MQSeries

Planning Guide

GC33-1349-06

MQSeries

Planning Guide

GC33-1349-06

Note!

Before using this information and the product it supports, be sure to read the general information under Appendix, “Notices” on page 259.

Seventh edition (February 1998)

This edition applies to the following products:

- MQSeries for AIX Version 5
- MQSeries for AS/400 Version 4 Release 2
- MQSeries for AT&T GIS UNIX Version 2 Release 2
- MQSeries for Digital OpenVMS AXP Version 2 Release 2
- MQSeries for Digital OpenVMS VAX Version 2 Release 2
- MQSeries for HP-UX Version 5
- MQSeries for MVS/ESA Version 1 Release 2
- MQSeries for OS/2 Warp Version 5
- MQSeries for SCO UNIX Version 1 Release 4
- MQSeries for SINIX and DC/OSx Version 2 Release 2
- MQSeries for SunOS Version 2 Release 2
- MQSeries for Sun Solaris Version 5
- MQSeries for Tandem NonStop Kernel Version 2 Release 2
- MQSeries Three Tier for OS/2 Version 1.0
- MQSeries Three Tier for AIX Version 1.0
- MQSeries for UnixWare Version 1 Release 4.1
- MQSeries for VSE/ESA Version 1 Release 4
- MQSeries for Windows NT Version 5
- MQSeries for Windows Version 2 Release 0
- MQSeries for Windows Version 2 Release 1
- MQSeries link for R/3 for AIX Version 1.0
- MQSeries link for R/3 for HP-UX Version 1.0
- MQSeries link for R/3 for Sun Solaris Version 1.0
- MQSeries link for R/3 for Windows NT Version 1.0

and to any subsequent releases and modifications until otherwise indicated in new editions.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the address given below.

At the back of this publication is a page titled “Sending your comments to IBM”. If you want to make comments, but the methods described are not available to you, please address them to:

IBM United Kingdom Laboratories,
Information Development,
Mail Point 095,
Hursley Park,
Winchester,
Hampshire,
England,
SO21 2JN

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

Contents

Figures

Tables

About this book

The MQSeries family of products provides application programming services that:

- Enable you to code indirect program-to-program communication using *message queues*.
- Are independent of the platform, for example, MVS/ESA or OS/400.

This book explains the planning necessary for the incorporation of MQSeries products into your enterprise.

Who this book is for

This book is for:

- Planners of systems that will use MQSeries message-queuing techniques.
- System programmers who have to install and customize MQSeries products for these systems.

What you need to know to understand this book

To understand this book, you should be familiar with the system facilities for the platform on which you are installing the MQSeries product.

If you are unfamiliar with the concepts of messaging and queuing, you should read *MQSeries: An Introduction to Messaging and Queuing.*

Terms used in this book

All new terms that this book introduces are defined in the “Glossary of terms and abbreviations” on page 263. These terms are shown like *this* at their first use. This book uses the following shortened names:

MQSeries	General term for IBM MQSeries products.
CICS	General term for CICS and Transaction Server on all platforms.
UNIX systems	General term referring to the following UNIX systems: • AIX • AT&T** GIS UNIX[1] • HP-UX** • SINIX** and DC/OSx** • SunOS** • Sun Solaris**

[1] This platform has become NCR UNIX SVR4 MP-RAS, R3.0

MQSeries publications

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 Level 2 products. The latest MQSeries Level 2 products are:

- MQSeries for AIX V5.0
- MQSeries for AS/400 V4R2
- MQSeries for AT&T GIS UNIX V2.2
- MQSeries for Digital OpenVMS V2.2
- MQSeries for HP-UX V5.0
- MQSeries for MVS/ESA V1.2
- MQSeries for OS/2 Warp V5.0
- MQSeries for SINIX and DC/OSx V2.2
- MQSeries for SunOS V2.2
- MQSeries for Sun Solaris V5.0
- MQSeries for Tandem NonStop Kernel V2.2
- MQSeries Three Tier
- MQSeries for Windows V2.0
- MQSeries for Windows V2.1
- MQSeries for Windows NT V5.0

Any exceptions to this general rule are indicated. (Publications that support the MQSeries Level 1 products are listed in "MQSeries Level 1 product publications" on page xv. For a functional comparison of the Level 1 and Level 2 MQSeries products, see Chapter 44, "MQSeries product summaries" on page 209.)

MQSeries Brochure
The *MQSeries Brochure*, G511-1908, gives a brief introduction to the benefits of MQSeries. It is intended to support the purchasing decision, and describes some authentic customer use of MQSeries.

MQSeries: An Introduction to Messaging and Queuing
MQSeries: An Introduction to Messaging and Queuing, GC33-0805, describes briefly what MQSeries is, how it works, and how it can solve some classic interoperability problems. This book is intended for a more technical audience than the *MQSeries Brochure*.

MQSeries Planning Guide
The *MQSeries Planning Guide*, GC33-1349, describes some key MQSeries concepts, identifies items that need to be considered before MQSeries is installed, including storage requirements, backup and recovery, security, and migration from earlier releases, and specifies hardware and software requirements for every MQSeries platform.

MQSeries Intercommunication
The *MQSeries Intercommunication* book, SC33-1872, defines the concepts of distributed queuing and explains how to set up a distributed queuing network in a variety of MQSeries environments. In particular, it demonstrates how to (1) configure communications to and from a representative sample of MQSeries products, (2) create required MQSeries objects, and (3) create and configure MQSeries channels. The use of channel exits is also described.

MQSeries Clients
The *MQSeries Clients* book, GC33-1632, describes how to install, configure, use, and manage MQSeries client systems.

MQSeries System Administration
The *MQSeries System Administration* book, SC33-1873, supports day-to-day management of local and remote MQSeries objects. It includes topics such as security, recovery and restart, transactional support, problem determination, the dead-letter queue handler, and the MQSeries links for Lotus Notes**. It also includes the syntax of the MQSeries control commands.

This book applies to the following MQSeries products only:

- MQSeries for AIX V5.0
- MQSeries for HP-UX V5.0
- MQSeries for OS/2 Warp V5.0
- MQSeries for Sun Solaris V5.0
- MQSeries for Windows NT V5.0

MQSeries Command Reference
The *MQSeries Command Reference*, SC33-1369, contains the syntax of the MQSC commands, which are used by MQSeries system operators and administrators to manage MQSeries objects.

MQSeries Programmable System Management
The *MQSeries Programmable System Management* book, SC33-1482, provides both reference and guidance information for users of MQSeries events, programmable command formats (PCFs), and installable services.

MQSeries Messages
The *MQSeries Messages* book, GC33-1876, which describes “AMQ” messages issued by MQSeries, applies to these MQSeries products only:

- MQSeries for AIX V5.0
- MQSeries for HP-UX V5.0
- MQSeries for OS/2 Warp V5.0
- MQSeries for Sun Solaris V5.0
- MQSeries for Windows NT V5.0
- MQSeries for Windows V2.0
- MQSeries for Windows V2.1

This book is available in softcopy only.

MQSeries Application Programming Guide
The *MQSeries Application Programming Guide*, SC33-0807, provides guidance information for users of the message queue interface (MQI). It describes how to design, write, and build an MQSeries application. It also includes full descriptions of the sample programs supplied with MQSeries.

MQSeries Application Programming Reference
The *MQSeries Application Programming Reference*, SC33-1673, provides comprehensive reference information for users of the MQI. It includes: data-type descriptions; MQI call syntax; attributes of MQSeries objects; return codes; constants; and code-page conversion tables.

MQSeries Application Programming Reference Summary
The *MQSeries Application Programming Reference Summary*, SX33-6095, summarizes the information in the *MQSeries Application Programming Reference* manual.

MQSeries Using C++
MQSeries Using C++, SC33-1877, provides both guidance and reference information for users of the MQSeries C++ programming-language binding to the MQI. MQSeries C++ is supported by V5.0 of MQSeries for AIX, HP-UX, OS/2 Warp, Sun Solaris, and Windows NT, and by MQSeries clients supplied with those products and installed in the following environments:

- AIX
- HP-UX
- OS/2
- Sun Solaris
- Windows NT
- Windows 3.1
- Windows 95

MQSeries C++ is also supported by MQSeries for AS/400 V4R2.

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

MQSeries for AIX V5.0 Quick Beginnings, GC33-1867

MQSeries for AS/400

MQSeries for AS/400 Version 4 Release 2 Licensed Program Specifications, GC33-1958

MQSeries for AS/400 Version 4 Release 2 Administration Guide, GC33-1956

MQSeries for AS/400 Version 4 Release 2 Application Programming Reference (RPG), SC33-1957

MQSeries for AT&T GIS UNIX

MQSeries for AT&T GIS UNIX Version 2.2 System Management Guide, SC33-1642

MQSeries for Digital OpenVMS

MQSeries for Digital OpenVMS Version 2.2 System Management Guide, GC33-1791

MQSeries for HP-UX

MQSeries for HP-UX V5.0 Quick Beginnings, GC33-1869

MQSeries for MVS/ESA

MQSeries for MVS/ESA Version 1 Release 2 Licensed Program Specifications, GC33-1350

MQSeries for MVS/ESA Version 1 Release 2 Program Directory

MQSeries for MVS/ESA Version 1 Release 2 System Management Guide, SC33-0806

MQSeries for MVS/ESA Version 1 Release 2 Messages and Codes, GC33-0819

MQSeries for MVS/ESA Version 1 Release 2 Problem Determination Guide, GC33-0808

MQSeries for OS/2 Warp

MQSeries for OS/2 Warp V5.0 Quick Beginnings, GC33-1868

MQSeries link for R/3

MQSeries link for R/3 Version 1.0 User's Guide, GC33-1934

MQSeries for SINIX and DC/OSx

MQSeries for SINIX and DC/OSx Version 2.2 System Management Guide, GC33-1768

MQSeries for SunOS

MQSeries for SunOS Version 2.2 System Management Guide, GC33-1772

MQSeries for Sun Solaris

MQSeries for Sun Solaris V5.0 Quick Beginnings, GC33-1870

MQSeries for Tandem NonStop Kernel

MQSeries for Tandem NonStop Kernel Version 2.2 System Management Guide, GC33-1893

MQSeries Three Tier

MQSeries Three Tier Administration Guide, SC33-1451
MQSeries Three Tier Reference Summary, SX33-6098
MQSeries Three Tier Application Design, SC33-1636
MQSeries Three Tier Application Programming, SC33-1452

MQSeries for Windows

MQSeries for Windows Version 2.0 User's Guide, GC33-1822

MQSeries for Windows Version 2.1 User's Guide, GC33-1965

MQSeries for Windows NT

MQSeries for Windows NT V5.0 Quick Beginnings, GC33-1871

MQSeries Level 1 product publications

For information about the MQSeries Level 1 products, see the following publications:

MQSeries: Concepts and Architecture, GC33-1141

MQSeries Version 1 Products for UNIX Operating Systems Messages and Codes, SC33-1754

MQSeries for SCO UNIX Version 1.4 User's Guide, SC33-1378

MQSeries for UnixWare Version 1.4.1 User's Guide, SC33-1379

MQSeries for VSE/ESA Version 1 Release 4 Licensed Program Specifications, GC33-1483

MQSeries for VSE/ESA Version 1 Release 4 User's Guide, SC33-1142

Softcopy books

Most of the MQSeries books are supplied in both hardcopy and softcopy formats.

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 products, including all MQSeries V5.0 products. Books in PostScript format can be printed on a PostScript printer or viewed with a suitable viewer.

HTML format

The MQSeries documentation is provided in HTML format with these MQSeries products:

- MQSeries for AIX V5.0
- MQSeries for HP-UX V5.0
- MQSeries for OS/2 Warp V5.0
- MQSeries for Sun Solaris V5.0
- MQSeries for Windows NT V5.0

The MQSeries books are also available from the MQSeries product family Web site:

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

Information Presentation Facility (IPF) format

In the OS/2 environment, the MQSeries documentation is supplied in IBM IPF format on the MQSeries product CD-ROM.

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

MQSeries web site

The MQSeries product family Web site is at:

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

By following links from this Web site you can:

- Obtain latest information about the MQSeries product family.
- Access the MQSeries books in HTML format.
- Download MQSeries SupportPacs.

Summary of changes

This section lists the changes that have been made to this book since previous editions. Changes for this edition are marked with vertical bars in the left-hand margin.

Changes for this edition

Changes for edition number GC33-1349-06 include:

- The book has been updated to contain information about the following new functions of the MQSeries for AS/400 Version 4 Release 2 product:
  - Distribution lists
  - Message segmentation
  - Fast channels for nonpersistent messages
  - Auto-definition of server-connection and receiver channels
  - Reference messages
  - C++ programming
- The book has been updated to contain revised information about MQSeries for Digital OpenVMS, Version 2 Release 2.
- The book has been updated to contain information about MQSeries for Tandem NSK, Version 2 Release 2.
- The book has been updated to contain information about the MQSeries client for VM/ESA, Version 2 Release 3.

Changes for the sixth edition

Changes for edition number GC33-1349-05 include:

- The book has been updated to contain information about the following new releases of MQSeries products:
  - MQSeries for AIX Version 5
  - MQSeries for Digital VMS VAX Version 1.5
  - MQSeries for HP-UX Version 5
  - MQSeries for MVS/ESA Version 1.2
  - MQSeries for OS/2 Warp Version 5
  - MQSeries for OS/400 Version 3.7
  - MQSeries for Sun Solaris Version 5
  - MQSeries for Tandem NonStop Kernel Version 1.5.1
  - MQSeries for Windows NT Version 5
- The following products have been added to the MQSeries family:
  - MQSeries for Digital OpenVMS AXP V2.2
  - MQSeries for Digital OpenVMS VAX V2.2
  - MQSeries for Windows V2.0 and V2.1
  - MQSeries link for R/3 for AIX V 1.0
  - MQSeries link for R/3 for HP-UX V 1.0
  - MQSeries link for R/3 for Sun Solaris V 1.0
  - MQSeries link for R/3 for Windows NT V 1.0

- The following new functions have been added for some platforms:
  - DCE naming component
  - Exits relating to DCE security
  - Default objects created when the queue manager is created
  - Distribution lists
  - Fast channels for nonpersistent messages
  - Interlink SNS/TCPaccess support
  - Maximum message length has been increased
  - Message segmentation
  - MQBEGIN function
  - MQSeries as a transaction manager
  - Reference messages
  - SPX support
- The “at a glance” sections have been updated to reflect the latest product levels.
- Chapter 32 gives overview and planning information about the MQSeries for Windows product.

Part 1. Introduction to MQSeries

Chapter 1. Introduction to IBM MQSeries

This chapter introduces IBM MQSeries and describes its relationship with other products. It contains basic explanations of the following topics:

For more detailed explanations of these topics, refer to the *MQSeries Application Programming Reference* manual.

MQSeries and message queuing

MQSeries products enable applications to use message queuing to participate in *message-driven processing*. With message-driven processing, applications can communicate with each other on the same or different platforms, by using the appropriate message queuing software products. For example, MVS/ESA and OS/400 applications can communicate through MQSeries for MVS/ESA and MQSeries for AS/400 respectively. With MQSeries products, all applications use the same kind of messages; communications protocols are hidden from the applications.

MQI – a common application programming interface

MQSeries products implement a common application programming interface, the *message queue interface* (MQI), that is used on whatever platform the applications run on. The calls made by the applications and the messages they exchange are common. This makes it much easier to write and maintain applications than using traditional methods. It also facilitates the migration of message queuing applications from one platform to another.

The MQI calls are shown in the following table.

Table 1 (Page 1 of 2). MQI calls

Call name	Description	Platforms
MQBEGIN	Begin a unit of work to be coordinated by the queue manager	AIX, HP-UX, OS/2 Warp, Sun Solaris, and Windows NT
MQCONN	Connect to a queue manager	All platforms
MQCONNX	Connect to a queue manager and specify some options	AIX, HP-UX, OS/2 Warp, Sun Solaris, and Windows NT
MQDISC	Disconnect from a queue manager	All platforms
MQOPEN	Open an object	All platforms
MQCLOSE	Close an object	All platforms
MQPUT	Put a message (queue already open)	All platforms

Table 1 (Page 2 of 2). MQI calls		
Call name	**Description**	**Platforms**
MQPUT1	Put one message	All platforms
MQGET	Get a message	All platforms
MQCMIT	Commit changes	MVS/ESA, OS/2 Warp, Windows NT, and UNIX systems
MQBACK	Back out changes	MVS/ESA, OS/2 Warp, Windows NT, and UNIX systems.
MQINQ	Inquire about object attributes	All platforms
MQSET	Set object attributes	All platforms

Time-independent applications

With message queuing, the exchange of messages between the sending and receiving programs is time independent. This means that the sending and receiving applications are decoupled so that the sender can continue processing without having to wait for the receiver to acknowledge the receipt of the message. The receiving application might be busy when the message is sent. Indeed, the receiving application doesn't even need to be running. MQSeries holds the message in the queue until it can be processed.

Message-driven processing

Message-driven processing is a style of application design.

With this style, the application is divided into a number of separate, discrete, functional blocks, with each block having well-defined input and output parameters. Each functional block is coded as an application program, with its input and output parameters being interchanged between other application programs by placing their values in messages, which are then put on queues.

By use of the appropriate MQSeries programming mechanisms, an application program can start executing as a result of one or more messages arriving on a queue. If required, the program can terminate when all the messages in a queue have been processed.

This style of application design allows new applications to be built, or existing applications to be modified, more quickly than with some other application design styles.

Data integrity and resource protection

MQSeries applications can transfer data with an extremely high degree of confidence. Message delivery can be implemented using a syncpoint mechanism–and the MQSeries logs or journals–for the recovery of important data in the event of system failure. See Chapter 4, “Introduction to MQSeries recovery concepts” on page 27.

All resources, such as message queues, can be protected using the security facilities available on the operating platform. For example, Resource Access Control Facility (RACF) on MVS/ESA, or the security facilities provided by OS/400.

Messages and queues

Messages and queues are basic to any queuing system.

What is a message

A *message* is a string of bytes that has meaning to the applications that use the message.

In MQSeries, messages have two parts, a *message descriptor* and *application data*. The content and structure of the application data are defined by the application programs that use them. The message descriptor identifies the message and contains other control information or attributes, such as the date and time the message was created, the type of message, and the priority assigned to the message by the sending application.

Figure 1 represents a message and shows how the message is logically divided into message data and application data.

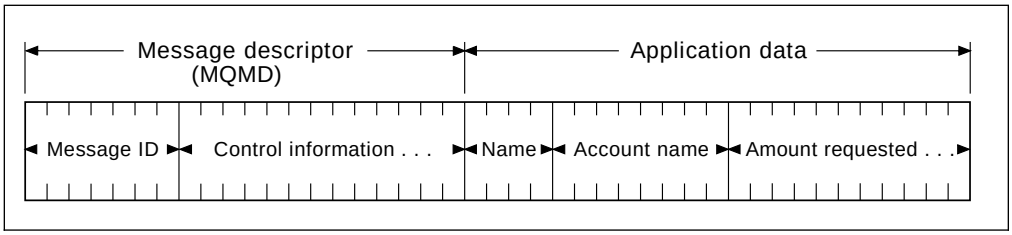

Figure 1. Representation of a message. The message descriptor and application data are shown as separate parts. Information that is specific to the application, such as Account name in this example, is in the application data part of the message.

What is a queue

In physical terms, a *queue* is a type of list that is used to store messages until they are retrieved by an application.

Queues exist independently of the applications that use them. A queue can exist:

- In main storage if it is temporary
- On disk or similar auxiliary storage if it must be kept in case of recovery
- In both places if it is currently being used, and must also be kept for recovery

Each queue belongs to a *queue manager*, which is responsible for maintaining it. The queue manager puts the messages it receives onto the appropriate queue.

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

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

- First-in-first-out (FIFO).
- Message priority, as defined in the message descriptor. Messages having the same priority are retrieved on a FIFO basis.

- A program request for a specific message.

For more information about queues and their attributes, refer to the *MQSeries Application Programming Guide*.

Message attributes

For system administrators, messages have two important attributes that are defined in the message descriptor: *persistence* and *priority*.

A message is termed persistent if it survives when MQSeries restarts. This implies that the message must be logged, or saved, and can be reinstated as part of the recovery procedure.

Each MQSeries message has a priority assigned to it by the sending application. The priority, which is a number in the range 0 through 9, can affect the order in which a message is retrieved from a queue, and also the way that *trigger events* are generated.

Triggering is a facility in some MQSeries products. It allows an application to be started automatically when predefined conditions on a queue are met. These conditions include reception of any message or of messages over a particular priority, the number of messages on a queue, and so on. For more information about triggering, see the *MQSeries Application Programming Guide*.

Message sizes

The maximum message size supported by MQSeries varies. Section “MQSeries product functional comparison” on page 212 gives details of the size of message supported by each of the MQSeries products.

In practice, the size of message that an application program can put on a queue is limited by:

- The maximum message length defined for the receiving queue
- The maximum message length defined for the queue manager
- The maximum message length supported by the platform

If the size of the data that an application program requires to place on a queue exceeds this limit, the program must split the data into a number of pieces, and put each piece on the queue as a separate message.

