file.** See *configuration file*.

**initiation queue.** A local queue on which the queue manager puts trigger messages.

**input/output parameter.** A parameter of an MQI call in which you supply information when you make the call, and in which the queue manager changes the information when the call completes or fails.

**input parameter.** A parameter of an MQI call in which you supply information when you make the call.

**installable services.** In MQSeries on UNIX systems, MQSeries for Compaq, MQSeries for OS/2 Warp, and MQSeries for Windows NT and Windows 2000, additional functionality provided as independent components. The installation of each component is optional: in-house or third-party components can be used instead. See also *authorization service*, *name service*, and *user identifier service*.

**instrumentation event.** A facility that can be used to monitor the operation of queue managers in a network of MQSeries systems. MQSeries provides instrumentation events for monitoring queue manager resource definitions, performance conditions, and channel conditions. Instrumentation events can be used by a user-written reporting mechanism in an administration application that displays the events to a system operator. They also allow applications acting as agents for other administration networks to monitor reports and create the appropriate alerts.

**Internet Protocol (IP).** A protocol used to route data from its source to its destination in an Internet environment. This is the base layer, on which other protocol layers, such as TCP and UDP are built.

**IP.** Internet Protocol

## L

**linear logging.** In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT and Windows 2000, the process of keeping restart data in a sequence of files. New files are added to the sequence as necessary. The space in which the data is written is not reused until the queue manager is restarted. Contrast with *circular logging*.

**listener.** In MQSeries distributed queuing, a program that monitors information about incoming network connections.

**local definition.** An MQSeries object that belongs to a local queue manager.

**local definition of a remote queue.** An MQSeries object that belongs to a local queue manager. This object defines the attributes of a queue that is owned by another queue manager. In addition, it is used for queue-manager aliasing and reply-to-queue aliasing.

**local queue.** A queue that belongs to the local queue manager. A local queue can contain a list of messages waiting to be processed. Contrast with *remote queue*.

**local queue manager.** The queue manager to which a program is connected and that provides message queuing services to the program. Queue managers to which a program is not connected are called *remote queue managers*, even if they are running on the same system as the program.

**log.** In MQSeries, records the work done by queue managers while they receive, transmit, and deliver messages, to enable them to recover in the event of failure.

**logical unit of work (LUW).** See *unit of work.*

**MCA (message channel agent).** A program that transmits prepared messages from a transmission queue to a communication link, or from a communication link to a destination queue.

**MCI (message channel interface).** The MQSeries interface to which customer- or vendor-written programs that transmit messages between an MQSeries queue manager and another messaging system must conform. A part of the MQSeries Framework.

**message.** In message queuing applications, a communication sent between programs. See also *persistent message* and *nonpersistent message*. In system programming, information intended for the terminal operator or system administrator.

**message channel.** In distributed message queuing, a mechanism for moving messages from one queue manager to another. A message channel comprises two message channel agents (a sender at one end and a receiver at the other end) and a communication link. Contrast with *MQI channel*.

**message channel agent (MCA).** A program that transmits prepared messages from a transmission queue to a communication link, or from a communication link to a destination queue. See also *message queue interface*.

**message channel interface (MCI).** The MQSeries interface to which customer- or vendor-written programs that transmit messages between an MQSeries queue manager and another messaging system must conform. A part of the MQSeries Framework.

**message descriptor.** Control information describing the message format and presentation that is carried as part of an MQSeries message. The format of the message descriptor is defined by the MQMD structure.

**message priority.** In MQSeries, an attribute of a message that can affect the order in which messages on a queue are retrieved and whether a trigger event is generated.

**message queue.** Synonym for *queue*.

**message queue interface (MQI).** The programming interface provided by the MQSeries queue managers. This programming interface lets application programs access message queuing services.

**message queue management.** The Message Queue Management (MQM) facility in MQSeries for Compaq NSK V2.2 uses PCF command formats and control commands. MQM runs as a PATHWAY SCOBOL requester under the Terminal Control Process (TCP) and uses an MQM SERVERCLASS server, which invokes the C language API to perform PCF commands. There is a separate instance of MQM for each queue manager configured on a system, since each queue manager is controlled under its own PATHWAY configuration. Consequently, an MQM is limited to the management of the queue manager to which it belongs.

**message queuing.** A programming technique in which each program within an application communicates with the other programs by putting messages on queues.

**message sequence numbering.** A programming technique in which messages are given unique numbers during transmission over a communication link. This enables the receiving process to check whether all messages are received, to place them in a queue in the original order, and to discard duplicate messages.