Dealing with large messages

On some platforms, there are several methods available for dealing with very large messages.

Maximum message length

On AIX, HP-UX, OS/2 Warp, Sun Solaris, and Windows NT, you can use the maximum message length attribute to set the maximum length allowed for messages on a queue manager; this can be up to 100 MB. The maximum message length used for communication between two queue managers is the lower of the maximum length for each queue manager, so you can use 100 MB messages only if both queue managers support this.

Message segmentation

On AIX, HP-UX, OS/2 Warp, OS/400, Sun Solaris, and Windows NT, MQSeries allows you to divide large messages into several parts (that is, one

logical message into several physical messages). Your application can define the size of physical messages, to allow for the maximum message size permitted at intermediate queue managers. MQSeries can start transmitting or receiving the physical messages before the whole of the logical message has arrived on the queue.

Reference messages

On AIX, HP-UX, OS/2 Warp, OS/400, Sun Solaris, and Windows NT, reference messages allow you to transfer a large data object from one queue manager to another without storing the object on a queue at either the source or destination node. A message exit program is used to add the data object to the message when it is transmitted. At the receiving end, another message exit program is used to create a data object from the data in the message. The reference message, without the data, is then put on the destination queue. Sample exit programs that do this are provided with MQSeries.

Some queues used with MQSeries

MQSeries uses the following types of queues.

Message queues

A *message queue* is a queue that is used to receive messages from applications as distinct from those queues that have a special purpose. Special purpose queues are defined in the same way as message queues, although they can have their attributes set in specific ways. The system administrator is the person responsible for defining and maintaining all queues in your enterprise.

Event queues

An *event queue* is a queue that is used to receive *event messages*, which indicate that a particular type of instrumentation event has occurred during the execution of an application program. Instrumentation events help you to monitor your system. There are three system administration event queues, one for each of the three categories of instrumentation event that can be generated:

- SYSTEM.ADMIN.QMGR.EVENT - for queue manager events
- SYSTEM.ADMIN.PERFM.EVENT - for performance events
- SYSTEM.ADMIN.CHANNEL.EVENT - for channel events

When an event is generated, it is put on one of these queues.

Initiation queues

An *initiation queue* receives *trigger messages*, which indicate that a trigger event has occurred. A trigger event is caused by a message that satisfies the specified conditions being put onto a queue. Messages are read from the initiation queue by a trigger monitor application which then starts the appropriate application to process the message. If triggers are active, at least one initiation queue must be defined for each queue manager.

Transmission queues

A *transmission queue* temporarily stores messages that are destined for a remote queue manager. You must define a transmission queue for each remote queue manager to which the local queue manager is to send messages. It is possible to associate several transmission queues with different characteristics with a remote queue manager. This allows different classes of transmission service.

Reply-to queues

If a message is a request message and so requires a reply, the sender of the message must specify the name of the queue to which the reply should be sent; this is called the *reply-to queue.* It is also the queue to which report messages are usually sent, if any are generated.

Dead-letter queues

A *dead-letter queue* (also known as an *undelivered-message queue*) receives messages that cannot be routed to their correct destinations. This occurs when, for example:

- The destination queue is full
- The message cannot be put on the destination queue
- The sender is not authorized to use the destination queue
- The destination queue does not exist

If you do not have a dead-letter queue, undelivered messages will remain on the transmission queue and the *message channel* will be stopped. You are therefore recommended to define a dead-letter queue for each queue manager in your system.

You also have to decide how to handle any message placed on a dead-letter queue. Such handling might be by the system administrator examining the queue and redirecting the message, or perhaps by an application which you create for the purpose of monitoring and handling dead-letter queues.

A dead-letter queue handler is provided with some MQSeries products to assist the system administrator with the task of dealing with messages placed on a dead-letter queue.

Command queues

A *command queue* is a queue owned by a queue manager to which suitably authorized applications can send messages containing MQSeries administration commands. The commands in these messages are processed by the *command server* part of the queue manager.

You can find more information on command queues in Chapter 5, “Introduction to MQSeries administration” on page 33.

System default queues

The *system default queues* are a set of queue definitions supplied with MQSeries. They are used to set default values for any attributes that you do not specify when defining your own queues. By modifying the supplied queue definitions you can vary the default queue attributes used at your installation.

MQSeries objects

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

- Queue managers
- Queues
- Namelists
- Distribution lists
- Process definitions
- Channels
- Storage classes

These objects are common across different MQSeries platforms.

For system administrators, commands are available to manipulate objects. The format of the commands is dependent on the platform. For example, the MQSeries for MVS/ESA command DEFINE QLOCAL (with the appropriate attributes) defines a local queue object for MQSeries for MVS/ESA. On MQSeries for AS/400, the equivalent command is CRTMQMQ.

On AIX, HP-UX, OS/2 Warp, Sun Solaris, and Windows NT, default objects are created for you when you create a queue manager. Default objects are supplied with the other MQSeries products, to help you define the objects that you need.

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

Note: The characters within the names given to all MQSeries objects are case sensitive. Therefore, be very careful when defining the names of objects, to select the appropriate uppercase or lowercase characters.

Queue managers

A *queue manager* is that part of an MQSeries product that provides the messaging and queuing services to application programs, through the Message Queue Interface (MQI) program calls.

In some environments, for example with MQSeries for AS/400, only one queue manager can be in use in one machine at any one time; in other environments, for example with MQSeries for MVS/ESA, more than one queue manager can be executing in one machine at one time.

In the MVS/ESA environment, applications are connected to a queue manager through adapters which are part of the MQSeries for MVS/ESA product. In the example shown in Figure 2 on page 10, there are two applications on MVS/ESA, one is a CICS transaction that is connected to the MQSeries queue manager by the CICS adapter, and the other a batch application connected to the queue manager by the batch adapter. Each application has access to local queues A1 and A2.

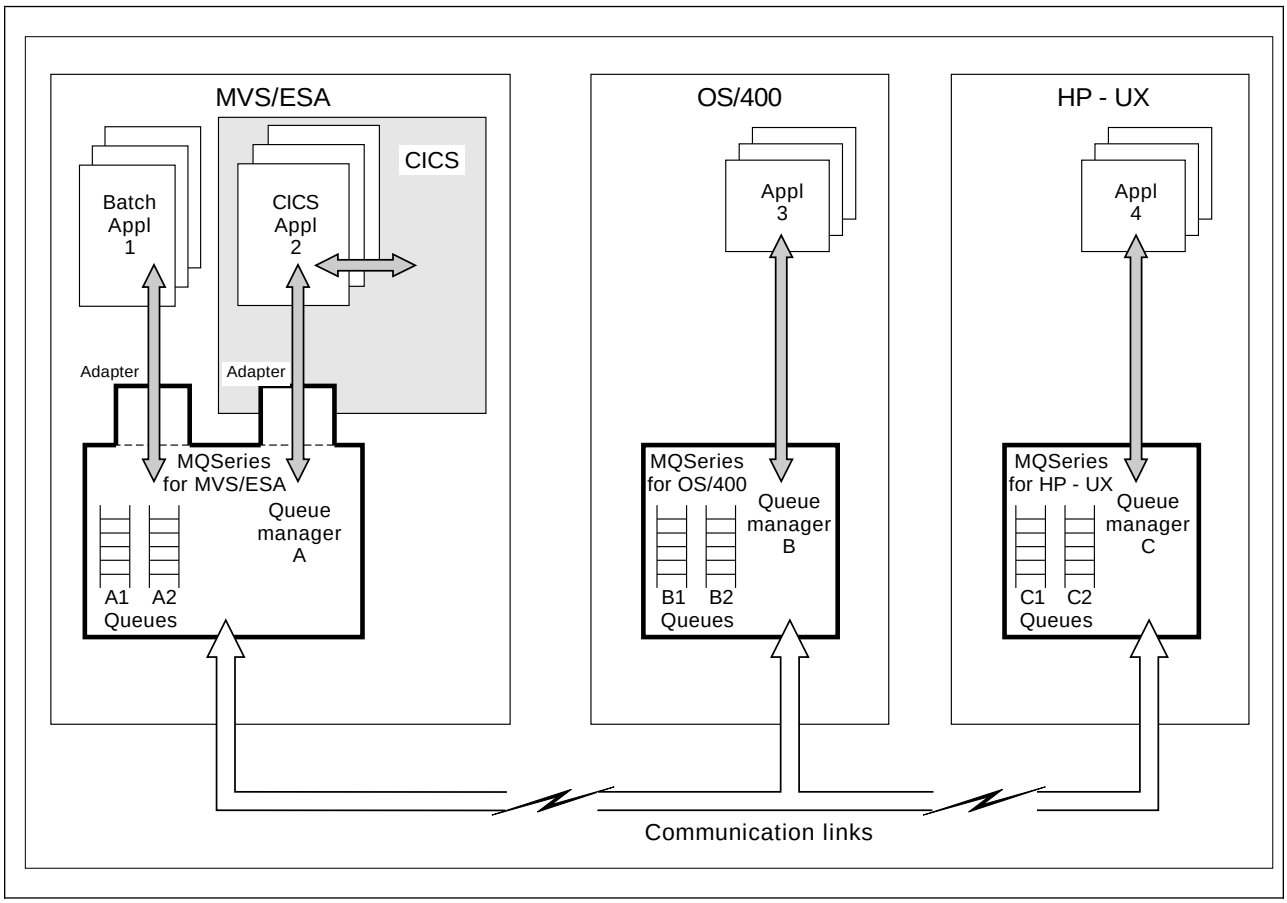

Figure 2. Example of messaging and queuing using MQSeries products

The applications issue the MQI calls that are implemented by the queue manager. For incoming messages, the queue manager directs them onto their respective destination queues; for outgoing messages, the queue manager sends the message to the destination queue manager. The destination queue manager ensures that the message is put onto the correct queue.

A queue is a *local queue* if it is managed by the same queue manager that is connected to the application. If the queue is managed by a different queue manager, it is called a *remote queue*.

In Figure 2, queues A1 and A2 are local queues to both batch application 1, CICS application 2, and to any other application connected directly to queue manager A. The figure also illustrates remote queues B1 and B2 controlled by queue manager B, and C1 and C2 controlled by queue manager C. Application 3 can put messages either to its local queues B1 and B2 controlled by MQSeries for AS/400, or to remote queues A1, A2, C1, or C2, as it requires.

Queues

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

- Whether applications can retrieve messages from the queue (get enabled)
- Whether applications can put messages onto the queue (put enabled)

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

Using queue objects

In MQSeries, there are several types of queue object. This does not mean that there are several different types of queue; essentially there is only one type of queue. Each type of queue object can be manipulated by MQSeries commands and is associated with queues in different ways:

1. A *local queue object* defines a local queue belonging to the queue manager to which the application is connected.
2. A *remote queue object* identifies a queue belonging to another queue manager. The remote queue is usually given a local definition. The definition specifies the name of the remote queue manager where the queue exists as well as the name of the remote queue itself. The information you specify when you define a remote queue object enables the queue manager to find the remote queue manager, so that any messages destined for the remote queue go to the correct queue manager.
3. An *alias queue object* enables applications to access queues by referring to them indirectly in MQI calls. When an alias queue name is used in an MQI call, the name is resolved to the name of a message queue at run time. This enables you to change the queues that applications use without changing the application in any way; you merely change the alias definition.
4. The *model queue object* defines a set of queue attributes that are used as a template for a *dynamic queue*. Dynamic queues are created by the queue manager when an application makes an open queue request specifying a queue that is a model queue. The dynamic queue that is created in this way is a local queue whose attributes are those of the model queue. You can specify a name (in full) for the dynamic queue, or the stem of a name (for example, ABC) and let the queue manager add a unique part to this, or you can let the queue manager assign a complete unique name for you.

   Dynamic queues can be of two types:

   - *Temporary*

     This type of queue is deleted when the queue is closed, and does not survive a restart of a queue manager. The queues can be used to store nonpersistent messages only.

   - *Permanent*

     This type of queue is not deleted when the queue is closed, unless the application program specifically requests it to be deleted. The queue survives a restart of a queue manager, and can be used to store persistent and nonpersistent messages.

Note: Some MQSeries products do not support all the different types of queue object given above. Refer to “MQSeries product functional comparison” on page 212 for a summary of the functions offered on the different platforms.

Queue naming convention

If your applications operate within one local system, your application designers will specify the names of each queue they want to use. These queues will be named either to some local protocol or at the programmer's whim. In the latter case, it is immediately obvious that names chosen at random by individuals could be duplicates, causing all sorts of problems.

When your applications start to deal with remote queues, the need to know and understand a naming convention becomes essential. It is no good sending a message to a remote queue manager if the name of the desired queue is not known.

For all of the above reasons, it is recommended that you develop a naming convention for your enterprise which is known to, and understood by, all your application designers and developers. The end users of your applications probably do not need to know the convention because they will usually have no need to know the queue names.

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.

Note: Not all MQSeries products support namelists. Refer to "MQSeries product functional comparison" on page 212 for a summary of the functions offered on the different platforms.

Distribution lists

A *distribution list* provides a way for an application to send a message to several destinations with a single MQPUT call. The list of destinations is supplied by the application. If more than one of these destinations uses the same transmission queue, only one copy of the message data is kept on the transmission queue, and sent down the channel.

Note: Not all MQSeries products support distribution lists. Refer to "MQSeries product functional comparison" on page 212 for a summary of the functions offered on the different platforms.

Process definitions

A *process definition object* defines an application to an MQSeries queue manager. Typically, in MQSeries, an application puts or gets messages from one or more queues and processes them. A process definition object is used for defining applications to be started by a trigger monitor. The definition includes the application ID, the application type, and application specific data.

Trigger monitors

A *trigger monitor* is an application that monitors an initiation queue (see “Initiation queues” on page 7) associated with a queue manager. When a trigger message arrives on the initiation queue, it is retrieved by the trigger monitor. Typically, the trigger monitor then starts an application that is specified in the message on the initiation queue.

Note: Only those products listed in “Level 2 products” on page 209 support process definition objects and triggering. For more information on triggering and trigger monitors see the *MQSeries Application Programming Guide*.

Channels

A channel provides a communication path. There are two types of channel, message channels and MQI channels.

Message channels

A *message channel* provides a communication path between two queue managers on the same, or different, platforms. The message channel is used for the transmission of messages from one queue manager to another, and shields the application programs from the complexities of the underlying networking protocols.

A message channel can transmit messages in one direction only. If two-way communication is required between two queue managers, two message channels are required.

In order to set up a channel, you usually define one channel definition for each end of the channel. There are four types of message channel; Sender, Server, Receiver, and Requester. On some platforms, some types of message channel can be defined automatically.

MQI channels

An *MQI channel* connects an MQSeries client to a queue manager on a server machine. It is for the transfer of MQI calls and responses only and is bidirectional. A channel definition exists for each end of the link. There are two types of MQI channel; Server-connection and Client-connection. On some platforms, some types of MQI channel can be defined automatically.

Storage classes

A *storage class* is used on MVS/ESA to map one or more queues to a page set. See the *MQSeries for MVS/ESA System Management Guide* for more information about storage classes.

Clients and servers

An *MQSeries client* is a part of an MQSeries product that can be installed on a machine without installing the full queue manager. It accepts Message Queue Interface (MQI) calls from application programs, and passes MQI requests to an *MQSeries server* that is usually executing on another processor.

The *MQSeries server* is a full queue manager, which can accept MQI calls directly from application programs that are running on the server processor; in addition, it can accept MQI requests from MQSeries clients.

This allows you to have an application that uses the MQI running on one machine, the client machine, and the queue manager itself running on a different machine.

For further information, see Chapter 7, “Introduction to MQSeries clients and servers” on page 47.

Where to find more information

For an introduction to MQSeries on your platform, see the following chapters:

- Chapter 9, “Introduction to MQSeries for AS/400” on page 63
- Chapter 14, “Introduction to MQSeries on Digital OpenVMS” on page 79
- Chapter 19, “Introduction to MQSeries for MVS/ESA” on page 99
- Chapter 27, “Introduction to MQSeries for OS/2 Warp and Windows NT” on page 135
- Chapter 32, “Introduction to MQSeries for Windows” on page 151
- Chapter 33, “Introduction to MQSeries on Tandem NSK” on page 161
- Chapter 37, “Introduction to MQSeries on UNIX systems” on page 175

Chapter 2. Introduction to distributed queuing

The MQSeries distributed queuing facility moves messages between queue managers. To allow MQSeries to do this, you must ensure that you define and install the platform-dependent connections that provide the physical links between the local queue manager and any remote queue managers that you want to exchange messages with.

This chapter describes how to plan for communication between local and remote queue managers. It contains basic information about:

How queue managers communicate

There are two ways in which a queue manager can be connected to other queue managers. These queue managers might be on the same or different platforms. The connections can allow:

- Simple transfer
- Staged transfer

Simple transfer is used between two queue managers that are connected by an MQSeries message channel.

Staged transfer is used to interconnect queue managers that are located in nodes that are not adjacent to the sending node, and can only be reached by staging through an adjacent queue manager. A remote queue manager might find that some messages received are not for its local queues but, instead, are to be passed on to another queue manager that performs the same process of delivering the messages or, passing them on again.

In each of the models introduced above, you need to define transmission queues and links to the adjacent nodes. The adjacent, or neighboring, nodes on your network must also have queue managers available that can handle your messages. Such handling places the message on one of the receiving node's local queues when that node recognizes the queue name as one of its own queues, or forwards the message to another queue manager when a queue manager name other than its own is received.

You must ensure that neighboring nodes with which your system communicates either have links to the intended destination or, at least, links to the next step toward the intended destination.

The communication between two queue managers is by means of a message channel; this consists of:

- A *transmission queue*
- A *message channel agent* for each queue manager

- A *communications link*

A *message channel agent* is a program controlling communications and is part of the queue manager.

For a detailed description of how these components allow messages to be exchanged between systems, and how to define them, refer to the *MQSeries Intercommunication* manual.

Channel speed

For a normal channel, all messages travel through the message channel at the same speed. Persistent and nonpersistent messages on the same transmission queue maintain their order relative to each other. None of the messages are lost if there is a channel failure.

On AIX, HP-UX, MVS/ESA, OS/2 Warp, OS/400, Sun Solaris, and Windows NT, you can define 'fast' channels. If a channel is defined to be fast, nonpersistent messages travel through the channel outside syncpoint. This improves the throughput of the channel, but means that nonpersistent messages are lost if there is a channel failure. (Persistent messages are never lost if a channel failure occurs.)

For a fast channel, it is possible for nonpersistent messages to jump ahead of persistent messages waiting on the same transmission queue, that is, the order of nonpersistent messages is not maintained relative to persistent messages. However the order of nonpersistent messages relative to each other is maintained. Similarly, the order of persistent messages relative to each other is maintained.

Setting up distributed queuing

To understand what you need to plan for, it is first necessary to review the queuing process.

When an application attempts to put a message to a queue, it must specify the destination queue. This is done by addressing the message to:

- An alias name
- A queue name, plus a queue manager name

If an alias name is used, the queue manager resolves that name to one of two destinations, either a local queue name, or the name of a remote queue. In the first case, the message is placed on the named local queue; in the second, the message is placed on the appropriate transmission queue.

In the case where the queue name, and the queue manager name are supplied, the queue manager name can be either that of the local queue manager, which is the default condition, or a remote queue manager name.

When a message is created by the queue manager as a result of an MQPUT call, the destination queue name, and queue manager name, are placed in the message descriptor part of the message.

When the queue manager sees that the queue manager name in a message is not its own name, it places the message on a special local queue called a transmission

queue. From the transmission queue, the message is transferred to the next node in the network by the the queue manager, using the message channel associated with that transmission queue. At the next node, the queue manager finds that the message is addressed to one of its local queues and accordingly places it on the destination queue, which can be another transmission queue giving further transmission to another node. You should note that there can be more than one transmission queue for each remote queue manager that you have defined. Refer to the *MQSeries Intercommunication* manual for more information about transmission queues.

A special case exists where a queue manager is called to handle a message that, though sent to, or defaulting to, this queue manager, does not have a known queue name. In this case the message is placed on the dead-letter queue. Special actions are required to recover or dispose of messages placed on the dead-letter queue. See “Dead-letter queues” on page 8 for more information.

To enable remote queuing, you might need to define remote queues. (This step is not essential; there are cases where the application can place the message directly to a queue manager and queue name combination.) The queue manager, when handling a request to place a message on a queue with either an alias name or a remote queue manager and queue name combination, uses the information to determine that the name refers to a remote queue manager. If so, the message is placed on a transmission queue associated with the remote queue manager. MQSeries then moves the message from the transmission queue to the remote queue manager.

You need to plan alias names for remote queues that you communicate with. You should consider the naming conventions that you wish to apply to these names, and also how you will ensure that your total network complies with your naming convention.

You must also plan the message *channels* that you will create for message transmission to remote queue managers. You might choose to define multiple channels to these remote queue managers to allow for high message traffic, different message priorities, or for different message types.

Application data conversion

The representation of character and numeric data is different on the various platforms for which MQSeries products are available. Because of this, you might need to plan for the conversion of the data within your messages, from one representation to another, when writing applications that span multiple platforms.

The message descriptor in all messages contains the coded character set identifier (CCSID) and encoding information; this identifies the representation used for the character and numeric data that is in the data portion of the message.

Some MQSeries products convert the data within messages from one representation to another, provided the format of the data conforms to one of the MQI built-in formats. This conversion can be performed at the following times:

- By a queue manager during the processing of an MQGET call, if the data conversion option is included in the call.
- By a message channel as it transmits a message to a remote queue manager, if the data conversion option is included in the channel definition. The channel converts the message data to the representation used by the platform at the receiving end of the channel.

For application defined formats that do not conform to the built-in formats, the conversion can be performed by user exit programs.

For further information about data conversion, refer to the *MQSeries Application Programming Guide*.

Assured delivery

MQSeries products are designed for assured message delivery. Processing is such that when messages are being transmitted to remote queue managers, the messages are moved in discrete transaction units, or batches, where confirmation of receipt is always obtained before a particular message is deleted at the transmitting queue manager. To achieve this, the sending and receiving ends of the link commit batches of messages in unison.

Recovering from errors

If a remote queuing error or session error occurs, error messages are sent to the local system operator or console, the local system being that where the queue manager detecting the error is running.

If any error occurs while sending or receiving a message, the transaction is terminated, and error messages are sent to both the local and remote system consoles. When you restart remote queuing, the queue manager checks for, and resolves, any in-doubt messages caused by the previous termination. An in-doubt message is one where the point of consistency prior to the message is known, but it is not known with certainty whether the new point of consistency that the message creates has been reached, or that a backout to a previous point of consistency has been completed. The application must decide what action to take to resolve the in-doubt situation.

If a message cannot be put on the queue on the remote queue manager, the message is written to the dead-letter queue on that queue manager and a report is sent back to the message sender (if so requested in the message).

Remember that it is important to ensure that there is a dead-letter queue defined for each queue manager, and that this queue is not allowed to fill completely. Otherwise, when an error is detected by the channel, the channel will stop transmission. See “Dead-letter queues” on page 8 for more information.

Where to find more information

You can find more information about distributed queuing, in the *MQSeries Intercommunication* manual.

Also, you might find it useful to read the following “Red Books” published by the IBM International Technical Support Organization:

- *Examples of Using MQSeries on S/390, RISC System/6000, AS/400, and PS/2*, GG24-4326
- *Multiplatform APPC Configuration Guide*, GG24-4485

Request these books through your IBM representative.

Chapter 3. Introduction to MQSeries security

This chapter gives an overview of the security facilities that are provided by MQSeries products. The ways of using these facilities as part of the MQSeries framework are described under “Security enabling interface (SEI)” on page 56.

This chapter contains basic information about:

- “MQSeries functional levels”
- “Security support in MQSeries” on page 22
- “Where to find more information” on page 26

MQSeries functional levels

MQSeries is provided at two functional levels:

- Level 1 function is provided by the following MQSeries products:
  - MQSeries for SCO UNIX Version 1.4
  - MQSeries for UnixWare Version 1.4.1
  - MQSeries for VSE/ESA Version 1.4
- Level 2 function is provided by the following products:
  - MQSeries for AS/400 Version 4.2
  - MQSeries for Digital OpenVMS Version 2.2
  - MQSeries for MVS/ESA Version 1
  - MQSeries for OS/2 Warp Version 5
  - MQSeries for Tandem NSK Version 2.2
  - MQSeries products on UNIX systems:
    - MQSeries for AIX Version 5
    - MQSeries for AT&T GIS UNIX Version 2.2
    - MQSeries for HP-UX Version 5
    - MQSeries for SINIX and DC/OSx Version 2.2
    - MQSeries for SunOS Version 2.2
    - MQSeries for Sun Solaris Version 5
  - MQSeries for Windows NT Version 5
  - MQSeries Three Tier for AIX
  - MQSeries Three Tier for OS/2

The differences in function are fully documented in “MQSeries product functional comparison” on page 212.

Security support in MQSeries

To understand the security functions provided by MQSeries, you need to understand the logical positioning of MQSeries relative to other components within a system and to understand the security services that are under discussion. The relative positioning of MQSeries is illustrated in Figure 3, which is an extract from the Open Blueprint, where MQSeries is represented by the Messaging and Queuing component.

Figure 3 shows that security services are separated from other components such as application services, resource managers, and communications. This separation means that security services need only be provided by one component in the node and other components simply call the appropriate services when needed.

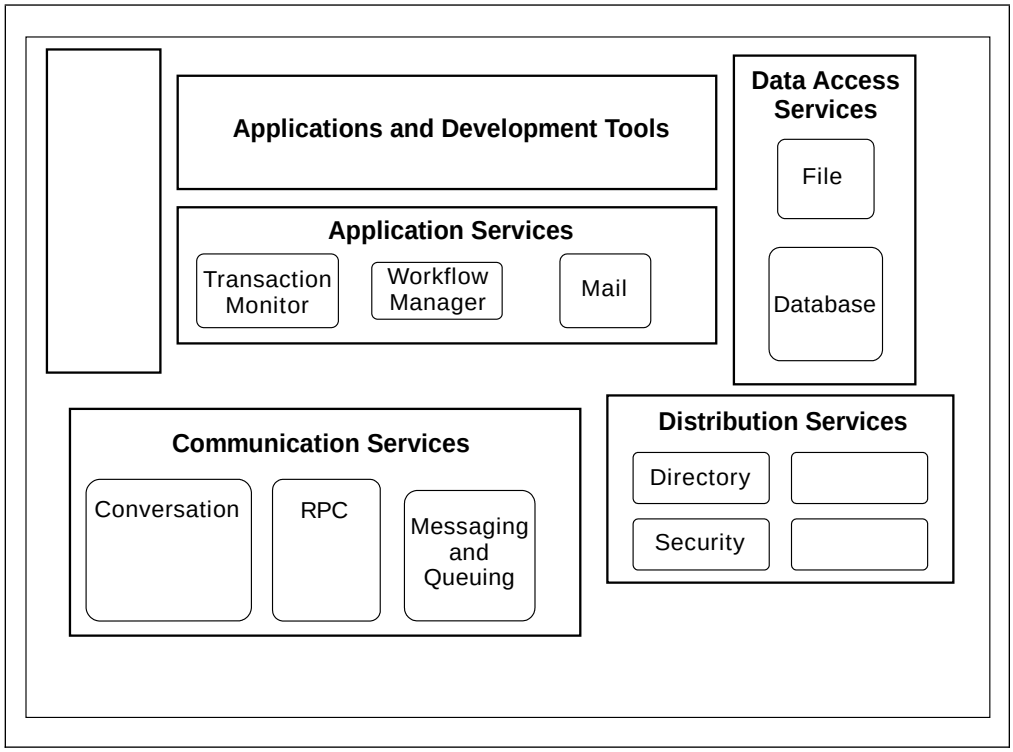

Figure 3. MQSeries and the Open Blueprint

The security services that can be provided by the (separate) security component are:

- Identification and authentication (I&A)

  This service forms the basis of many of the other services and involves the provision of a user identifier (user ID or principal) and the verification that the identifier is valid (that is, it represents the actual user and is not some intruder impersonating a valid user).

- Authorization

  This is access control and relies upon the availability of some user identifier to compare against access control lists (ACLs).

  Note that the authorization service is only useful if it is used; an intruder might attempt to bypass (and neutralize) the authorization service.

- Data confidentiality, or encryption

- Data integrity

  Even though data might be visible (that is, not encrypted), the data integrity service ensures that data is not altered.

- Non-repudiation

  This is the provision of some form of token (such as a digital signature) which guarantees that a particular piece of data originated from a particular user.

Note that although the Security component is positioned as one of the distribution services, it also provides services within the node (particularly authorization but all of the other services as well).

The way in which these security services are used by the various aspects of MQSeries is explained in the following sections.

MQSeries level 1 function

The Level 1 products do not provide any support for security. There is no attempt to recognize the user identifier associated with an application and no attempt to provide an authorization service or pass security information to other queue managers. This means that any security services that are required must be provided at the application level.

The following sections refer only to MQSeries Level 2 products.

MQSeries applications

Before an application connects to a queue manager, it will have undergone some form of I&A procedure. This might be the provision of a user ID and password or might be some more elaborate process (such as smart card identification). Alternatively, it might be that there is no requirement for the I&A procedure and, thus, no user identifier associated with the application (for example, the building that houses the system might be physically secure and user identifiers might not be considered necessary).

The consequence of this is that each application that issues MQI calls has an associated user identifier (which might be null) that is used to authorize the use of certain MQI functions (primarily MQCONN and MQOPEN) and options (such as PUT and GET) against particular objects (usually queues). This means that any application user identifier trying to access MQSeries resources must be suitably authorized.

Because the I&A procedure takes place before the application connects to the queue manager, it is the responsibility of components other than MQSeries to provide the I&A service. MQSeries is responsible only for capturing the user identifier for use in providing other security services, such as authorization. (The user identifier is captured when the application connects to the queue manager.)

Exceptions to this scope of responsibility are the OS/2 Warp and Windows NT platforms, where MQSeries provides a service to provide a user identifier for applications when they connect to the queue manager. This is called the user identifier service and is documented in the *MQSeries Programmable System Management* book. The user identifier service is part of the MQSeries framework; see “Security enabling interface (SEI)” on page 56 for further details. This service is provided because there are no security functions on these platforms.

Similarly, it is generally the responsibility of some other component to provide the authorization service, with MQSeries having responsibility for calling that service. This works satisfactorily for systems such as MVS and OS/400 where there is a standardized interface to the authorization service (for example, SAF on MVS). For the OS/2 Warp, Windows NT, and UNIX platforms, however, there is no standard authorization service or interface provided and so MQSeries provides its own interface. This is called the authorization service and is documented in the *MQSeries Programmable System Management* book. The authorization service is part of the MQSeries framework; see "Security enabling interface (SEI)" on page 56 for further details.

For MQSeries for Digital OpenVMS, MQSeries for Tandem NSK, and MQSeries on UNIX systems, there is an additional component written to this interface that provides authorization service. This is called the object authority manager (OAM) and it restricts access to MQSeries objects based upon the ACLs that it manages. The OAM is documented in the *MQSeries for Digital OpenVMS System Management Guide*, the *MQSeries for Tandem NonStop Kernel System Management Guide*, and the *MQSeries System Administration* manual. The object authority manager is part of the MQSeries framework; see "Security enabling interface (SEI)" on page 56 for further details.

MQSeries messages

The basic function of MQSeries is to pass messages between applications. The message header (the message descriptor, MQMD) contains a field (named *UserIdentifier*) where a user identifier can be placed, allowing the application that gets the message to know from which user the message originated. The user identifier can be placed in the MQMD in one of three ways:

- The user identifier from a previous message (that is, one for which an MQGET has been performed) can be passed to a subsequent message. This is known as passing the security context.
- A suitably authorized (that is, trusted) application can place *any* user identifier in the field.
- If neither of the above is used, MQSeries automatically places in this field the user identifier of the application that did the MQPUT.

Therefore, this aspect of MQSeries operation provides an identification service (associating a user identifier with the message); it does *not* provide an authentication service. It is currently the responsibility of MQSeries applications both to provide any required authentication token (on the MQPUT side) and to verify that token (on the MQGET side).

Note also that, when an application does an MQGET for a message, there is no attempt to reset the application's user identifier to that contained in the message header. This function is called context management and is not supported by MQSeries.

Point-to-point security

In addition to providing services as a local resource manager, a queue manager also provides distributed queuing, enabling messages to be distributed around a network of queue managers. This distributed messaging function is provided by means of MQSeries channels. Each channel is composed of a pair of message channel agent programs (MCAs), which provide the protocol for assured, once-only message delivery using an underlying transport mechanism to exchange messages.

When the two MCAs establish communication, it might be necessary for each to verify the identity of the other. This would be the case if one queue manager did not trust the connection or the identity of the partner queue manager (for example, if they were owned by separate enterprises). This verification can be accomplished in one of the following ways:

1. Some transport mechanisms (in particular APPC) provide security features such as session authentication. Note that this provides verification of the partner system (the partner logical unit), rather than the partner application (the MCA) but this might satisfy the security requirements of the queue manager.
2. The MCAs each provide a security exit point which can be used to call user-written security exits for the exchange of user identifiers and associated authentication tokens (password, ticket, and so on). This allows each MCA to verify the identity of its partner.

   Using the MCA security exit allows the channel to be independent of the underlying transport mechanism and to provide a consistent service across many transports. This is especially important when providing a service (like security) that is available only on a limited set of transports.

   This aspect of MQSeries operation provides support for the I&A service. If required, the security exit can use the central security services to provide authentication tokens but this is not a requirement. See “Security enabling interface (SEI)” on page 56 for more information about MCA exits.
3. On AIX, HP-UX, OS/2 Warp, Sun Solaris, Windows NT, and Windows 95, MQSeries provides exits relating to DCE security. Source code is also provided to help you understand the program, and to assist you in creating your own.

Note that it is possible to use any or all of the above mechanisms for point-to-point security, enabling an MCA to be assured that it is exchanging messages with the correct queue manager.

End-to-end security

End-to-end security refers to security services that can be provided when a message is PUT by an application and to the corresponding services that are available when the target application performs a GET. The relevant services are identification and authentication (possibly across the network), data confidentiality, data integrity, and non-repudiation.

MQSeries is not responsible for the provision of these services but it is responsible for providing appropriate interfaces to call these services.

Today, the only aspect of end-to-end security that is (directly) supported by MQSeries is Identification, where a user identifier can be placed in a message header. MQSeries does not provide (direct) support for any of the other services. However, these services can be implemented in either MQSeries application programs or the MCA message exit, which is a customer exit invoked each time a message is passed between two queue managers.

Where to find more information

For information about MQSeries security on your platform, see the following chapters:

- Chapter 11, “Security planning for MQSeries for AS/400” on page 69
- Chapter 16, “Security planning for MQSeries for Digital OpenVMS” on page 87
- Chapter 22, “Security planning for MQSeries for MVS/ESA” on page 113
- Chapter 29, “Security planning for MQSeries for OS/2 Warp and Windows NT” on page 145
- Chapter 35, “Security planning for MQSeries for Tandem NSK” on page 167
- Chapter 39, “Security planning for MQSeries on UNIX systems” on page 185

For information about the security enabling interface, see Chapter 8, “Introduction to the MQSeries Framework” on page 53.

Chapter 4. Introduction to MQSeries recovery concepts

MQSeries products provide facilities for applications to be able to keep related sets of data consistent when changes are made to them. This chapter describes the background concepts of recovery and restart that you will need to understand before going on to the later chapters in this book.

This chapter gives basic information about:

How changes are made to data

You need to understand how MQSeries queue managers interact with other programs to keep all the data consistent. This section discusses *units of recovery*.

Units of recovery

In a single queue manager, a *unit of recovery* is a piece of work that changes data from one point of consistency to another. A *point of consistency* (also called a *syncpoint* or *commit point*) is a moment at which all the recoverable data that an application program accesses is consistent. This means that a unit of recovery begins with an attempt to change data and ends at a point of consistency. An example of a unit of recovery within an application program is shown in Figure 4.

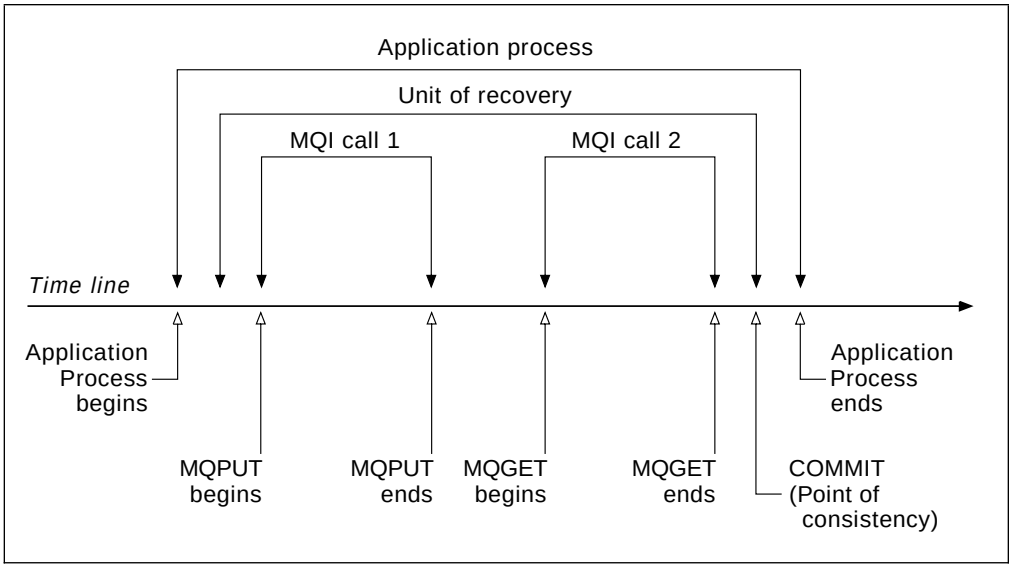

Figure 4. A unit of recovery within an application

In this example, the application process makes changes to queues at *MQI call 1* and *MQI call 2*. The application process can include a number of units of recovery or just one, but any complete unit of recovery ends with a commit point.

For example, a bank transaction might transfer funds from account A to account B. First, a debit program subtracts the amount from account A. Next, account B must

be credited with the amount. After subtracting the amount from account A, the two accounts are inconsistent and the queue manager cannot commit. The next step in the process is for the debit program to put a message on a queue to the credit program, giving information about the amount of money and the account to be credited. When the credit program gets the message from the queue, it performs the credit to account B and informs the debit program that it has completed the task. The data is now consistent. The application can announce a point of consistency and make the changes visible to other applications.

A point of consistency should be created by an application before it terminates. It can do this by issuing a commit call. Normal termination of an application on some platforms automatically causes a point of consistency, but you should verify that this will occur on your platform.

If an error occurs within a unit of recovery, you can use the queue manager to remove changes to data, returning the data to its state at the start of the unit of recovery; that is, the queue manager undoes the work.

Some systems have syncpoint facilities. In these cases it is the system function that causes a syncpoint to be created for both that system's data, and the data that is being managed by MQSeries queue managers, simultaneously.

MQSeries products can operate with a number of others, to provide coordination between MQSeries and non-MQSeries resources. The MQSeries products where this is possible include:

Table 2. MQSeries and cooperating products

MQSeries product	**Cooperating products**
MQSeries for AIX	CICS for AIX, Transaction Server for AIX, ENCINA, TUXEDO
MQSeries for AS/400	CICS for OS/400
MQSeries for AT&T GIS UNIX	TUXEDO
MQSeries for HP-UX	CICS for HP 9000, ENCINA, TUXEDO
MQSeries for MVS/ESA	CICS for MVS/ESA, CICS/MVS, IMS/ESA
MQSeries for OS/2 Warp	CICS for OS/2, Transaction Server for OS/2
MQSeries for SINIX and DC/OSx	TUXEDO
MQSeries for SunOS	TUXEDO
MQSeries for Sun Solaris	CICS for Solaris, ENCINA, TUXEDO
MQSeries for Windows NT	CICS for Windows NT, Transaction Server for Windows NT, TUXEDO, ENCINA

How consistency is maintained

If the data being managed by a queue manager is to be consistent with the data in other subsystems, any data changed in one must be matched by a change in the others. Before one system commits the changed data, it must know that the other system, or systems, can make the corresponding changes. So, the systems must communicate.

During a *two-phase commit* one subsystem coordinates the process. That subsystem is called the *coordinator*; the others are the *participants*. On AIX, HP-UX, OS/2 Warp, Sun Solaris, and Windows NT, MQSeries can be either the coordinator or a participant. On other platforms, CICS or IMS is always the coordinator in interactions with MQSeries, and MQSeries is always a participant. The coordinator first checks that all participants are ready to perform the commit; that is phase one. When the coordinator receives positive replies from the participants that they are ready to commit, it issues the commit request, this being phase 2. Together with the coordinator, all participants now commit the changed data. All data is once again consistent.

During a *one-phase commit* there is no coordinator as such in the interactions. When the commit call is issued, the response to the call is that:

- The commit was successful
- The call was partially complete
- The call failed

It is up to the application issuing the call to decide what further action, if any, needs to be taken.

When the queue manager is restarted after an abnormal termination, it must determine whether to commit or to back out units of recovery that were active at the time of the error. For certain units of recovery, the queue manager has enough information to make the decision. For others, it does not, and must get information from the coordinator when the connection is reestablished.

Syncpointing cannot occur directly across remote links. Instead, use is made of the assured message delivery feature of the MQSeries products. A commit request is sent in a message to a remote system by an application with the assurance that the message will be delivered. The application designers must ensure that they create, and respond to, commit requests. They must also allow for situations where the commit cannot be accepted, and respond accordingly to the commit requester.

An example of the problems of remote commits is when the remote system is not available. The messages to it will remain on the transmission queues until they can be delivered. When delivery eventually occurs, any desired commitments can then be made. In this situation, we are dealing with time independent applications. It is for the application designers to consider how, and when, they require the data shared between their remote systems to be consistent.