**messaging.** See *synchronous messaging* and *asynchronous messaging*.

**model queue object.** A set of queue attributes that act as a template when a program creates a dynamic queue.

**MQAI.** MQSeries Administration Interface.

**MQI (message queuing interface).** The programming interface provided by the MQSeries queue managers. This programming interface lets application programs access message queuing services.

**MQI channel.** Connects an MQI client to a queue manager on a server system, and transfers only MQI calls and responses in a bidirectional manner. Contrast with *message channel*.

**MQM.** Message Queue Management.

**MQI client.** Part of an MQSeries product that can be installed on a system without installing the full queue manager. The MQI client accepts MQI calls from applications and communicates with a queue manager on a server system.

**MQI server.** 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 system, that is, on the MQI server machine. A server can support normal local MQI applications as well.

**MQSC.** MQSeries commands.

**MQSeries.** A family of IBM licensed programs that provides message queuing services.

**MQSeries Administration Interface (MQAI).** A programming interface to MQSeries.

**MQSeries commands (MQSC).** Human readable commands, uniform across all platforms, that are used to manipulate MQSeries objects. Contrast with *programmable command format (PCF)*.

## N

**namelist.** An MQSeries object that contains a list of names, for example, queue names.

**name service interface (NSI).** The MQSeries interface to which customer- or vendor-written programs that resolve queue-name ownership must conform. A part of the MQSeries Framework.

**NetBIOS.** Network Basic Input/Output System. An operating system interface for application programs used on IBM personal computers that are attached to the IBM Token-Ring Network.

**nonpersistent message.** A message that does not survive a restart of the queue manager. Contrast with *persistent message*.

**NSI.** Name service interface.

**null character.** The character that is represented by X'00'.

## O

**object.** In MQSeries, an object is a queue manager, a queue, a process definition, a namelist, or a channel.

**Object authority manager (OAM).** In MQSeries on UNIX systems, MQSeries for Compaq, and MQSeries for Windows NT and Windows 2000;, the default authorization service for command and object management. The OAM can be replaced by, or run in combination with, a customer-supplied security service.

**object descriptor.** A data structure that identifies a particular MQSeries object (MQOD). Included in the descriptor are the name of the object and the object type.

**object handle.** The identifier, or token, by which a program accesses the MQSeries object with which it is working.

**output parameter.** A parameter of an MQI call in which the queue manager returns information when the call completes or fails.

# P

**PCF.** Programmable command format.

**PCF command.** See *programmable command format*.

**pending event.** An unscheduled event that occurs as a result of a connect request from a CICS adapter.

**performance event.** A category of event that indicates a limit condition has occurred.

**performance trace.** An MQSeries trace option where the trace data is to be used for performance analysis and tuning.

**permanent dynamic queue.** A dynamic queue that is deleted when it is closed only if deletion is explicitly requested. Permanent dynamic queues are recovered if the queue manager fails, so they can contain persistent messages. Contrast with *temporary dynamic queue*.

**persistent message.** A message that survives a restart of the queue manager. Contrast with *nonpersistent message*.

**ping.** In distributed queuing, a diagnostic aid that uses the exchange of a test message to confirm that a message channel is functioning.

**platform.** In MQSeries, the operating system under which a queue manager is running.

**preemptive shutdown.** In MQSeries, a shutdown of a queue manager that does not wait for connected applications to disconnect, nor for current MQI calls to complete. Contrast with *immediate shutdown* and *quiesced shutdown*.

**process definition object.** An MQSeries object that contains the definition of an MQSeries application. For example, a queue manager uses the definition when it works with trigger messages.

**programmable command format (PCF).** A type of MQSeries message that is used by:
- User administration applications that put PCF commands onto the system command input queue of a specified queue manager.
- User administration applications, to get the results of a PCF command from a specified queue manager.
- A queue manager, as a notification that an event has occurred.

Contrast with *MQSC*.

**program temporary fix (PTF).** A solution or by-pass of a problem diagnosed by IBM field engineering as the result of a defect in a current, unaltered release of a program.

**PTF.** Program temporary fix.

# Q

**queue.** An MQSeries object. Message queuing applications can put messages on, and get messages from, a queue. A queue is owned and maintained by a queue manager. Local queues can contain a list of messages waiting to be processed. Queues of other types cannot contain messages—they point to other queues, or can be used as models for dynamic queues.

**queue manager.** A system program that provides queuing services to applications. It provides an application programming interface so that programs can access messages on the queues that the queue manager owns. See also *local queue manager* and *remote queue manager*. An MQSeries object that defines the attributes of a particular queue manager.