MQSeries as a transaction manager

On some platforms, MQSeries can act as a transaction manager and will coordinate updates made by external resource managers within MQSeries units of work. These external resource managers must comply to the X/Open XA interface.

Table 3 shows the MQSeries platforms that can act as a transaction manager, and the XA-compliant database managers they support. Figure 5 through Figure 7 on page 31 show how you must configure your queue manager and database servers to enable this.

Table 3. XA-compliant database managers supported by MQSeries

Platform	DB2	Oracle
AIX	√	√
HP-UX	√	√
OS/2 Warp	√	
Sun Solaris	√	√
Windows NT	√	√

The following figures describe the possible configurations for your queue manager and database. The configuration shown in Figure 5 is allowed:

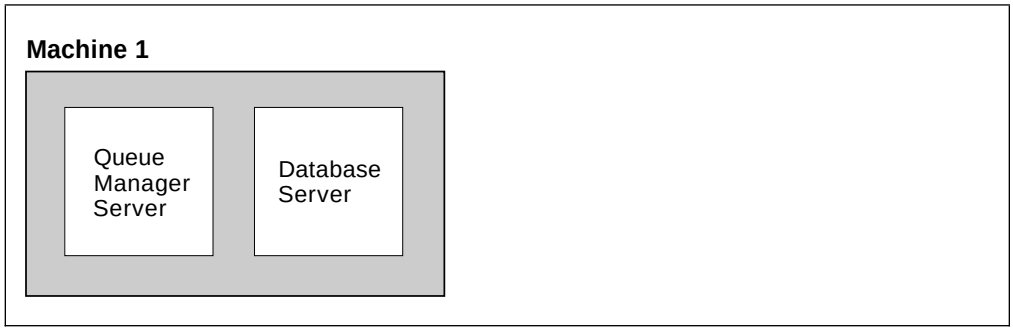

Figure 5. Queue manager and database server configuration 1. Queue manager server and database server on the same machine

The configuration shown in Figure 6 is also allowed provided that you use appropriate database connection features (for example, CAE/DDCS for DB/2):

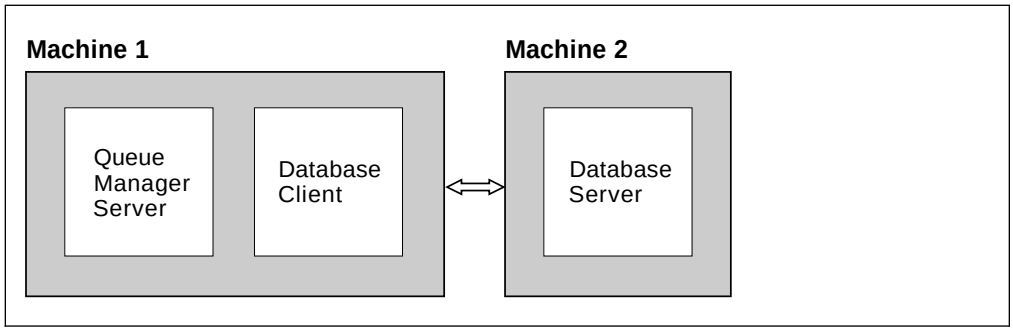

Figure 6. Queue manager and database server configuration 2. Queue manager server and database client on the same machine

The configuration shown in Figure 7 on page 31 is not allowed because coordination can be provided only by the MQSeries server, not the client:

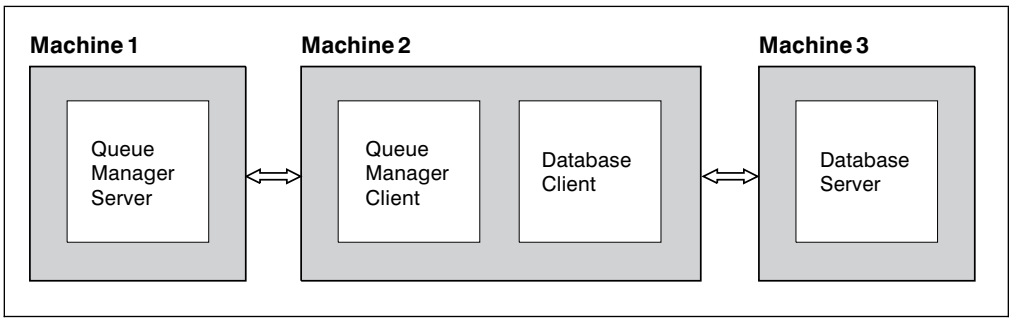

Figure 7. Queue manager and database server configuration 3. Queue manager server and database server on different machine

Where to find more information

For information about MQSeries recovery on your platform, see the following chapters:

- Chapter 10, “Backup and recovery planning for MQSeries for AS/400” on page 67
- Chapter 15, “Backup and recovery planning for MQSeries for Digital OpenVMS” on page 83
- Chapter 21, “Backup and recovery planning for MQSeries for MVS/ESA” on page 109
- Chapter 28, “Backup and recovery planning for MQSeries for OS/2 Warp and Windows NT” on page 141
- Chapter 34, “Backup and recovery planning for MQSeries for Tandem NSK” on page 165
- Chapter 38, “Backup and recovery planning for MQSeries on UNIX systems” on page 181

Chapter 5. Introduction to MQSeries administration

This chapter gives a summary of the administration facilities that are provided by the MQSeries products.

Administration of an application that uses MQSeries products is performed by a system administrator, system programmer, or computer operator who has the appropriate authority. It is the administrator's responsibility to monitor the MQSeries products, and the resources that they are using, and make any changes that might be necessary to keep the applications running without problems.

This chapter has the following sections:

Some of the facilities described in this chapter are not supported by all MQSeries products.

Introducing commands

MQSeries products provide a set of commands, command interfaces, and utilities that provide messaging and queuing administration functions, including, for example:

Queue administration	Creating and deleting queues
Channel administration	Stopping or starting channels
Security administration	Changing the access authority to queues

These functions are provided by *commands* in MQSeries, which are processed, by the queue manager, in the following ways:

- **Entered by the system administrator**.

  A command is entered by a system administrator and processed immediately by the queue manager. A response is given to the administrator.

  The method used to enter the commands is platform dependent, and is the normal method that is used for entering commands on the platform. This could be a command console, a command line, a display panel utility, or some other technique.

- **Stored as a list of commands**.

  The commands are stored as a list in a file. Later, the administrator can cause the queue manager to read the commands in the file, process them, and generate responses.

- **Stored as messages in a queue**.

  An application program generates a command, and puts it to a queue manager *command queue*, using the MQPUT call. The queue manager gets the

message off the queue, processes the command that is contained in the message, and generates a response.

Formats of command

The MQSeries products provide the following types of command:

- **MQSeries commands (MQSC)** - human-readable
- **Programmable Command Format (PCF) commands** - machine-readable

Both sets of commands perform similar functions, but are intended for use in different situations.

In addition to the MQSC and PCF commands, the MQSeries products provide *control commands*. These are the commands used by the system administrator to pass control information to the queue manager, and are designed to be compatible with the other administrative commands used on a particular platform. Because of the differences in the way that administrative commands are processed by the various operating systems in which the MQSeries products operate, the technique used to enter control commands varies with the particular MQSeries product.

MQSC commands

MQSeries commands (MQSC) provide a uniform method of issuing commands in a human-readable form across MQSeries platforms. They are intended for use in those situations where people have to be able to read the commands, for example, whenever commands are entered through a command line. The responses to these commands are also human-readable, however the content and format depends on the platform in which they are used.

The commands can be issued from the following sources:

- Entered through the command line
- Stored in a file, and processed by the queue manager
- Stored within messages on a command queue

The commands are described in the *MQSeries Command Reference* manual.

Programmable Command Format commands

MQSeries Programmable Command Format (PCF) commands are intended for use by application programs that provide facilities for the administration of queue managers and the resources that they are using (for example, queues and channels), from a single point within the network. The format of these commands, and their responses, is independent of the environment in which they are used.

PCFs define commands and reply messages that can be exchanged between a program and any queue manager in a network.

You can use PCF commands in a systems-management application program for administration of MQSeries objects: queue managers, process definitions, queues, and channels. The application can operate from a single point in the network to communicate command and reply information with any queue manager, local or remote, via the local queue manager.

It is possible, by using the PCF **escape** command, to include the MQSC command as character strings within a PCF command.

PCF commands are described in the *MQSeries Programmable System Management* manual.

Command queues

Some MQSeries products have a command queue that accepts administration messages containing commands, called *command messages*. The queue manager has a *command server* that processes the command messages from the command queue.

Command messages can be placed on the queue by any local or remote application program. The command messages are processed by the command server, and a response is returned to the application program using the name of the reply-to queue that is contained within the original command message. PCF commands and reply messages are sent and received using the MQPUT and MQGET calls defined in the Message Queue Interface (MQI).

There are two types of command queue, each with system-defined queue names.

1. SYSTEM.ADMIN.COMMAND.QUEUE

   This queue accepts PCF commands only, including the **escape** command. This type of queue is supported by the following:

   - MQSeries for AS/400
   - MQSeries for Digital OpenVMS
   - MQSeries for OS/2 Warp
   - MQSeries for Tandem NSK
   - MQSeries on UNIX systems
   - MQSeries for Windows NT
   - MQSeries for Windows (Version 2.1)

2. SYSTEM.COMMAND.INPUT

   This queue accepts MQSC commands only. This type of queue is supported by MQSeries for MVS/ESA only.

The MQSeries products that are not mentioned above do not have command queues.

Administration application programs

MQSeries products provide application programs, called *administration utilities*, that allow the system administrator on one processor to control the MQSeries product that is running on that processor. Some of these utilities also allow the administrator to control MQSeries products that are running on other processors, so-called *remote administration*.

In those situations where no administration utility is available, you can write your own application program to provide those functions. This application can manage any queue manager (local or remote) that has a command queue, by putting command messages to that queue using the MQPUT call. When a command is processed, a response message is returned to the reply-to queue that your application program specified in the original command message.

MQSeries product administration facilities

Details of the control commands and administration facilities provided by some of the MQSeries products are given in later sections of this manual, as follows:

- **MQSeries for AS/400** – see Chapter 12, “Administration of MQSeries for AS/400” on page 71, and the *MQSeries for AS/400 Administration Guide* for more detail.
- **MQSeries for Digital OpenVMS** – see Chapter 17, “Administration of MQSeries for Digital OpenVMS” on page 91, and the *MQSeries for Digital OpenVMS System Management Guide* for more detail.
- **MQSeries for MVS/ESA** – see Chapter 23, “Administration of MQSeries for MVS/ESA” on page 117, and the *MQSeries for MVS/ESA System Management Guide* for more detail. MQSeries for MVS/ESA does not support PCF commands.
- **MQSeries for OS/2 Warp** and **MQSeries for Windows NT** – see Chapter 30, “Administration of MQSeries for OS/2 and Windows NT” on page 147, and the *MQSeries System Administration* manual for more detail.
- **MQSeries for Tandem NonStop Kernel** – see Chapter 36, “Administration of MQSeries for Tandem NSK” on page 171, and the *MQSeries for Tandem NonStop Kernel System Management Guide* for more detail.
- **MQSeries on UNIX systems** – see Chapter 40, “Administration of MQSeries on UNIX systems” on page 189, and the *MQSeries System Administration* manual for more detail.
- **MQSeries for Windows** – see Chapter 32, “Introduction to MQSeries for Windows” on page 151, and the *MQSeries for Windows User’s Guide* for more detail.

A tabular summary of the commands supported by these products is given in the “Command summary” on page 37 section.

Administration for other MQSeries products

The MQSeries products that are not mentioned above have utility programs that use display panels to provide a user-friendly interface for the administration of the product. Details of these programs, and the administration facilities that they provide, can be found in the *User Guide* that is available for each of the products.

None of these products has an administration queue to which commands (MQSC or PCF) can be sent. Therefore administration of these products by local or remote application programs is not possible.

Command summary

The following tables show how the various command formats in MQSeries relate to each other. The command formats available are:

- Programmable command format (PCF) commands
- MQSeries (MQSC) commands
- MQSeries for OS/400 CL commands
- Control commands for MQSeries products on distributed platforms, that is, MQSeries on UNIX systems, MQSeries for Digital OpenVMS, MQSeries for Tandem NonStop Kernel, MQSeries for OS/2 Warp, and MQSeries for Windows NT

Notes:

1. The PCF commands are not supported on MVS/ESA.
2. Unless otherwise specified, the MQSC commands are supported on all platforms.
3. An empty cell indicates that there is no equivalent command in the specified format.
4. In MQSeries for Tandem NonStop Kernel control commands are entered in lowercase.

Table 4 (Page 1 of 2). Commands for queue manager administration

Operation	PCF	MQSC	OS/400 CL	Commands for distributed platforms
Change Queue Manager attributes	Change Queue Manager	ALTER QMGR DEFINE MAXSMSGS (See note 1)	CHGMQM	
Display Queue Manager attributes	Inquire Queue Manager	DISPLAY QMGR DISPLAY MAXSMSGS (See note 1)	DSPMQM	
Connect a Queue Manager			CCTMQM	
Create a Queue Manager			CRTMQM	CRTMQM
Delete a Queue Manager			DLTMQM	DLTMQM
Disconnect a Queue Manager			DSCMQM	
Stop a Queue Manager		STOP QMGR (See note 1)	ENDMQM	ENDMQM
Ping a Queue Manager	Ping Queue Manager	PING QMGR (See note 2)		
Start a Queue Manager		START QMGR (See note 1)	STRMQM	STRMQM
Add a Queue Manager to Windows NT Service Control Manager				SCMMQM (See note 3)
Start an MQSeries trial period				SETMQTRY (See note 4)

Table 4 (Page 2 of 2). Commands for queue manager administration

Operation	PCF	MQSC	OS/400 CL	Commands for distributed platforms
Enroll an MQSeries production licence				SETMQPRD (See note 4)

Notes:

1. Applies on MVS/ESA only
2. Does not apply on MVS/ESA
3. Applies on Windows NT only
4. Applies on V5.0 of MQSeries for AIX, HP-UX, OS/2 Warp, Sun Solaris, and Windows NT

Table 5 (Page 1 of 2). Commands for queue administration

Operation	PCF	MQSC	OS/400 CL	Commands for distributed platforms
Change queue attributes	Change Queue	ALTER QALIAS ALTER QLOCAL ALTER QMODEL ALTER QREMOTE	CHGMQMQ	
Clear a queue	Clear Queue	CLEAR QLOCAL (See note 1) The following sequence: DELETE QLOCAL(x), DEFINE QLOCAL(x) or the following sequence: DEFINE QLOCAL(y) LIKE(x), DELETE QLOCAL(x), DEFINE QLOCAL(x) LIKE(y), DELETE QLOCAL(y)	CLRMQMQ	
Copy a queue definition	Copy Queue	DEFINE QALIAS(x) LIKE(y) DEFINE QLOCAL(x) LIKE(y) DEFINE QMODEL(x) LIKE(y) DEFINE QREMOTE(x) LIKE(y)	CPYMQMQ	
Create a queue	Create Queue	DEFINE QALIAS DEFINE QLOCAL DEFINE QMODEL DEFINE QREMOTE	CRTMQMQ	
Delete a queue	Delete Queue	DELETE QALIAS DELETE QLOCAL DELETE QMODEL DELETE QREMOTE	DLTMQMQ	
Display queue attributes	Inquire Queue	DISPLAY QUEUE DISPLAY QALIAS (See note 2) DISPLAY QLOCAL (See note 2) DISPLAY QMODEL (See note 2) DISPLAY QREMOTE (See note 2)	DSPMQMQ	
Display queue names	Inquire Queue Names	DISPLAY QUEUE	WRKMQMQ	
Work with a queue			WRKMQMQ	
Work with messages			WRKMQMMSG	

Table 5 (Page 2 of 2). Commands for queue administration

Operation	PCF	MQSC	OS/400 CL	Commands for distributed platforms
Reset queue statistics	Reset Queue Statistics (See note 3)			

Notes:

1. Does not apply on MVS/ESA
2. Applies on AIX, HP-UX, MVS/ESA, OS/2, Sun Solaris, and Windows NT only
3. Does not apply on Tandem NSK

Table 6. Commands for process definition administration

Operation	PCF	MQSC	OS/400 CL	Commands for distributed platforms
Change process attributes	Change Process	ALTER PROCESS	CHGMQMPRC	
Copy a process	Copy Process	DEFINE PROCESS(x) LIKE(y)	CPYMQMPRC	
Create a process	Create Process	DEFINE PROCESS	CRTMQMPRC	
Delete a process	Delete Process	DELETE PROCESS	DLTMQMPRC	
Display process attributes	Inquire Process	DISPLAY PROCESS	DSPMQMPRC	
Display process names	Inquire Process Names	DISPLAY PROCESS	WRKMQMPRC	
Work with a process			WRKMQMPRC	

Table 7. Commands for namelist administration (MVS/ESA only)

Operation	PCF	MQSC	OS/400 CL	Commands for distributed platforms
Alter a namelist		ALTER NAMELIST		
Copy a namelist		DEFINE NAMELIST(x) LIKE(y)		
Define a namelist		DEFINE NAMELIST		
Delete a namelist		DELETE NAMELIST		
Display a namelist		DISPLAY NAMELIST		

Table 8 (Page 1 of 2). Commands for channel administration

Operation	PCF	MQSC (See note 1)	OS/400 CL	Commands for distributed platforms
Change channel attributes	Change Channel	ALTER CHANNEL	CHGMQMCHL	
Copy channel attributes	Copy Channel	DEFINE CHANNEL (x) LIKE (y)	CPYMQMCHL	
Create a channel	Create Channel	DEFINE CHANNEL	CRTMQMCHL	
Delete a channel	Delete Channel	DELETE CHANNEL	DLTMQMCHL	
Display a channel	Inquire Channel	DISPLAY CHANNEL	DSPMQMCHL	
Display channel names	Inquire Channel Names	DISPLAY CHANNEL	WRKMQMCHL	

Table 8 (Page 2 of 2). Commands for channel administration

Operation	PCF	MQSC (See note 1)	OS/400 CL	Commands for distributed platforms
Display channel status	Inquire Channel Status	DISPLAY CHSTATUS	WRKMQMCHST	
Display distributed queuing		DISPLAY DQM (See note 2)		
Ping a channel	Ping Channel	PING CHANNEL	PNGMQMCHL	
Reset a channel	Reset Channel	RESET CHANNEL	RSTMQMCHL	
Resolve a channel	Resolve Channel	RESOLVE CHANNEL	RSVMQMCHL	
Start a channel	Start Channel	START CHANNEL	STRMQMCHL	RUNMQCHL
Start a channel initiator	Start Channel Initiator	START CHINIT (See note 2)	STRMQMCHLI	RUNMQCHI
Start a channel listener	Start Channel Listener	START LISTENER (See note 3)	STRMQMLSR	RUNMQLSR (See note 4)
Stop a channel	Stop Channel	STOP CHANNEL	ENDMQMCHL	
Stop a channel initiator		STOP CHINIT (See note 2)		
Stop a channel listener		STOP LISTENER (See note 2)		ENDMQLSR (See note 5)
Work with channels			WRKMQMCHL	
Work with channel status			WRKMQMCHST	

Notes:

1. Does not apply on MVS/ESA if you are using CICS for distributed queuing
2. Applies on MVS/ESA only
3. Does not apply on UNIX systems, Digital OpenVMS, or Tandem NSK
4. Applies on OS/2, Windows NT, Digital OVMS, and Tandem NSK only
5. Applies on OS/2 and Windows NT only

In MQSeries for Tandem NonStop Kernel, use TS/MP or the control command **runmqlsr** to start TCP/IP channel listeners.

Table 9. Commands for security administration

Operation	PCF	MQSC (See note 1)	OS/400 CL	Commands for distributed platforms
Display object authority			DSPMQMAUT	DSPMQAUT
Grant object authority			GRTMQMAUT	SETMQAUT
Revoke object authority			RVKMQMAUT	SETMQAUT
Alter security options		ALTER SECURITY		
Display security settings		DISPLAY SECURITY		DSPMQAUT
Refresh security		REFRESH SECURITY		
Set a reverification flag		RVERIFY SECURITY		

Note:

1. Applies on MVS/ESA only

Table 10 (Page 1 of 2). Commands for system-dependent function

Operation	PCF	MQSC (see note 1)	OS/400 CL	Commands for distributed platforms
Alter trace parameters		ALTER TRACE		

Table 10 (Page 2 of 2). Commands for system-dependent function

Operation	PCF	MQSC (see note 1)	OS/400 CL	Commands for distributed platforms
Display trace activity		DISPLAY TRACE		
Start a trace		START TRACE	TRCMQM	STRMQTRC (See note 2)
Stop a trace		STOP TRACE	TRCMQM	ENDMQTRC (See note 2)
Archive a log		ARCHIVE LOG		
Define a buffer pool		DEFINE BUFFPOOL		
Define a page set		DEFINE PSID		
Display page set information		DISPLAY USAGE		
Alter a storage class		ALTER STGCLASS		
Define a storage class		DEFINE STGCLASS		
Delete a storage class		DELETE STGCLASS		
Display storage class information		DISPLAY STGCLASS		
Display a thread		DISPLAY THREAD		
Recover a bootstrap data set		RECOVER BSDS		
Resolve in-doubt threads		RESOLVE INDOUBT		
Display the command server		DISPLAY CMDSERV	DSPMQMCSVR	DSPMQCSV
Start the command server		START CMDSERV	STRMQMCSVR	STRMQCSV
Stop the command server		STOP CMDSERV	ENDMQMCSVR	ENDMQCSV
Reset an IMS transaction pipe		RESET TPIPE		
Display an object name			DSPMQMOBJN	
Start a service job			STRMQMSRV	
End a service job			ENDMQMSRV	
Start the administrator			STRMQMADM	
Record an object image			RCDMQMIMG	RCDMQIMG (See note 3)
Recreate an object			RCRMQMOBJ	RCRMQOBJ (See note 3)
Display MQSeries formatted trace output				DSPMQTRC (See note 4)
Dump contents of MQSeries log				DMPMQLOG (See note 5)

Notes:

1. Applies on MVS/ESA only
2. Does not apply on AIX
3. Does not apply on Tandem NSK
4. Applies on AT&T, HP-UX, SINIX and DC/OSx, SunOS, Sun Solaris, and Tandem NSK
5. Applies on V5.0 of MQSeries for AIX, HP-UX, OS/2 Warp, Sun Solaris, and Windows NT

In MQSeries for Tandem NonStop Kernel, as an alternative to the control commands dspmqcsv, strmqcsv, and endmqcsv, you may use PATHCOM commands.

Table 11. Other control commands in MQSeries for Tandem NonStop Kernel	
Operation	**Commands**
Alter queue volume	altmqfls
Perform housekeeping on a queue manager	cleanqm
Convert V1.5.1 queues and channels to V2.2	cnv1520
Convert V1.5.1 messages to V2.2	cnvmsgs
Convert client channel definition table	cnvclchl
Install MQSeries for Tandem NonStop Kernel	instmqm
Run dead-letter queue handler	runmqdlq
Note: As an alternative to the control command **runmqtrm**, you may use PATHCOM commands. There are no MQSC or PCF equivalents of commands in this group.	

Where to find more information

For information about MQSeries administration on your platform, see the following chapters:

- Chapter 12, “Administration of MQSeries for AS/400” on page 71
- Chapter 17, “Administration of MQSeries for Digital OpenVMS” on page 91
- Chapter 23, “Administration of MQSeries for MVS/ESA” on page 117
- Chapter 30, “Administration of MQSeries for OS/2 and Windows NT” on page 147
- Chapter 32, “Introduction to MQSeries for Windows” on page 151
- Chapter 36, “Administration of MQSeries for Tandem NSK” on page 171
- Chapter 40, “Administration of MQSeries on UNIX systems” on page 189

Chapter 6. Introduction to MQSeries instrumentation events

You can use the MQSeries instrumentation events to monitor the operation of queue managers. This chapter tells you what these events are, and describes how they can be used for system measurement and system management purposes.

The chapter contains these sections:

Instrumentation events are supported by the following:

- MQSeries for AS/400
- MQSeries for Digital OpenVMS
- MQSeries for MVS/ESA
- MQSeries for OS/2 Warp
- MQSeries for Tandem NSK
- MQSeries on UNIX systems
- MQSeries for Windows NT
- MQSeries for Windows Version 2.1

Monitoring queue managers

An *instrumentation event* causes a special message, called an *event message*, to be generated by a queue manager whenever a set of predefined conditions occurs within the execution of your application.

You can write a system-monitoring application that collects event messages from many queue managers on different platforms, analyses them, and presents them to an administrator in summary form. This would allow you to monitor all the MQSeries products in your system from a single node.

Instrumentation events also enable applications acting as agents for other administration networks, for example NetView, to monitor events and create the appropriate alerts.

What is an instrumentation event?

In MQSeries an instrumentation event is a logical combination of conditions that is detected by a queue manager. The result of such an event is that the queue manager puts a special message, called an *event message*, on an event queue.

For example, the conditions giving rise to a *queue full* event are:

- Queue full events are enabled for a specified queue, and
- An application issues an MQPUT call to put a message on that queue, but the call fails because the queue is full

Other conditions can also give rise to instrumentation events. For example:

- A threshold for the number of messages on a queue is reached

- A channel instance is started or stopped
- An application attempts to open a queue specifying a user ID that is not authorized

Some instrumentation events must be enabled before they can be generated; this is described in “Enabling and disabling events” on page 45.

What types of event are there?

MQSeries events can be categorized as follows:

Queue manager events
These events are related to the definitions of resources within queue managers. For example, when an application attempts to put a message to a queue that does not exist.

Performance events
These events are notifications that a threshold has been reached by a resource. For example, when a queue depth threshold has been reached or, following a get, when the queue was not serviced within a predefined time limit.

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

Event notification through event queues

When an event occurs, the queue manager puts an event message on the appropriate event queue. The event message contains information about the event that you can retrieve by writing a suitable MQSeries application program that:

- Gets the message from the queue
- Processes the message to extract the event data

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.

Table 12. Event queue contents

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

You can define event queues as either local or remote queues. If you define all your event queues as remote queues on the same queue manager, you can centralize your monitoring activities.

You must not define event queues as transmission queues, or initiation queues, because event messages have formats that are incompatible with the formats of messages required for those queues.

Using triggered event queues

You can set up the event queues with triggers, so that when an event is generated, the event message being put onto the event queue starts a (user-written) monitoring application. This application can process the event messages and take appropriate action. For example, certain events might require that an operator be informed, while other events might start off an application that performs some administration tasks automatically.

When an event queue is unavailable

If an event occurs when the event queue is not available, the event message is lost. For example, if you do not define an event queue for a category of event, all event messages for that category will be lost. The event messages are *not*, for example, saved on the dead-letter queue. An event queue might be unavailable for many different reasons, for example:

- The queue has not been defined
- The queue has been deleted
- The queue is full
- The queue has been put-inhibited

The absence of an event queue does not prevent the event from occurring. For example, after a performance event, the queue manager changes the queue attributes and might, depending on the event, reset the queue statistics. This happens whether or not the event message is put on the performance event queue.

Enabling and disabling events

You can enable and disable events by specifying the appropriate values for queue manager or queue attributes, or both, depending on the type of event. Commands are provided with the appropriate MQSeries products to allow you to set these attributes.

Note: Attributes related to events for both queues and queue managers can be set and inquired upon by command only. They are not supported by the MQI functions MQSET and MQINQ.

How you enable and disable events depends on the category of the event:

- Queue manager events are enabled by setting attributes on the queue manager.
- Performance events as a whole must be enabled on the queue manager, otherwise no performance events can occur. Then, you enable the specific performance events by setting the appropriate queue attribute. You also have to specify the conditions that give rise to the event.
- Channel events do not require enabling, they occur automatically. Similarly, channel events cannot be disabled.

Format of event messages

Event messages contain information about the event and its origin. Typically, these messages are processed by a system-management application program that is tailored to meet the requirements of the site at which it runs.

As with all MQSeries messages, an event message has two parts: a message descriptor, and the message data.

Message descriptor
The descriptor of an event message is a standard descriptor, as defined in the *MQSeries Application Programming Reference*. Some of the values in the descriptor might be of particular interest to a system-monitoring application, for example, the date and time when the event message was put on the event queue.

Event message data
The message data specifies:

- That the message is an event message.
- The category of the event, that is, whether the event is a queue manager, performance, or channel event.
- A reason code specifying the cause of the event.
- Event data specific to the event. This includes the name of the queue manager and, where appropriate, the name of the queue.

Where to find more information

More information on how to use events and details on the various types of event message, can be found in the *MQSeries Programmable System Management* manual.

Information about enabling events can be found in the *MQSeries Command Reference* manual.

Chapter 7. Introduction to MQSeries clients and servers

This chapter introduces the concept of MQSeries clients and servers and describes how they can be of benefit in your MQSeries installation. It gives you information about some of the items that you need to consider when you are planning your applications.

The chapter has the following sections:

What are MQSeries clients and servers?

An *MQSeries client* is a part of an MQSeries product that can be installed on a machine without installing the full queue manager. It accepts Message Queue Interface (MQI) calls from application programs, and passes MQI requests to an *MQSeries server* that is executing on another processor.

The *MQSeries server* is a full queue manager, which can accept MQI calls directly from application programs that are running on the server processor; in addition, it can accept MQI requests from MQSeries clients.

This allows you to have an application that uses the MQI running on one machine, the client machine, and the queue manager itself running on a different machine.

Clients and servers can be useful in a number of situations:

- Where there is no full MQSeries implementation for the machine (for example, because it is a DOS platform)
- Where the client machine is too small, or of insufficient processing power, to run a full queue manager with good performance
- Where you want to allow the application program on the client processor to connect to multiple queue managers on different server processors
- Where you might want to reduce systems administration effort

You can run an MQSeries application in both a full MQSeries environment and in a MQSeries client environment without changing your code. However, the libraries you use at link-edit time determine the environment your application must run in.

When an application program in the client issues an MQI call, the client formats the parameter values of the call into an MQI request, and sends the request to the server. The server receives the request, performs the action specified in the request, and sends a response back to the client. The response is used by the client to generate information that is returned to the application program using the normal MQI return mechanism.

An additional function supported by MQSeries clients, is the ability of an application program to be connected to more than one queue manager at a time, with the queue managers being on different processors or on the same processor.

Communication between clients and servers

An MQSeries client communicates with an MQSeries server, using an *MQI channel*, which is used to transfer MQI call requests from the client to the server, and responses from the server back to the client.

MQI channels differ from *message channels* (that are used to connect queue managers) in two ways:

- **An MQI channel is bidirectional.** One MQI channel can be used to send requests in one direction, and responses in the opposite direction.

  With message channels, data can be passed in one direction only. If two-way communication is required between two queue managers (for example, in the situation where reply messages are to be sent to the same queue manager that handled an initial request message), then two message channels are required, one to handle messages travelling in one direction, and another for messages travelling in the other direction.

- **Communication on an MQI channel is synchronous.** When an MQI request is transmitted from a client to a server, the MQSeries client product must wait for a response from the server before it can send the next MQI request.

  With message channels, the message traffic on the channel is time-independent. Multiple messages can be sent from one queue manager to the other, without the sending queue manager having to wait for any replies from the receiving queue manager.

The transmission protocol that is to be used on an MQI channel is specified as part of the channel definition. The MQSeries application program is unaware of the particular protocol that is being used on the channel. Furthermore, in the situation where an application program is connected to more than one MQSeries server, the MQI channels that are used for these connections could use different protocols. For example, an application program could connect to one queue manager using TCP/IP on one channel, and to another queue manager using NetBIOS on a different channel.

With both MQI and message channels, a channel definition is required at each end of the channel, and each of these definitions must include a *channel type* and a *channel name*. You can choose to use different channel types according to the application you are designing, but the same channel name must be used at both ends of the channel.

Installing clients and servers

For those products that support both clients and servers, the product, as delivered, contains the files for the MQSeries server, together with the client files for the server platform and several other platforms.

For example, if you order MQSeries for AIX V5, in addition to the base product you will receive files for the AIX, DOS, HP-UX, OS/2, Sun Solaris, and Windows clients.

The client software can be loaded to memory in the client processor either from the disk on the client or server machine, or from a disk on a LAN file server.

You can install the client and the server either by installing from the media on which the products are supplied, to the client or server disk, or by installing to a LAN file server, and loading the client or server from there.

The MQSeries for AS/400, MQSeries for MVS/ESA, and MQSeries for Tandem NSK products cannot be used as MQSeries clients and do not include any client files. However, they do include the MQSeries server code needed to support any clients that you might choose to install, either from another MQSeries product or from the Internet.

MQSeries clients from IBM Transaction Processing SupportPacs

You can install the MQSeries client files from an IBM Transaction Processing SupportPac.

The IBM Transaction Processing SupportPacs library consists of material that complements the family of CICS and MQSeries products marketed by IBM. The Transaction Processing SupportPacs library is available on the Internet at:

```
http://www.software.ibm.com/ts/mqseries/txppacs/txpsumm.html
```

MQSeries client software is available at no charge but is subject to the IPLA and License Information terms defined when requesting the MQSeries clients on the Internet. You have the right to make as many copies of the MQSeries client as necessary.

The VM/ESA client is shipped with the VM/ESA product; it is not available as a SupportPac.

National language considerations for clients

The client part of an MQSeries product includes a file that contains all the program and operator messages that are used by the product. This file has been translated to the national languages of the server product, so that no further translation of these messages is required.

Data conversion considerations for clients

The data conversion facilities provided for MQSeries application programs are the same as those that are available for application programs that are executing with MQSeries servers:

- For certain built-in formats, conversion can be performed during the processing of an MQGET call, if the data conversion option is included in the call.

- For application defined formats, the conversion can be performed by a user-exit program, called during the processing of an MQGET call.

For further information about data conversion with MQSeries clients and servers, refer to the *MQSeries Application Programming Guide* and the *MQSeries Clients* book.

Product support for MQSeries clients

The platform support for MQSeries clients and servers is as follows. Any of the MQSeries products listed is installed as a *Base product and Server* (*Base product and Distributed Queuing without CICS feature, and Client Attachment feature* on MQSeries for MVS/ESA). These MQSeries products can accept connections from the MQSeries clients on the platforms listed, subject to differences in coded character set identifier (CCSID) and communications protocol.

Note: If you are using previous versions of MQSeries products, make sure that code conversion from the CCSID of your client is supported by the server. See the Language support tables in the *MQSeries Application Programming Reference*.

The following MQSeries products:

- MQSeries for AIX Version 5
- MQSeries for AS/400 Version 4 Release 2
- MQSeries for AT&T GIS UNIX Version 2.2
- MQSeries for Digital OpenVMS Version 2.2
- MQSeries for HP-UX Version 5
- MQSeries for MVS/ESA Version 1 Release 2
- MQSeries for OS/2 Warp Version 5
- MQSeries for SINIX and DC/OSx Version 2.2
- MQSeries for SunOS Version 2.2
- MQSeries for Sun Solaris Version 5
- MQSeries for Tandem NSK Version 2.2
- MQSeries for Windows NT Version 5

can accept connections from MQSeries clients on:

- AIX
- AT&T GIS UNIX (this platform has become NCR UNIX)
- Digital OpenVMS
- DOS
- HP-UX
- OS/2
- SINIX and DC/OSx
- SunOS
- Sun Solaris
- VM/ESA
- Windows 3.1
- Windows 95
- Windows NT

MQSeries clients on other platforms

The MQSeries clients included in this book are the ones supplied with the MQSeries products listed above (“Product support for MQSeries clients” on page 50). Each MQSeries product (except MQSeries for MVS/ESA, MQSeries for AS/400, and MQSeries for Tandem NSK) supplies files for clients on the same platform as the server and for a number of other platforms.

Further MQSeries clients that can connect to the MQSeries products listed above are available through the Internet. For details of the platform and support status, see:

```
http://www.software.ibm.com/ts/mqseries/txppacs/txpsumm.html
```

Where to find more information

More information about MQSeries clients can be found in the *MQSeries Clients* book.

Chapter 8. Introduction to the MQSeries Framework

This chapter describes the MQSeries Framework. It contains the following sections:

Why the MQSeries Framework?

The MQSeries Framework offers users and independent software vendors the opportunity to extend or replace queue manager functionality, using defined interfaces.

These interfaces are provided in various forms. Some, for example the MQSeries name service interface, are provided by user-supplied modules that interface to the queue manager through an application programming interface. The trigger monitor interface, on the other hand, is provided by means of trigger messages that are written to a special queue.

In some cases components are shipped ready for you to use. You can choose whether to make use of them, and you can also decide to use your own versions instead of, or as well as, the supplied versions.

Not all MQSeries products provide all of the interfaces defined by the MQSeries framework. When an interface has been provided by an MQSeries product on a particular platform, it will be retained for future releases on that platform.

The major reasons for allowing modifications of the functions are:

- To provide the flexibility of choosing whether to use components provided by MQSeries products, or to replace or augment them with others
- To allow independent software vendors to participate, by providing components that might be using new technologies, without requiring internal changes to MQSeries products
- To allow MQSeries to exploit new technologies faster, and so provide products sooner

The components of the MQSeries Framework are:

- Trigger monitor interface (TMI)
- Message channel interface (MCI)
- Name service interface (NSI)
- Security enabling interface (SEI)
- Data conversion interface (DCI)

These are shown in Figure 8, and are outlined in the following text. A detailed definition of each interface can be found in the book or books indicated.

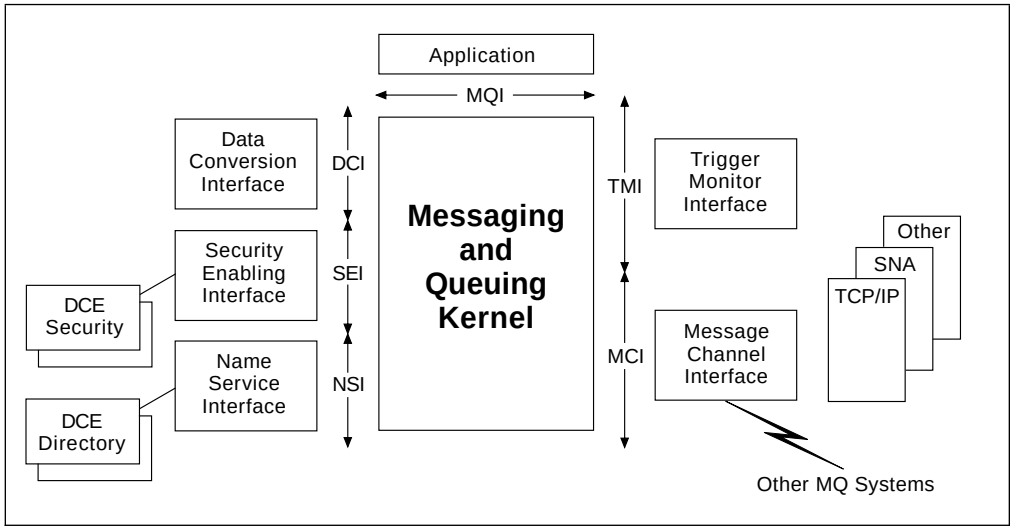

Figure 8. The MQSeries Framework

Invoking MQSeries Framework components

Where components are supported, they can be installed as necessary to provide some, or all, of the available function.

The queue manager refers to an initialization file to determine the particular components installed, and to get the information that it requires to invoke these components.

Services and components provided

This section gives information on the MQSeries Framework components and on the particular components that are provided by the MQSeries products.

Trigger monitor interface (TMI)

When messages arrive on a queue, the queue manager can generate trigger messages on a special kind of queue called an initiation queue. The generation of these messages is controlled by setting attributes of the queue.

The structure of the data in a trigger message is defined by the TMI. Trigger messages can be read by a long-running transaction called a trigger monitor, which starts applications to process the messages that have arrived on the queue.

MQSeries provides trigger monitors for use in various environments. When they start applications, they pass a defined structure as a parameter to the application. You can also provide your own trigger monitors, making use of the defined format of the trigger message. Your trigger monitors can pass the same structure as the supplied ones, or you can choose to do something different.

See the information on the trigger monitor interface (TMI) in the *MQSeries Application Programming Guide* for more information.

Message channel interface (MCI)

If a message is destined to go to a remote queue manager, the queue manager places it on a special kind of queue called a transmission queue. The message data has a header that includes the names of the destination queue manager and queue.

Triggering (see “Trigger monitor interface (TMI)” on page 54) can be used to start a process that reads the messages from a transmission queue, and sends them to their destination. Alternatively, this process can be started by an administrator.

MQSeries provides message channel agent (MCA) programs that transmit messages to MCAs on other queue managers. Commands are also provided to carry out channel administration.

You can provide alternative processes to take messages from transmission queues, and:

- Transmit the messages to similar processes you have provided on other queue managers, or
- Insert the messages into some other messaging system.

For details of how to implement the MCI, refer to the information on the transmission queue header structure (MQXQH) in the *MQSeries Application Programming Reference* manual to see how to write a message channel agent (MCA) program.

You can take advantage of triggering to initiate your process. You might need to provide administrative functions, similar to those available for MQSeries channels, for defining and monitoring the activity of your processes.

Note that the MCAs you write cannot communicate with MQSeries-supplied MCAs.

Name service interface (NSI)

Queue managers provide functions that allow administrators to define and manipulate queue definitions. Each queue manager maintains a directory of the queues that have been defined to it.

When an application opens a queue, the local queue manager looks up the queue in its directory if the application does not provide the name of a remote queue manager to which the queue belongs. If the queue is not found in that directory, the open normally fails.

However, you can install a naming service, which will be invoked if the queue manager is unable to resolve a queue name. Your service is passed the name of the queue that cannot be resolved. If your service recognizes the name, it returns the name of the queue manager to which the queue belongs.