**queue manager event.** An event that indicates:
- An error condition has occurred in relation to the resources used by a queue manager. For example, an error condition caused by a queue being unavailable.
- A significant change has occurred in the queue manager. For example, a queue manager has stopped or started.

**queue server.** NonStop process pair that supports all messaging operations for local queues.

**queuing.** See *message queuing*.

**quiesced shutdown.** In MQSeries, a shutdown of a queue manager that allows all connected applications to disconnect. Contrast with *immediate shutdown* and *preemptive shutdown*.

**quiescing.** In MQSeries, the state of a queue manager prior to it being stopped. In this state, programs are allowed to finish processing, but no new programs are allowed to start.

# R

**RBA.** Relative byte address.

**reason code.** A return code that describes the reason for the failure or partial success of an MQI call.

**receiver channel.** In message queuing, a channel that responds to a sender channel, takes messages from a communication link, and puts them on a local queue.

**remote queue.** A queue that belongs to a remote queue manager. Programs can put messages on remote queues, but they cannot get messages from remote queues. Contrast with *local queue*.

**remote queue manager.** To a program, a queue manager is remote if it is not the queue manager to which the program is connected.

**remote queue object.** See *local definition of a remote queue*.

**remote queuing.** In message queuing, the provision of services to enable applications to put messages on queues belonging to other queue managers.

**reply message.** A type of message used for replies to request messages.

**reply-to queue.** The name of a queue to which the program that issued an MQPUT call wants a reply message or report message sent.

**report message.** A type of message that gives information about another message. A report message can indicate that a message has been delivered, has arrived at its destination, has expired, or could not be processed for some reason.

**repository.** A collection of information about the queue managers that are members of a cluster. This information includes queue manager names, their locations, their channels, what queues they host, and so on.

**requester channel.** In message queuing, a channel that may be started remotely by a sender channel. The requester channel accepts messages from the sender channel over a communication link and puts the messages on the local queue designated in the message. See also *server channel*.

**request message.** A type of message used for requesting a reply from another program.

**resolution path.** The set of queues that are opened when an application specifies an alias or a remote queue on input to the MQOPEN call.

**responder.** In distributed queuing, a program that replies to network connection requests from another system.

**resynch.** In MQSeries, an option to direct a channel to start up and resolve any in-doubt status messages, but without restarting message transfer.

**return codes.** The collective name for completion codes and reason codes.

**rollback.** Synonym for *back out*.

**rules table.** A control file containing one or more rules that the dead-letter queue handler applies to messages on the DLQ.

# S

**security enabling interface (SEI).** The MQSeries interface to which customer- or vendor-written programs that check authorization, supply a user identifier, or perform authentication must conform. A part of the MQSeries Framework.

**SEI.** Security enabling interface.

**sender channel.** In message queuing, a channel that initiates transfers, removes messages from a transmission queue, and moves them over a communication link to a receiver or requester channel.

**sequential delivery.** In MQSeries, a method of transmitting messages with a sequence number so that the receiving channel can reestablish the message sequence when storing the messages. This is required where messages must be delivered only once, and in the correct order.

**sequential number wrap value.** In MQSeries, a method of ensuring that both ends of a communication link reset their current message sequence numbers at the same time. Transmitting messages with a sequence number ensures that the receiving channel can reestablish the message sequence when storing the messages.

**server.** (1) In MQSeries, a queue manager that provides queue services to client applications running on a remote workstation. (2) The program that responds to requests for information in the particular two-program, information-flow model of client/server. See also *client*.

**server channel.** In message queuing, a channel that responds to a requester channel, removes messages from a transmission queue, and moves them over a communication link to the requester channel.

**server connection channel type.** The type of MQI channel definition associated with the server that runs a queue manager. See also *client connection channel type*.

**service interval.** A time interval, against which the elapsed time between a put or a get and a subsequent get is compared by the queue manager in deciding whether the conditions for a service interval event have been met. The service interval for a queue is specified by a queue attribute.

**service interval event.** An event related to the service interval.

**shutdown.** See *immediate shutdown*, *preemptive shutdown*, and *quiesced shutdown*.

**single-phase back out.** A method in which an action that is in progress must not be allowed to finish, and all changes that are part of that action must be undone.

**single-phase commit.** A method in which a program can commit updates to a queue without coordinating those updates with updates the program has made to resources controlled by another resource manager. Contrast with *two-phase commit*.