The service provides the following functions:

- Lookup queue - given a queue name, finds the directory entry that contains that name, and returns the queue manager name
- Insert queue - inserts a new entry into the directory
- Delete queue - deletes an entry from the directory

The interface to the naming service is a programming interface, called the NSI. It is described in the *MQSeries Programmable System Management* book.

The NSI is available on MQSeries for OS/2 Warp, MQSeries for Windows NT, and MQSeries on UNIX systems.

DCE naming component

The *DCE naming* installable component provides naming services for queues within one DCE cell.

Details of this component can be found in the *MQSeries Programmable System Management* book.

The component is provided with MQSeries for AIX, Digital OpenVMS, HP-UX, OS/2 Warp, Sun Solaris, and Windows NT, but is not invoked unless it is included in the queue manager initialization file. The component requires that DCE support is available.

Security enabling interface (SEI)

There are three parts to the SEI:

- Authorization service
- User identifier service
- Message channel agent exits

See also Chapter 3, “Introduction to MQSeries security” on page 21 for further information.

Authorization service

This service provides access control facilities to the queue manager when an application issues an MQI call (for example, MQOPEN) or a command (for example, DELETE QLOCAL) that requires an authorization check to be carried out.

This enables the queue manager to check that users, or programs, have the appropriate authority for the actions they are trying to perform on queue manager objects.

The authorization service provides the following functions:

- Check object authority
- Set object authority
- Set initial authority
- Delete object authority
- Get object authority
- Copy all object authority

You can install your own version of this service, which can either be self-contained, or might in turn interface to some other authorization service that is available on the platform. Your service is invoked through a programming interface, which is part of the SEI. It maintains (either itself or by using another underlying service) lists of authorizations, and upon request returns information to the queue manager about whether a particular principal has authority to perform a certain action on a specified object.

This service is provided on MQSeries for UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT.

Object authority manager: The *object authority manager (OAM)* is an installable component that provides authorization services.

The component is provided as part of the MQSeries on UNIX systems products.

MQSeries for OS/2 Warp does not provide an authorization component, but is designed to accept any components (that provide authorization service functions) you might want to supply or obtain from independent software vendors.

User identifier service

On OS/2 Warp, it is not necessary for a user to log into the system, and therefore the queue manager cannot normally find a specific user identifier to associate with an application running on OS/2 Warp.

The user identifier service defines a programming interface, part of the SEI, which allows you to install your own service to supply a user identifier to the queue manager.

This service is provided only on MQSeries for OS/2 Warp.

This service is used by the queue manager to obtain a user ID. By default, the queue manager places the user ID in the message descriptor of messages when it puts them on queues, so that the application program that gets the messages can verify that they originated from authorized users or programs.

The queue manager can also make use of this user identifier for authorization checking.

The user identifier service provides just one function:

- Find user ID - obtain the predefined user ID

The user identifier service is not provided for client applications, for which another technique is available. See the *MQSeries Clients* manual.

Environment user ID: The *environment user ID* installable component provides the user identifier service, by obtaining a user ID value from an environment variable. The value of this variable is set when the component is installed.

Message channel agent exits

A mechanism for transmitting messages from one queue manager to another is referred to as a message channel. At each end of a channel a program called a message channel agent (MCA) controls the transmission of messages along that channel.

Exit points are defined at various points in the operation of an MCA. You can define functions that will be invoked at these points. Exit points are of particular relevance to security, and are therefore part of the SEI. They are:

- Security exits

  At each end, a security exit is invoked after communication has been established with the partner, but before message transfer starts. Each security

exit has the opportunity to exchange security messages with the security exit at the partner, in order to satisfy itself of the partner's authenticity. If either is not satisfied, it can prevent any message transfer.

- Message exits

  A message exit is invoked at the sending end just before a message is transferred, and at the receiving end just before it is stored. It has the opportunity to change any information in the message, including information in the message header such as the user identifier. This facility can be used to translate user identifiers on entering a new security domain or, for example, to set a blanket user identifier that has low authorization when receiving messages along a channel from an untrusted node. Message exits can also reject or reroute messages.

  On AIX, HP-UX, OS/2 Warp, OS/400, Sun Solaris, and Windows NT, you can call more than one message exit.

- Send and receive exits

  These are invoked just before any transmission is sent, and just after one is received. You can make use of this to provide encryption of sensitive data sent across open networks.

  On AIX, HP-UX, OS/2 Warp, OS/400, Sun Solaris, and Windows NT, you can call more than one send or receive exit.

You can define different exit modules for different channels.

See the *MQSeries Intercommunication* manual for further information about channel exits.

Data conversion interface (DCI)

Messages sent between platforms that normally use different encodings (for example, where integer fields are byte-swapped) and coded character set identifiers (CCSIDs) require conversion. An application getting a message from a queue can request that the queue manager converts the message data into the encoding and CCSID of its choice, which will normally be the standard ones in use on the platform on which it is running.

The message header contains information about the encoding and CCSID of the message data. It also contains a format name, which identifies the shape of the data. You can install exits, which have the same names as format names, to perform data conversions on the corresponding messages. These exits interface to the queue manager through a programming interface, called the DCI.

To help you provide these exits, MQSeries provides a utility that accepts a C language structure definition and generates C source code that can be built to provide an exit.

Built-in conversion is provided for MQSeries-defined standard formats.

The DCI is implemented only on MQSeries for MVS/ESA, MQSeries for OS/2 Warp, MQSeries for Windows NT, MQSeries on UNIX systems, and MQSeries for AS/400. When transmitting messages from a platform that does support it to one that does not, you can ask for this function to be carried out on all messages sent

along a particular channel. This means that when the messages arrive at their destination, they are already in the standard encoding and CCSID for that platform.

See the *MQSeries Application Programming Guide* for information about data-conversion exit programs.

Part 2. Planning for MQSeries for AS/400

Chapter 9. Introduction to MQSeries for AS/400

MQSeries for AS/400 runs on an AS/400 capable of running OS/400 to provide messaging and queuing applications support.

This chapter includes basic information on:

For more information on the hardware and software environment needed by the MQSeries for AS/400 product, see "MQSeries for AS/400 V4R2" on page 218.

Planning for MQSeries

This chapter helps you to plan for the introduction of MQSeries for AS/400 into your enterprise and introduces the items you must consider when doing this planning.

This chapter does not provide detailed information to enable you to perform the installation of MQSeries for AS/400. That information is provided in the *MQSeries for AS/400 Administration Guide*.

There are several stages in planning for the use of MQSeries for AS/400 that you must go through. They are:

1. Preparing your applications for the use of MQSeries for AS/400
2. Planning to include MQSeries for AS/400 in a network
3. Preparing to install MQSeries for AS/400
4. Planning to set up MQSeries for AS/400

A prime requirement for a message delivery system is that it must be reliable. Many functions are built into MQSeries for AS/400 to ensure that:

- Messages are not lost despite system failures
- Messages are not delivered more than once
- Messages are not accessed or delivered to unauthorized persons or applications

MQSeries for AS/400 uses OS/400 journaling and other facilities to support these functions.

You must also plan for the operation and administration of MQSeries for AS/400 in your enterprise and consider the implementation of the MQSeries for AS/400 security facilities in addition to those of OS/400. Brief outlines of these planning operations are included in this chapter.

Preparing your applications

MQSeries for AS/400 brings the Message Queue Interface (MQI) to your applications. This interface allows you to modify existing applications and to write new applications. The MQI removes much of the need to understand the network and communication systems that you use. Thus you can expect to generate applications more speedily than before. However, you must plan to take advantage of the MQI by planning its use in your applications and by understanding the ways in which it assists you.

You can find more information on how to use the MQI in your applications by referring to the *MQSeries Application Programming Guide*.

Interfacing with CICS

With the MQSeries for AS/400 products you can create application programs for the CICS for AS/400 transaction environment. These applications can use the MQI to communicate with CICS or non-CICS programs in any of the environments supported by the MQSeries products.

Using C++

You can use the OS/2 environment to write C++ programs for AS/400, using Client Access/400 to provide links to an AS/400 host, and VisualAge C++ for AS/400, which is a cross-compiler add-on for VisualAge C++ for OS/2. (Client Access/400 is an OS/2 program; it is not related to the MQSeries concept of clients.)

An OS/2 programmer can use Client Access/400 to mount an OS/400 file system. MQSeries for AS/400 includes an image for such a file system (IFS), containing a number of items specifically for OS/2 cross-compiler usage.

An OS/2 installation program is provided with MQSeries for AS/400 V4R2 on the IFS. This installs an OS/2 toolkit for users of VisualAge C++ for AS/400, including C and C++ header files, C++ sample program source, and C++ documentation. See the *MQSeries Using C++* manual for more information.

Planning to use MQSeries in a network

MQSeries for AS/400 uses the distributed queuing facility to exchange messages between MQSeries platforms using the SNA LU 6.2 and TCP/IP transmission protocols.

You must consider how you will attach MQSeries for AS/400 to a network, and how you will define the channels that are used to exchange messages.

According to the way that you have set your systems up, security checks can be performed at various times. MQSeries for AS/400 does not provide communications link authorization or data encryption on these links. Instead, various exits are provided that can be used by your applications to provide these facilities.

“MQSeries interoperability summary” on page 210 shows some of the links that are possible to other MQSeries products, and the transmission protocols that are used on these links. For further information about distributed queuing, refer to the *MQSeries Intercommunication* manual.

Planning recovery services

MQSeries for AS/400 makes use of the OS/400 journaling service to allow backup and recovery of the messaging system. Chapter 10, “Backup and recovery planning for MQSeries for AS/400” on page 67 introduces you to the recovery facilities and to the items you must consider in order to include MQSeries for AS/400 in your backup and recovery plans.

Planning data security

MQSeries for AS/400 makes use of the existing OS/400 security management facilities. This book indicates the particular way that MQSeries for AS/400 uses the standard OS/400 security.

Administration

A summary of the administration facilities provided by the MQSeries for AS/400 products is given in Chapter 12, “Administration of MQSeries for AS/400” on page 71. Full details of these facilities can be found in the *MQSeries for AS/400 Administration Guide*.

Concurrent use-based pricing

MQSeries for AS/400 counts concurrent users. A user is defined as a concurrent MQSeries connect. An MQSeries connect is established within an OS/400 job by an application program or command (including the Systems Administration Application and IBM-supplied commands) instigating an MQCONN call. It is terminated when an MQDISC call is instigated. (These calls can be issued explicitly or alternatively they can be implied by the use of the MQOPEN and MQCLOSE calls.) MQSeries does not charge for connections performed by the product itself, for example, the Command Server.

Each OS/400 job simultaneously connected to the active queue manager (server) is counted as one concurrent user. Only one connect can be established per job at any given time.

The number of users licensed for concurrent use is normally determined by the maximum who will be active at any one time. OS/400 provides monitoring and reporting for use-based programs. Customers will be advised when they approach the authorized user limit (90%). One use is included in the base MQSeries charge.

You can use the license information command (WRKLICINF) to manage the use of the MQSeries product.

Clients are not charged for per se, but their use implies a concurrent use at the server.

Installing and setting up

MQSeries for AS/400 is installed by using the OS/400 GO LICPGM command, issued from the command line. When you issue this command, you are presented with a menu from which you can choose to install MQSeries for AS/400.

To prepare for the actual installation, you need to plan how much DASD you require in your OS/400 system to accommodate MQSeries for AS/400.

Assistance is given in Chapter 13, “Storage planning for MQSeries for AS/400” on page 75 to help you plan the amount of DASD required by MQSeries for AS/400.

Installation

For MQSeries for AS/400, you must follow the instructions in the *MQSeries for AS/400 Administration Guide*.

The installation verification procedure is performed by creating and starting a queue manager, and creating the undelivered-message (dead-letter) queue.

Setting up

MQSeries for AS/400 requires some setting up after installation in order to meet the individual and special requirements of your system, and to use your system resources in the most effective way. Below are the items you must consider:

- Define queues
  - Consider your naming conventions for queues
- Define trigger processes
- Define remote links
  - Define associated transmission queues
  - Consider your naming conventions for remote queues
  - Consider your naming conventions for channels

The setup procedures are described in the *MQSeries for AS/400 Administration Guide*.

Chapter 10. Backup and recovery planning for MQSeries for AS/400

MQSeries for AS/400 uses the journaling and recovery facilities provided by OS/400. You should be familiar with these facilities and refer to the *AS/400 Backup and Recovery* manual.

MQSeries for AS/400 has two unique journals that are created when MQSeries is created. MQSeries data events are recorded in these journals under the normal AS/400 controls. The events that are recorded are those that are required to recover MQSeries in case of failure. For message queues, all persistent messages are recorded in the journals; nonpersistent messages are not. This means that, in the event of a problem causing loss of messages that exist on a queue, the persistent contents can be recovered by use of the journals. The methods for recovery of such data or messages are discussed in the *MQSeries for AS/400 Administration Guide*.

You need to consider how the message data routed through MQSeries is to be journaled, and how you will extend your existing recovery plans to include MQSeries related data.

Journal control

When data in a journal receiver reaches a threshold value, a new one is automatically created, attached, and brought into use by MQSeries. The old journal receiver is not deleted by MQSeries; this is a task that must be completed by the system administrator. MQSeries will issue a message to the administrator containing some key dates that are associated with the journal receiver that has been replaced by the new one. It is up to your system administrator to decide what action must be taken to dispose of the old journal receiver using the dates supplied in the message. The action might be to move the old journal receiver to long term storage, or simply to delete it. However, deletion of journal receivers must be carefully considered as the records might still be needed for restart. The receivers might contain information regarding long-lived persistent messages. The choice is dependent on your strategy for data security and recovery.

Figure 9 on page 68 illustrates the concepts of journaling with MQSeries for AS/400 and shows how the message sent to the administrator contains date information pointers to indicate the journal receivers that should be saved to off-line media, those saved journal receivers that can be deleted, and those online journal receivers that can be freed. System backups have not been considered in the above description of journaling for MQSeries. You should continue to take system backups as required by your standard system operating procedures.

For more information on creating a system backup plan, refer to the *AS/400 Backup and Recovery* manual.

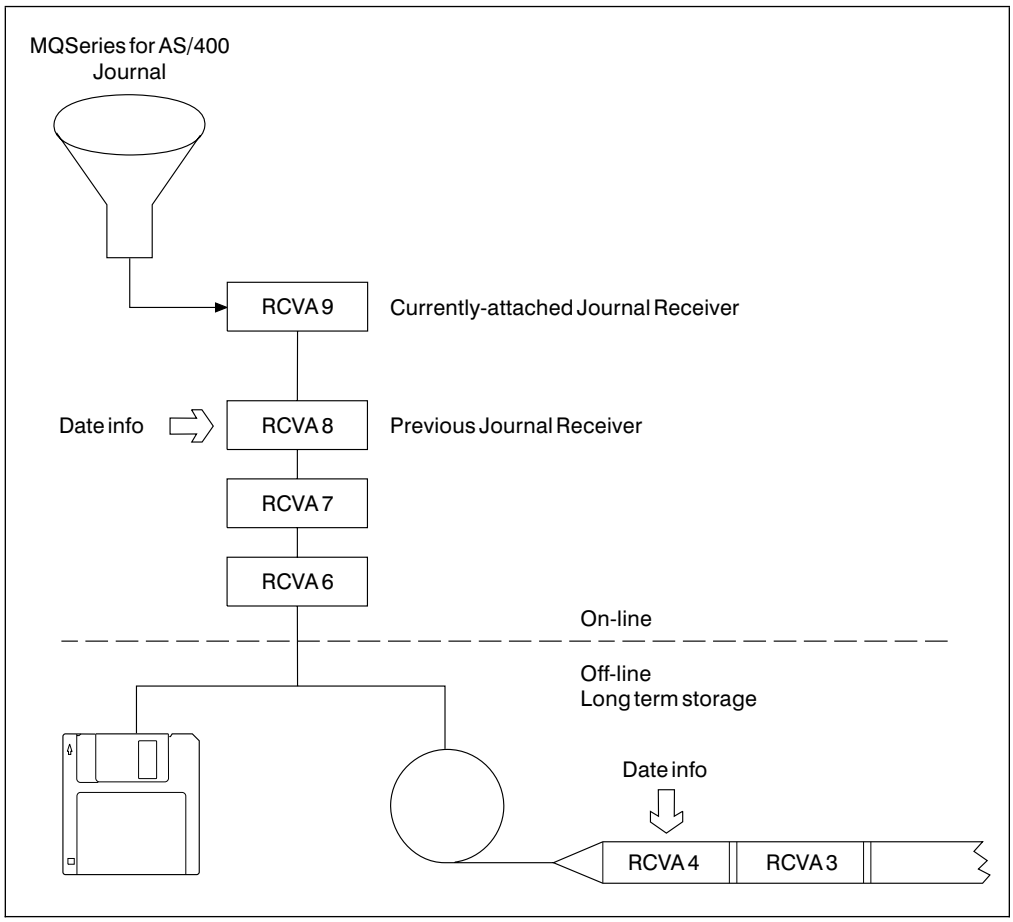

Figure 9. Journaling and date messages

Chapter 11. Security planning for MQSeries for AS/400

MQSeries for AS/400 uses OS/400 system security when accessing system objects. However, as an MQSeries product, the security accesses and controls are slightly different from other OS/400 operations.

You should be familiar with the security information contained in the *AS/400 Security - Reference* manual.

This chapter includes basic information about:

- "Naming differences between OS/400 and MQSeries for AS/400"
- "Security planning"
- "Security exits" on page 70

Naming differences between OS/400 and MQSeries for AS/400

MQSeries for AS/400 objects have 48-character names; OS/400 objects have 10-character names. MQSeries for AS/400 creates, owns, and maintains, a mapping between the 48-character MQSeries for AS/400 names and the 10-character names of the OS/400 objects that represent the MQSeries for AS/400 objects. If the MQSeries for AS/400 name is a valid OS/400 name, the same name is used in the mapping.

Security planning

The MQSeries for AS/400 objects that you must plan security authorities for are the queue manager object itself, the catalog object, and the administration object. You must also consider the security of your queues and processes.

The queue manager object contains the queue manager attributes, the catalog object contains the mapping information, and the administration object is used to hold the authorities to pass on or set message context.

MQSeries for AS/400 provides commands to grant, revoke, and query the object authority for MQSeries objects. This is because the MQSeries 48-character name is used, not the name of an OS/400 object. The authorities to MQSeries objects are those of the OS/400 application accessing the objects.

The mapping from MQSeries object authority to the system access authority occurs when an application is checked for the proper authority. For example, the authority on a queue is checked when an application issues an MQOPEN command through the MQI; it is not checked on each GET or PUT.

You need to plan how your users will be granted authority to the queue manager objects. You also need to plan authorizations to the programs and commands supplied with MQSeries. Authorities can be granted either individually, as a group, or by authorization lists. Granting authority follows normal OS/400 practice and you should refer to the *AS/400 Security - Reference* manual.

Note: The authorizations set by the MQSeries commands can be displayed using the DSPMQMAUT MQSeries command. You can also use the normal OS/400 commands, although it will not be obvious which MQSeries objects they apply to.

The authorizations can be copied from one user to another using the GRTMQMAUT MQSeries command, or by using the OS/400 grant object authority command.

For further details of MQSeries security handling, see the *MQSeries for AS/400 Administration Guide*.

Security exits

The message channels that are used for distributed queuing have exit facilities that can be used to invoke programs supplied by you. Examples of the types of function for which these exit facilities are intended include:

- Verification that the two partners at the ends of the channel are genuine, and have the appropriate security authorizations to take part in the exchange
- Encryption and decryption of messages

You need to consider whether you should provide exit programs to support your security plan.

For more information about security exits, refer to the *MQSeries Intercommunication* manual.

Chapter 12. Administration of MQSeries for AS/400

This chapter is a summary of the administration facilities provided by the MQSeries for AS/400 products. The chapter has the following sections:

Details of the commands, command interfaces, and utilities that are provided by MQSeries for AS/400 are given in the *MQSeries for AS/400 Administration Guide*.

Managing objects

It is the administrator's job to monitor MQSeries for AS/400 and make any changes that might be necessary. To do this, the administrator needs to know where each MQSeries object resides, what its characteristics are, and who has access to it.

The administrator can manage and monitor the resources either by use of command line entries, or by using the administration utility supplied with the product. Alternatively, if you have sets of commands that you issue regularly, you can write a CL program. Details of how to write these administration programs are contained in the *MQSeries for AS/400 Administration Guide*.

MQSeries for AS/400 can access security checks to ensure that the user is authorized to issue particular commands for particular resources.

Commands

MQSeries for AS/400 supports the following administration facilities:

- **CL Commands**

  MQSeries for AS/400 provides Control Language commands (CL commands). These can be issued either at the command line, or by writing a CL program.

  Examples of CL tasks include:

  – Authorize systems administrators
  – Start and stop a queue manager
  – Define, change, and display a queue
  – Define, change, and display a process
  – Define, test, and delete a channel

  CL commands are designed exclusively for OS/400, and CL responses are designed to be human-readable. The commands perform similar functions to PCF commands, but the format is designed to match the standard OS/400 format, whereas PCF commands are platform independent, with both the command and response formats being intended for program use.

- **MQSeries commands (MQSC)**

  With MQSeries for AS/400, you can build commands by entering the MQSC commands into a member of a specified library and running the STRMQMMQSC command. For information about how to do this, see the *MQSeries Command Reference* manual.

- **PCF commands**

  Any local or remote application program can generate PCF commands in messages and put them to the command queue, SYSTEM.ADMIN.COMMAND.QUEUE, to be processed by the MQSeries for AS/400 command server.

More information on how to use all these facilities is given in the *MQSeries for AS/400 Administration Guide*.

MQSeries for AS/400 can verify that the user is authorized to issue particular commands for particular resources.

Changes made to the resource definitions of a queue manager using the commands (directly or indirectly) are preserved across restarts of the MQSeries for AS/400 system.

Managing communications

Part of the administrator's role is to ensure that the required communications links are activated, and to monitor the status of these links as required by your enterprise. You can find information describing these tasks in the *MQSeries Intercommunication* manual.

Remote administration

There are two aspects to the MQSeries remote administration facilities:

- MQSeries for AS/400 can be used to manage remote systems
- Other remote products can be used to manage MQSeries for AS/400

Managing remote systems

Facilities are provided by MQSeries for AS/400 to allow an administrator to manage the following systems remotely:

- MQSeries for AS/400
- MQSeries for Digital OpenVMS
- MQSeries for OS/2 Warp
- MQSeries for Tandem NSK
- MQSeries on UNIX systems
- MQSeries for Windows NT

Because of the differences in the MQSeries products, it is not always possible to manage remotely the same set of MQSeries objects or attributes that you can manage locally.

If you wish to manage any other MQSeries product (for example, MQSeries for MVS/ESA) you can write an application program to send the appropriate commands to the command queue at the remote queue manager. However, some products do not have a command queue, so they cannot be administered from a local or remote application program.

Managing MQSeries from remote systems

MQSeries for AS/400 can be managed from a remote MQSeries system, by an administrator using the facilities provided by the following products:

- MQSeries for AS/400
- MQSeries for Digital OpenVMS
- MQSeries for OS/2 Warp
- MQSeries for Tandem NSK
- MQSeries on UNIX systems
- MQSeries for Windows NT

Note: You can manage MQSeries for AS/400 from MQSeries for MVS/ESA by writing an application program to send the appropriate PCF commands to the command queue.

Using the administration utility

Much of the functionality of the CL commands is available in a user-friendly way from the MQSeries for AS/400 administration utility, described in the *MQSeries for AS/400 Administration Guide*.

The utility can be used for the administration of local MQSeries for AS/400 products, and for remote administration of a number of other MQSeries products, as listed in “Remote administration” on page 72.

The MQSeries administration utility, when started, supplies a set of panels to help you perform various administrative tasks. These panels help you to:

- Start and stop the local queue manager
- Create, copy, and delete local and remote queue manager objects
- Display and change the attributes of a queue manager (local or remote), and its objects
- Grant and revoke security access to local queue manager objects
- View the contents of a queue and individual messages on a local queue manager

You can find information about the administration utility in the *MQSeries for AS/400 Administration Guide*.

Chapter 13. Storage planning for MQSeries for AS/400

This chapter tells you how to plan the type and amount of storage you require when you include MQSeries for AS/400 in your system. It contains information about:

Product storage

MQSeries for AS/400 requires approximately 60 MB of main storage in the QMQM library.

This size includes the product programs, message files, panels and prompts, commands, and ancillaries. The size varies slightly according to any options that you select.

See the appropriate *MQSeries for AS/400 Administration Guide* for details on the options that you can select.

Journal storage

Significant events and data changes are journaled by OS/400. Such data is sent by MQSeries for AS/400 to unique journals. These journals direct the data to a series of predefined journal receivers. The threshold values of these receivers are set up by MQSeries for AS/400 to a default size of 16 MB.

When data in a journal receiver reaches its threshold value, a new one is automatically created, attached, and brought into use by MQSeries for AS/400. A message is sent to the system administrator containing key dates associated with the journal receivers. Action must be taken by the administrator to dispose of the old journal receiver according to the dates supplied in the message. See Chapter 10, "Backup and recovery planning for MQSeries for AS/400" on page 67 for more information about journaling.

You need to plan for some permanent storage to which journal receivers can be offloaded when they are no longer required to be maintained in main storage. The journals can be offloaded to diskettes, DASD, tape, or other media supported in your enterprise.

Storage for other data sets

The total amount of storage you should reserve for MQSeries for AS/400 depends mainly on the amount needed to store messages; in the majority of cases, the amount needed to store the other MQSeries objects that are required, such as process objects, and the queue manager object, is small compared to that required for messages.

Therefore, you can get an approximate figure for the amount of storage required in the data sets, by calculating the amount required for both persistent and nonpersistent messages, using the algorithms described in the next section, and adding the two amounts together.

Message queues

All messages are stored in the QMQMDATA library, with nonpersistent and persistent messages being stored within different storage spaces. The amount of storage you should reserve is dependent on the number of messages you expect your applications to be handling at any one time, and the expected size of these messages.

Each message consist of two portions: a message descriptor, and message data.

The actual size of a message descriptor is 456 bytes. If you are using distribution lists, or grouped or segmented messages, the size of the header will increase for the transmission queue. However, because data storage is obtained by the queue manager in 1024 byte (1 KB) blocks, it is necessary to allow for one block of storage for the descriptor when calculating the amount of storage for messages.

The amount of storage required for the data portion of a message is the size of your message data, rounded up to the next 1 KB block boundary.

So, the amount of storage, in 1 KB blocks, that you should reserve for persistent messages is:

```
Number of 1 KB blocks required = n * (1 + D)

  where  n = number of messages

    and  D = message data size / 1024
             (D is rounded up to the next higher
              integer)
```

The amount of storage for nonpersistent messages is computed using the same algorithm.

Performance information

Information about MQSeries performance is available on the Internet at:

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

Part 3. Planning for MQSeries on Digital OpenVMS

Chapter 14. Introduction to MQSeries on Digital OpenVMS

There are two MQSeries products for Digital OpenVMS; one for AXP and one for VAX. The products comprise two parts, the *server* and the *clients*. The server runs on a machine that is capable of running an MQSeries queue manager; the clients provided with the product are for Digital OpenVMS, DOS, OS/2 Warp, and Windows 3.1. (The Windows 3.1 client can operate under Windows 3.1, Windows 95 or within the WIN-OS/2 environment under OS/2 Warp.)

MQSeries provides the MQI programming interface for use by application programs that are running on the server or the client processor. More detail on MQSeries clients and servers is given in Chapter 7, “Introduction to MQSeries clients and servers” on page 47.

For information on the hardware and software environments, see “MQSeries for Digital OpenVMS” on page 220.

Planning for MQSeries

This chapter helps you to plan for the introduction of MQSeries for Digital OpenVMS into your enterprise, and introduces the items that you need to consider when doing this planning.

There are several stages in planning for the use of MQSeries for Digital OpenVMS that you must go through. They are:

1. Preparing your applications for the use of MQSeries for Digital OpenVMS
2. Planning to include MQSeries for Digital OpenVMS in a network
3. Preparing to install MQSeries for Digital OpenVMS
4. Planning to set up MQSeries for Digital OpenVMS

A prime requirement for a message delivery system is that it must be reliable. Many functions are built into MQSeries for Digital OpenVMS to ensure that:

- Messages are not lost despite system failures
- Messages are not delivered more than once
- Messages are not accessed by, or delivered to, unauthorized persons or applications

MQSeries for Digital OpenVMS uses logging and other facilities to support these functions.

You must also plan for the operation and administration of MQSeries for Digital OpenVMS in your enterprise, and consider the implementation of an appropriate set of security facilities. Brief outlines of these planning operations are included in this chapter.

Preparing your applications

MQSeries for Digital OpenVMS brings the Message Queue Interface (MQI) to your applications. This interface allows you to modify existing applications and to write new applications. The MQI removes much of the need to understand the network and communication systems that you use. Thus you can expect to generate applications more speedily than before. However, you must prepare to take advantage of the MQI by planning its use in your applications and by understanding the ways in which it assists you.

You can find more information on how to use the MQI in your applications by referring to the *MQSeries Application Programming Guide*.

Planning to use MQSeries in a network

MQSeries for Digital OpenVMS uses the distributed queue management (DQM) facility to exchange messages between MQSeries platforms, using either the DECnet, SNA LU 6.2, or TCP/IP transmission protocols.

You must consider how you will attach MQSeries for Digital OpenVMS to a network, and how you will define the message channels that will be used to exchange messages.

According to the way that you have set your systems up, security checks can be performed at various times. Various exits are provided that can be used by your applications to provide these facilities.

"MQSeries interoperability summary" on page 210 shows the links that are possible to other MQSeries products, and the transmission protocols that can be used.

For further information about distributed queue management, refer to the *MQSeries Intercommunication* manual.

Installation

MQSeries for Digital OpenVMS is installed with the OpenVMS VMSINSTALL utility. For information about installing MQSeries, see the *MQSeries for Digital OpenVMS System Management Guide*.

To prepare for the actual installation, you need to plan how much disk space will be required in your Digital OpenVMS system to accommodate MQSeries. Assistance is given in Chapter 18, "Storage planning for MQSeries for Digital OpenVMS" on page 93 to help you plan the amount of space required.

Setting up

After installation, MQSeries for Digital OpenVMS needs to be set up, and customized for your own use. This ensures that the appropriate Digital OpenVMS facilities are made available to MQSeries, and that your MQSeries system is correctly initialized and ready to work with your applications.

MQSeries for Digital OpenVMS uses configuration files to hold the product configuration information used for logging, communications protocols and installable components. After installing the product, you can edit these files to tailor the operation of the product to meet the requirements of your installation.

In addition, you need to do the following:

- Define queues
  - Consider your naming conventions for queues
- Define trigger processes
- Define remote links
  - Define associated transmission queues
  - Consider your naming conventions for remote queues
  - Consider your naming conventions for channels

Note: The characters within the names given to all MQSeries objects are case sensitive. Therefore, be very careful when defining the names of objects, to select the appropriate uppercase or lowercase characters.

You can find more information on the setting up and customizing processes for MQSeries for Digital OpenVMS in the *MQSeries for Digital OpenVMS System Management Guide*.

Planning recovery services

MQSeries for Digital OpenVMS provides logging services to allow backup and recovery of the messaging system. Chapter 15, “Backup and recovery planning for MQSeries for Digital OpenVMS” on page 83 introduces you to these facilities, and to the items that you need to consider in order to include MQSeries for Digital OpenVMS in your backup and recovery plans.

You can find more information on the backup and recovery facilities provided by MQSeries for Digital OpenVMS in the *MQSeries for Digital OpenVMS System Management Guide*.

Planning data security

MQSeries for Digital OpenVMS uses the facilities of the MQSeries *object authority manager (OAM)* installable component to control access to the various different types of queue manager resource (queues, process definitions, channels, and queue managers).

You can find more general information on authorization installable components in Chapter 8, “Introduction to the MQSeries Framework” on page 53.

Chapter 16, “Security planning for MQSeries for Digital OpenVMS” on page 87 introduces you to the security facilities provided by MQSeries for Digital OpenVMS and to some of the items you need to consider when planning for security.

You can find more information on the security facilities provided by MQSeries for Digital OpenVMS in the *MQSeries for Digital OpenVMS System Management Guide*.

Administration

A summary of the administration facilities provided is given in Chapter 17, “Administration of MQSeries for Digital OpenVMS” on page 91. Full details of these facilities can be found in the *MQSeries for Digital OpenVMS System Management Guide*.

See also “MQSeries product administration facilities” on page 36.

Migration from MQSeries Version 1

To use MQSeries application programs that were written for MQSeries V1 with MQSeries for Digital OpenVMS, you need to do the following:

1. Redefine all message queues.
2. Redefine all message channels.
3. Recompile the application programs, using the MQSeries Version 2 header files.

This might be a suitable time to consider whether you need to re-design any parts of your application, to take advantage of the additional function provided by MQSeries for Digital OpenVMS.

One difference between MQSeries for Digital OpenVMS V2.2 or later, and the earlier products is that MQSeries for Digital OpenVMS does its own queue storage management. It is not necessary to run a utility program to recover the space that was occupied by messages that have been removed from queues by MQGET calls; this is done automatically by MQSeries.

Chapter 15. Backup and recovery planning for MQSeries for Digital OpenVMS

This chapter describes the background concepts of recovery and restart. It contains the following sections:

More details of the logging and recovery facilities of MQSeries for Digital OpenVMS are given in the *MQSeries for Digital OpenVMS System Management Guide*.

Logging

The basic premise of a messaging system is that messages entered into the system are assured of delivery to the destination. One of the ways of ensuring that messages are not lost is to maintain a record of the activities of the queue manager that handles the receipt, transmission, and delivery of messages.

MQSeries for Digital OpenVMS does this by recording all the significant changes to the data controlled or managed by the queue manager in a log. This process is called logging. The data changes that are logged include the puts and gets of persistent messages to and from queues, changes to queue attributes, and channel activity.

The purpose of logging is to create and maintain a log that:

- Keeps records of queue manager changes
- Keeps records of queue updates for use by the restart process
- Is a source for restoration of data should there be a hardware or software failure

Each MQSeries log consists of a log control file, together with one or more log files for the storage of data.

The log control file is, as its name implies, used to control and monitor the use of the log files. It contains information relating to the size, location, next available file, and other data related particularly to the log files themselves. All the log files within one log are the same size; there is a default value for this size, but you can override this value when you set up the log.

Types of logging

MQSeries for Digital OpenVMS has two approaches to maintaining records of queue manager activities:

- Circular logging
- Linear logging

Each type of logging stores the recorded data in a set of files. The differences between the two types of logging are the contents, and the way that the files are linked together.

With circular logging, the set of log files are effectively linked together so as to form a ring. When data is collected, it is written sequentially into the files, in such a way as to reuse the log files in the ring. You can use circular logging for:

- Crash recovery - that is, after a system failure of some kind has stopped the queue manager unexpectedly
- Restart recovery - after a planned closedown of the system

With linear logging, the log is maintained as a continuous sequence of files. When data is collected, it is written sequentially into the log files; the space in the files is not reused, so that you can always retrieve any record from the time that the queue manager was created.

Because disk space is finite, you might have to plan for some form of archiving. Also, if you are handling a high volume of persistent messages, all your log files will eventually be filled. This will result in operator messages being written to an error log file; some action will need to be taken by the system administrator to make more log space available, or to reuse the existing space. You can use linear logging for:

- Crash recovery
- Restart recovery
- Media recovery - to recreate lost or damaged data after a media failure by replaying the contents of the log

Selecting a logging method

You must base your selection of log type on your requirements for recovery.

Both types of logging can cope with unexpected power outages in the absence of hardware failure. If you accept that only crash or restart recovery is required, circular logging might be adequate. If media recovery is important to you, select linear logging.

With each type of logging, you need to decide on the number of files to use in the log, and their size. The total amount of space needed depends on the amount of data to be recorded, which depends on various parameters, including:

- The size of messages
- The number of puts and gets from queues
- The number of messages being transmitted by the message channel agents

Recovering from problems

MQSeries can recover from communications failures and power loss incidents. In addition, it is sometimes possible to recover from other types of problem with the MQSeries data, such as inadvertent deletion of a file.

In the case of a communications failure, messages remain on the queues until they are removed by a receiving application. If the message is being transmitted, it remains on the transmission queue until it can be successfully transmitted. To recover from a communications failure, it is normally sufficient simply to restart the channels using the link that failed.

On a restart after your system has lost power, the queue manager restores all the persistent messages that were on the queues to the state that existed just before the power failure, so that no persistent messages are lost; nonpersistent messages are discarded.

There are ways in which an MQSeries object can become unusable, for example due to inadvertent damage. In such situations, you will have to take steps to recover either your complete system or some part of it. The action required depends on when the damage is detected, whether the log method selected supports media recovery, and which object or objects are damaged.

Chapter 16. Security planning for MQSeries for Digital OpenVMS

This chapter describes the access control security features in MQSeries for Digital OpenVMS. It contains these sections:

Full details of MQSeries for Digital OpenVMS security handling are given in the *MQSeries for Digital OpenVMS System Management Guide*.

Controlling access to resources

With MQSeries for Digital OpenVMS, access to queue manager resources is controlled through the *object authority manager (OAM)*, which is the default authorization installable component. Because the OAM is an installable component, you can implement your own security controls in place of, or in addition to, those supplied by the OAM. (For more information on installable services and installable components, see Chapter 8, "Introduction to the MQSeries Framework" on page 53.)

Users can access queue manager objects (queues, process definitions, channels, and queue managers) only if they have the required authority. The OAM manages a user's authorization to manipulate MQSeries objects, and provides a command interface through which you can grant or revoke access authority to an object for a specific group of users.

Managing access through user groups

In discussing security in a Digital OpenVMS environment, we use the term *principal* rather than user ID. The reason for this is that authorities granted to a user ID can also be granted to other entities, for example, an application program that issues MQI calls, or an administration program that issues PCF commands. In these cases, the principal associated with a program is not necessarily the user ID that was used when the program was started.

Managing access permissions to MQSeries resources is based on Digital OpenVMS *rights identifiers*, that is, identifiers held by principals. A principal can hold one or more OpenVMS rights identifier. A group is defined as the set of all principals that have been granted a specific rights identifier.

The OAM does not maintain authorizations at the level of individual principals. The mapping of principals to identifier names is carried out within the OAM, and operations are carried out at the rights identifier level.

The authorizations that a principal has are the union of the authorizations of all the rights identifiers that it holds, that is, its process rights. Whenever a principal requests access to a resource, the OAM computes this union, and then checks the authorization against it.

Resources you can protect

Through MQSeries for Digital OpenVMS 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 the identifier held) 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 principals that hold rights identifier **mqm** can execute queue manager administration commands, for example, to create a queue.
- Permission to use control commands. Only principals that hold rights identifier **mqm** can execute control commands, for example, creating a queue manager, starting a command server, or using runmqsc.
- Permission to use PCF commands.

Different users can be granted different kinds of access authority to the same object. For example, for a specific queue, users holding one identifier might be allowed to perform both put and get operations; users with another identifier might only be allowed to browse the queue (MQGET with browse option). Similarly, users with identifiers might have get and put authority to a queue, but might not be allowed to alter or delete the queue.

Using rights identifiers for authorizations

Using identifiers for authorization, rather than individual principals, 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 simply by granting the appropriate identifier to their OpenVMS user ID.

Try to keep the number of groups as small as possible. Dividing principals into one group for application users, and one for administrators, is a good place to start.

Using the security commands

The OAM provides a command interface for granting and revoking authority. Before you can use these commands, you must be suitably authorized - your user ID must hold the OpenVMS rights identifier **mqm**. This identifier should have been set up when you installed the product.

If your user ID holds identifier **mqm**, you have a ‘super user’ authority to the queue manager. This means that you are authorized to issue any MQI request or command from your user ID.

The OAM provides two commands that you can invoke from your OpenVMS DCL to manage the authorizations of users. These are:

- setmqaut (Set or reset authority)
- dspmqaut (Display authority)

Details of these commands can be found in the *MQSeries for Digital OpenVMS System Management Guide*.

Security exits

The message channels that are used for distributed queuing, and the MQI channels that are used between clients and servers, both have security exit facilities that can invoke programs that you have supplied.

For more information on these security exit programs, see the *MQSeries Intercommunication* manual.

Chapter 17. Administration of MQSeries for Digital OpenVMS

This chapter is a summary of the administration facilities provided by MQSeries for Digital OpenVMS. It has the following sections:

Details of the commands, command interfaces, and utilities that are provided by MQSeries for Digital OpenVMS are given in the *MQSeries for Digital OpenVMS System Management Guide*. You need to arrange for users who need to be able to use these administration facilities to have the necessary authorizations, using the procedures given in the *System Management Guide*.

Managing objects

It is the administrator's job to monitor MQSeries for Digital OpenVMS and make any changes that might be necessary. To do this, the administrator needs to know where each MQSeries object resides, what its characteristics are, and who has access to it.

The administrator can manage and monitor the resources using MQSeries commands (MQSC), or, if there are sets of commands that are issued regularly, by writing an application program that places them on the command queue.

MQSeries can use the security features provided by the OAM, or by a security component that you have installed, to ensure that the user is authorized to issue particular commands for particular resources.

Commands

MQSeries for Digital OpenVMS supports the following administration commands and facilities:

- You can enter control commands on the command line
- You can use the **runmqsc** control command to cause MQSC commands from standard input to be executed
- Any local or remote MQSeries application program can generate PCF commands in messages and put them to the command queue SYSTEM.ADMIN.COMMAND.QUEUE, to be processed by the MQSeries for Digital OpenVMS command server

More information on how to use all these facilities is given in the *MQSeries for Digital OpenVMS System Management Guide*.

Changes made to the resource definitions of a queue manager using the commands (directly or indirectly) are preserved across restarts of MQSeries for Digital OpenVMS.

Managing communications

Part of the administrator's role is to ensure that the required communications links are activated, and to monitor the status of these links as required by your enterprise. You can find information describing these tasks in the *MQSeries Intercommunication* manual.