**SIT.** System initialization table.

**SNA.** Systems Network Architecture.

**stanza.** A group of lines in a configuration file that assigns a value to a parameter that modifies the behavior of a queue manager, client, or channel. In MQSeries on systems, a configuration (ini) file can contain a number of stanzas.

**Status Server.** Supports all status information for all objects other than local queues. The default Status Server also handles channel status information for the queue manager.

**store and forward.** The temporary storing of packets, messages, or frames in a data network before they are retransmitted toward their destination.

**symptom string.** Diagnostic information displayed in a structured format designed for searching the IBM software support database.

**synchronous messaging.** A method of communication between programs in which programs place messages on message queues. With synchronous messaging, the sending program waits for a reply to its message before resuming its own processing. Contrast with *asynchronous messaging*.

**syncpoint.** An intermediate or end point during processing of a transaction at which the transaction's protected resources are consistent. At a syncpoint, changes to the resources can safely be committed, or they can be backed out to the previous syncpoint.

**system bag.** A type of data bag that is created by the MQAI.

**system.command.input queue.** A local queue on which application programs can put MQSeries commands. The commands are retrieved from the queue by the command server, which validates them and passes them to the command processor to be run.

**system control commands.** Commands used to manipulate platform-specific entities such as buffer pools, storage classes, and page sets.

# T

**TCP.** Transmission Control Protocol.

**TCP/IP.** Transmission Control Protocol/Internet Protocol.

**TACL.** Tandem Advanced Command Language.

**temporary dynamic queue.** A dynamic queue that is deleted when it is closed. Temporary dynamic queues are not recovered if the queue manager fails, so they can contain nonpersistent messages only. Contrast with *permanent dynamic queue*.

**thread.** In MQSeries, the lowest level of parallel execution available on an operating system platform.

**time-independent messaging.** See *asynchronous messaging*.

**TMF.** Transaction Management Facility.

**TMI.** Trigger monitor interface.

**TM/MP.** NonStop Transaction Manager/MP.

**tranid.** See *transaction identifier*.

**transmission program.** See *message channel agent*.

**Transmission Control Protocol (TCP).** Part of the TCP/IP protocol suite. A host-to-host protocol between hosts in packet-switched communications networks. TCP provides connection-oriented data stream delivery. Delivery is reliable and orderly.

**Transmission Control Protocol/Internet Protocol (TCP/IP) .** A suite of communication protocols that support peer-to-peer connectivity functions for both local and wide area networks.

**transmission queue.** A local queue on which prepared messages destined for a remote queue manager are temporarily stored.

**trigger event.** An event (such as a message arriving on a queue) that causes a queue manager to create a trigger message on an initiation queue.

**triggering.** In MQSeries, a facility that lets a queue manager start an application automatically when predetermined conditions on a queue are satisfied.

**trigger message.** A message that contains information about the program that a trigger monitor is to start.

**trigger monitor.** A continuously-running application that serves one or more initiation queues. When a trigger message arrives on an initiation queue, the trigger monitor retrieves the message. It uses the information in the trigger message to start a process that serves the queue on which a trigger event occurred.

**trigger monitor interface (TMI).** The MQSeries interface to which customer- or vendor-written trigger monitor programs must conform. A part of the MQSeries Framework.

**two-phase commit.** A protocol for the coordination of changes to recoverable resources when more than one resource manager is used by a single transaction. Contrast with *single-phase commit*.

# U

**UDP.** User Datagram Protocol.

**undelivered-message queue.** See *dead-letter queue*.

**unit of recovery.** A recoverable sequence of operations within a single resource manager. Contrast with *unit of work*.

**unit of work.** A recoverable sequence of operations performed by an application between two points of consistency. A unit of work begins when a transaction starts or after a user-requested syncpoint. It ends either at a user-requested syncpoint or at the end of a transaction. Contrast with *unit of recovery*.

**user bag.** In the MQAI, a type of data bag that is created by the user.

**User Datagram Protocol (UDP).** Part of the TCP/IP protocol suite. A packet-level protocol built directly on the Internet Protocol layer. UDP is a connectionless and less reliable alternative to TCP. It is used for application-to-application programs between TCP/IP host systems.

**utility.** In MQSeries, a supplied set of programs that provide the system operator or system administrator with facilities in addition to those provided by the MQSeries commands. Some utilities invoke more than one function.

# Index

© Copyright IBM Corp. 1993, 2001

# 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 44–1962–816151
  - From within the U.K., use 01962–816151
- 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.

© Copyright IBM Corp. 1993, 2001

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