Remote administration

There are two aspects to the MQSeries remote administration facilities:

- MQSeries for Digital OpenVMS can be used to manage remote systems
- Other remote products can be used to manage MQSeries for Digital OpenVMS

Managing remote systems from MQSeries for Digital OpenVMS

Facilities are provided by MQSeries for Digital OpenVMS to allow an administrator to manage the following remote systems:

- MQSeries for AS/400
- MQSeries for Digital OpenVMS
- MQSeries for MVS/ESA
- MQSeries for OS/2 Warp
- MQSeries for Tandem NSK
- MQSeries on UNIX systems
- MQSeries for Windows NT

Because of the differences in the MQSeries products, it is not always possible to manage remotely the same set of MQSeries objects or attributes that you can manage locally.

If you want to manage any other MQSeries product, you can write an application program to send the appropriate commands to the command queue for that product. However, some MQSeries products do not have a command queue, so they cannot accept commands from local or remote application programs.

Managing MQSeries from remote systems

MQSeries for Digital OpenVMS can be managed from a remote MQSeries system, by an administrator using the facilities provided by the following products:

- MQSeries for AS/400
- MQSeries for Digital OpenVMS
- MQSeries for MVS/ESA
- MQSeries for OS/2 Warp
- MQSeries for Tandem NSK
- MQSeries on UNIX systems
- MQSeries for Windows NT

Note: You can manage MQSeries for Digital OpenVMS from MQSeries for MVS/ESA by writing an application program to send the appropriate PCF commands to the command queue.

Chapter 18. Storage planning for MQSeries for Digital OpenVMS

This chapter tells you how to plan the type and amount of storage you require when you include MQSeries for Digital OpenVMS in your network. It has the following sections:

RAM considerations

The processor memory (RAM) is used by MQSeries for Digital OpenVMS in the execution of the product modules, and as a paging area for the messages that are being processed. Parts of the paging area are written to, and read from, disk as necessary.

The minimum amount of RAM required to run the MQSeries for Digital OpenVMS server is 16 MB on VAX and 32 MB on AXP, if more RAM is available, the performance of the message processing improves.

The amount of RAM required on each client system for an MQSeries client is small compared to that required for the operating system on each of the platforms.

Disk space considerations

Disk space is used by MQSeries for Digital OpenVMS for the following:

- Product modules - client and server executable modules and the toolkit
- Paging space - server only
- Message queues - server only
- Logs - server only

Product modules

The disk space that you require for the product modules depends on the options that you decide to install: the options are described in the *MQSeries for Digital OpenVMS System Management Guide*.

Space might be required for client and server executable program modules and the toolkit. If all options are selected, 16 MB are required on VAX and 18 MB on AXP.

Paging space

The messages that the server is processing are stored in memory, and are paged to disk.

The algorithm used to calculate the size of the paging space required to support a particular application is complex, and is not given in this manual. Instead, you should refer to "Sample configuration" on page 95, which gives the paging space used on the typical minimum configuration.

Message queues

In order to estimate the total amount of storage that you will need for queues, you need to know:

- The number of queues that you have.
- The maximum number of messages there will be on each of the queues at any one time.
- The average size of message on each of the queues. The amount of storage required for one message varies. It is based on the size of the message data plus the size of the message header (456 bytes), rounded up the nearest 512-byte block. If you are using distribution lists, or grouped or segmented messages, the size of the header will increase for the transmission queue.

Given these values, you can calculate the total amount of space required for queues. However, this value is likely to be an approximate value only, and it is advisable to add a contingency value, to avoid the situation where there is no space left for messages on the queues when your application is running.

Log files

Significant events and data changes can be logged in circular or sequential logs. In particular, the logs are used for recording persistent messages.

All the log files in a log are of the same size. By default, this size is 4 MB, but this value can be changed by the system administrator when the log is defined.

For a circular log, the system administrator needs to specify how many files should be included in the log. For a sequential log, the number of files will increase over time, until the system administrator archives some of the files, and disposes of them. You need to plan for the permanent storage (diskettes, tape, or other media supported in your enterprise) that is to be used for these archived files.

Sample configuration

This section gives the amounts of storage required for a sample configuration consisting of a single server on the Digital OpenVMS processor with:

- The disk on the server holding the server and client product modules and the toolkit
- System default objects
- 20 local queues, each with 50 messages of 1 KB total length; half of the messages are persistent messages
- A single client
- No message channels for distributed queuing

The storage requirements for the server in this configuration are:

- RAM: 16 MB for VAX, 32 MB for AXP
- Disk space for MQSeries for Digital OpenVMS:
  - Product modules: 16 MB for VAX, 18 MB for AXP
  - Paging space: 64 MB
  - Message queues: 1.5 MB
  - Logs: 12 MB (circular log)

The storage requirements for the clients in this configuration are small compared to the storage required for the operating system on the platforms.

Capacity planning figures

The following table shows the notional limits of the number of operations that can be performed on each MQSeries-connected workstation. It is recommended that, for adequate performance, you do not exceed these limits.

The numbers quoted were measured on an Alpha Server 1000 4/266 server that had 128 MB of installed memory.

Table 13. MQSeries for Digital OpenVMS, capacity planning

Resource	**Maximum number**
Queue managers (heavily used)	4
Connections (including channels)	112
Open queues	1024
Active channels	112
Clients	112

Note: The number of connections, active channels, and clients can be doubled if you use the MQSeries fastpath bindings.

Part 4. Planning for MQSeries for MVS/ESA

Chapter 19. Introduction to MQSeries for MVS/ESA

MQSeries for MVS/ESA runs on a System/370 or System/390 that is capable of running MVS/ESA, to provide messaging and queuing support for the following types of application:

- CICS for MVS/ESA and CICS/MVS
- IMS/ESA
- TSO and batch

This chapter provides basic information about:

- "Planning for MQSeries"
- "Installing and customizing" on page 102

Chapter 24, "Storage planning for MQSeries for MVS/ESA" on page 121 gives information on the storage requirements for MQSeries for MVS/ESA. For more information on the hardware and software requirements for MQSeries for MVS/ESA, see "MQSeries for MVS/ESA" on page 224.

Planning for MQSeries

This chapter helps you to plan for the introduction of MQSeries for MVS/ESA into your enterprise. Chapter 1, "Introduction to IBM MQSeries" on page 3 introduces you to the concepts you must consider.

This chapter does not provide you with detailed information to enable you to perform the installation of MQSeries for MVS/ESA. That information is provided in the *MQSeries for MVS/ESA Program Directory.* Equally, you are directed to other publications for detailed information on particular topics.

There are several stages in planning for the use of MQSeries for MVS/ESA that you must go through. They are:

1. Preparing your applications for the use of MQSeries for MVS/ESA
2. Planning to include MQSeries for MVS/ESA in a network
3. Preparing to install MQSeries for MVS/ESA
4. Planning to customize MQSeries for MVS/ESA

A prime requirement for a message delivery system is that it must be reliable. Many functions are built into MQSeries for MVS/ESA to ensure that:

- Messages are not lost despite system failures
- Messages are not delivered more than once
- Messages are not accessed by, or delivered to, unauthorized persons or applications

The features provided by MQSeries for MVS/ESA to support this, and which you must plan for, include:

- Logging (including options for dual logging and archiving)
- Automatic recovery from transaction, system, and storage media failures (for which suitable backups are needed)
- Restart from backup files
- Access to security management facilities

You must also plan for the administration of MQSeries for MVS/ESA in your operation, make decisions regarding the performance aspects of the product, and consider, if necessary, possible migration of both your system and applications from other products. Brief outlines of these planning operations are included in this chapter.

Preparing your applications

MQSeries for MVS/ESA brings the Message Queue Interface (MQI) to your applications. This interface allows you to modify existing applications and to write new applications. The MQI removes much of the need to understand any network or communication systems that you use. Thus you can expect to complete applications more speedily than before. However, you must plan to take advantage of the MQI by planning its use in your applications and by understanding the ways in which it assists you.

You can find more information on how to use MQSeries for MVS/ESA in your applications by referring to the *MQSeries Application Programming Guide*.

Interfacing with CICS, IMS, or Batch

With MQSeries for MVS/ESA you can create applications in these environments:

- A CICS transaction environment
- An IMS transaction environment
- An MVS Batch and Time Sharing Option (TSO) environment

Applications (or transactions) connect to MQSeries by means of an *adapter*. There are three adapters, one for each of these environments, included in MQSeries for MVS/ESA.

The MQSeries-IMS bridge

The MQSeries-IMS bridge is a component of MQSeries for MVS/ESA that allows direct access from MQSeries applications to applications on your IMS system. It enables implicit MQSeries API support. This means that you can re-engineer legacy applications that were controlled by 3270-connected terminals to be controlled by MQSeries messages, without having to rewrite, recompile, or relink them.

The bridge is an IMS *Open Transaction Manager Access* (OTMA) client.

Planning to use MQSeries in a network

MQSeries for MVS/ESA uses the distributed queue management (DQM) facility to exchange messages between MQSeries platforms.

With MQSeries for MVS/ESA, access to remote queues can be with:

- SNA LU 6.2
- TCP/IP
- Interlink SNS/TCPaccess
- CICS for MVS/ESA (using ISC)

Which of these alternative methods of connection is best for your installation depends on a number of factors, including:

- Whether you require CICS for MVS/ESA for use by your applications
- What other MQSeries products you wish to connect to, and the methods of connection that they support
- If there are alternatives possible, whether you require the best possible performance

You need to decide which of these protocols you are going to use before you install MQSeries for MVS/ESA. "MQSeries interoperability summary" on page 210 shows the connections that are possible with networks containing MQSeries products.

Having decided on which form of connection to use, you need to define the channel definitions that are to be used in the exchange of messages.

For further information about distributed queuing, refer to the *MQSeries Intercommunication* manual. Refer to the *CICS for MVS/ESA Intercommunication Guide* for details on managing remote CICS for MVS/ESA links.

Planning recovery services

Chapter 21, "Backup and recovery planning for MQSeries for MVS/ESA" on page 109 introduces the concepts of recovery management in MQSeries for MVS/ESA subsystems. This chapter leads you into the planning you must do to ensure that your data can be recovered in cases of network or system failures.

Planning data security

MQSeries for MVS/ESA does not provide any security facilities of its own, but uses those provided by existing security management facilities. This book explores the possible options you must select from to protect your data and users from unauthorized access.

Administration

A summary of the administration facilities provided by the MQSeries for MVS/ESA products is given in Chapter 23, "Administration of MQSeries for MVS/ESA" on page 117. Full details of these facilities can be found in the *MQSeries Command Reference*, and in the *MQSeries for MVS/ESA System Management Guide*.

Installing and customizing

MQSeries for MVS/ESA uses the standard MVS/ESA installation procedure. This section reminds you of that procedure and also introduces some of the work that must be planned in order to customize MQSeries for MVS/ESA to your enterprise's particular needs.

Before you install MQSeries for MVS/ESA, you must decide the following:

- Which communications protocol you are going to use
- Whether you are going to install one of the following optional national language features:
  - Japanese
  - Simplified Chinese
  - US English (upper case)
- Whether you are going to install the optional client attachment feature

You also need to plan how much storage you require in your MVS/ESA system to accommodate MQSeries for MVS/ESA. Assistance is given in Chapter 24, "Storage planning for MQSeries for MVS/ESA" on page 121 to help you plan the amount of storage required for MQSeries for MVS/ESA. The chapter also gives an indication of the amount of storage you should plan to use, for example, for logs, page data sets, and objects.

Installation

MQSeries for MVS/ESA is supplied with a Program Directory that contains specific instructions for installing the program on an MVS/ESA system. You must follow the instructions in the *MQSeries for MVS/ESA Program Directory.* They include not only details of the installation process but also information about the necessary prerequisite products and their service or maintenance levels.

SMP/E, used for installation on the MVS/ESA platform, validates the service levels, prerequisite and corequisite products, and maintains the SMP/E history records to record the installation of MQSeries for MVS/ESA. It loads the MQSeries for MVS/ESA libraries and checks that the loads have been successful. You then have to customize the product to your own requirements.

Customization

MQSeries requires some customization after installation in order to meet the individual and special requirements of your system, and to use your system resources in the most effective way. Customization is described in detail in the *MQSeries for MVS/ESA System Management Guide*. However, below are the items you must consider when planning to customize your system.

You need to:

- Define the MQSeries for MVS/ESA subsystem to MVS
- Authorize the MQSeries for MVS/ESA load libraries
- Include the MQSeries for MVS/ESA load library in the link list
- Include the MQSeries for MVS/ESA dump formatting member
- Update the MVS/ESA Program Properties Table (PPT)
- Create procedures for the MQSeries for MVS/ESA subsystem
- Tailor your security procedures

- Customize the initialization files
- Create the bootstrap and log data sets
- Define your page sets
- Tailor your logging environment
- Install required CICS and IMS adapters
- Define queues
- Set up distributed queuing

Verifying your installation

After the installation and customization has been completed, you can use an installation verification program (IVP) supplied with the product to verify that the installation has been completed successfully. The IVP supplied is an assembler language program and should be run after MQSeries for MVS/ESA has been customized to suit your enterprise's needs. Details of the IVP and customization are given in the *MQSeries for MVS/ESA System Management Guide*.

Migrating from previous versions

If you are migrating from a previous version of MQSeries for MVS/ESA you do not have to perform most of the customization tasks listed in "Customization" on page 102. Information about what you need to do when migrating from a previous version is given in the *MQSeries for MVS/ESA System Management Guide*.

Chapter 20. Data sets used by MQSeries for MVS/ESA

This chapter provides a general introduction to the specialized data sets used by MQSeries for MVS/ESA. MQSeries for MVS/ESA uses data sets called *page sets* for storing messages. It maintains *logs* of data changes and significant events as they occur to provide backup and recovery facilities. The logs contain information about queues and messages. The *bootstrap data set (BSDS)* stores information about the data sets that contain the logs.

This chapter contains basic information about:

Page sets

A page set is a linear VSAM data set that has been formatted for use by MQSeries for MVS/ESA. Page sets are used primarily to store messages and object definitions. Each page set is identified by a page set ID (PSID), an integer in the range 00 through 99. In particular, MQSeries for MVS/ESA uses page set 00 to store queue definitions and other important information relevant to the queue manager.

Management of page sets is done through the use of buffer pools and storage classes.

Buffer pools and buffers

For efficiency, MQSeries for MVS/ESA implements a form of caching whereby messages are stored temporarily in buffers before being stored in page sets in DASD.

The buffers are organized into buffer pools. You can define up to four buffer pools for each MQSeries subsystem. Each buffer is 4 KB long. The maximum number of buffers is determined by the amount of storage available in the MQSeries address space. You are recommended to use four buffer pools.

Storage classes

A storage class maps one or more queues to a page set. When you define a queue you can also specify its storage class. More than one queue can use the same storage class, and you can define as many storage classes as you wish.

Log data sets

MQSeries for MVS/ESA records all *persistent* messages in the *active log* as they are put onto queues by applications. The active log resides on DASD. Entries to it are placed into a log buffer in main storage. The log buffer is written to main storage when a persistent message is placed in it or when it reaches a threshold value set at customization time. The log contains the information needed to recover messages, queues, and the queue manager; it does not contain information on statistics, tracing, or performance evaluation. (The *MQSeries for MVS/ESA System Management Guide* shows how to specify destinations for trace, statistic, and performance data.) The active log contains information from all the queues being used with MQSeries for MVS/ESA.

When the active log is full, MQSeries for MVS/ESA switches to the next available log data set and, if archiving has been switched on during customization, copies the contents of this log to an *archive log*. The archive log can be a data set on a direct access storage device (DASD) or on magnetic tape. If there is a problem, MQSeries for MVS/ESA uses these log entries to restore the message queues.

The archive log consists of up to 1000 sequential data sets. Each data set can be cataloged using the Integrated Catalog Facility (ICF).

MQSeries for MVS/ESA allows you to have either *single logging* or *dual logging*. Dual logging is used to minimize the likelihood of problems during restart. When dual logging is in use, MQSeries for MVS/ESA records the same information into two data sets.

Single logging gives you between 2 and 53 active log data sets. Dual logging gives you between 4 and 106 active log data sets. If possible, when using dual logging, the log data sets should be on separate volumes. This reduces the risk of them both being lost if the volume is corrupted or destroyed. Each active log data set is a single-volume, single-extent VSAM entry-sequenced data set (ESDS).

There is no relationship between the number of queues and the number of log data sets that you have. Instead, the number of data sets required is dependent on the amount of message traffic that you plan for. An algorithm is given in “Logs and archive storage” on page 122 to help you to calculate the amount of storage you should reserve for your logs.

For a complete description of logs, their contents, customization, and archiving, see the *MQSeries for MVS/ESA System Management Guide.*

Bootstrap data set

The *bootstrap data set* (BSDS) is a VSAM key-sequenced data set (KSDS) that holds information needed by MQSeries for MVS/ESA. It contains:

- An inventory of all active and archived log data sets known to MQSeries for MVS/ESA. This inventory is used by MQSeries for MVS/ESA to:
  - Track the active and archived log data sets
  - Locate log records so that it can satisfy log read requests during normal processing
  - Locate log records so that it can handle restart processing

  MQSeries for MVS/ESA stores information in the inventory each time an archive log data set is defined or an active log data set is reused. For active logs, the inventory shows which are full and which are available for reuse. The inventory holds the relative byte address (RBA) of each log data set. There can be more than one RBA if the log data set spans more than one volume.
- A *wrap-around* inventory of all recent MQSeries for MVS/ESA activity. This is needed if MQSeries for MVS/ESA has to be restarted.

The active logs are first registered in the BSDS when MQSeries for MVS/ESA is initiated. They cannot be replaced, nor can new ones be added, without terminating and restarting MQSeries. The BSDS is required if the subsystem has an error and has to be restarted. To minimize the likelihood of problems during a restart, MQSeries can be configured with dual BSDSs, each recording the same information. This is known as running in “dual mode”. As for dual active log data sets, copies should, if possible, be on separate volumes. This reduces the risk of them both being lost if the volume is corrupted or destroyed.

Archive log data sets and BSDS copies

A copy of the BSDS is placed in each new archive data set. If the archive log is on tape, the BSDS copy is the first file on the first output volume. If the archive log is on DASD, the BSDS copy is a separate file on the same volume.

What a log contains

An active log can contain up to 2^{48} bytes. Each byte in the active log can be addressed by its offset from the beginning of the log, and that offset is known as its *relative byte address* (RBA).

The log is made up of *log records*, each of which is a set of log data treated as a single unit. A log record is identified by the RBA of the first byte of its header; that RBA is called the relative byte address *of the record*. The RBA is like a time stamp because it uniquely identifies a record that starts at a particular point in the log.

Each log record has a header that gives its type, the MQSeries component that caused the record to be made, and, for unit of recovery records, the unit of recovery identifier.

Most of the log records describe changes to MQSeries queues. All such changes are made within units of recovery.

Checkpoint records

To reduce restart time, MQSeries for MVS/ESA takes periodic checkpoints during normal operation:

- When a predefined, customized number of log records has been written
- At the end of a successful restart
- At normal termination

At a checkpoint, MQSeries logs its current status and registers the RBA of the checkpoint in the bootstrap data set (BSDS). At restart, MQSeries uses the information in the checkpoint records to reconstruct the state it was in when it terminated.

Many log records can be written for a single checkpoint.

Chapter 21. Backup and recovery planning for MQSeries for MVS/ESA

Developing backup and recovery procedures at your site is vital to avoid costly and time-consuming losses of data. MQSeries for MVS/ESA provides the means for recovering both queues and messages to their current state after a system failure. You should develop procedures for backing up page sets and creating a point of consistency.

This chapter describes what you should consider to minimize problems following any system failure. It contains information about:

Planning your logging environment

The MQSeries for MVS/ESA logging environment is established during customization to specify options, such as whether to have single or dual active logs, what media to use for the archive log volumes, and how many log buffers to have.

In a production subsystem, it is important to establish dual archiving and dual logging to minimize the risk of losing your data (for example, because of DASD failures).

Planning your archive storage

This section describes the different ways of maintaining your archive log data sets.

Archive log data sets can be placed on standard-label tapes, or DASD, and can be managed by Data Facility Hierarchical Storage Manager (DFHSM).

Archive log data sets are dynamically allocated, with names chosen by MQSeries for MVS/ESA. The data set name prefix, block size, unit name, and DASD sizes needed for such allocations are specified when MQSeries for MVS/ESA is customized. You can also choose, at customization time, to have MQSeries for MVS/ESA add a date and time to the archive log data set name. You can change this information by customizing MQSeries for MVS/ESA again.

If you specify dual archive logs at installation time, each log *control interval* (CI) retrieved from the active log is written to two archive log data sets. The log records that are contained on the pair of archive log data sets are identical, but the end-of-volume points are not synchronized for multivolume data sets.

Other recovery considerations

In addition to the logging and archiving facilities provided by MQSeries for MVS/ESA, you should also consider other space management and recovery facilities that are provided in your enterprise. The following sections provide a reminder of some of these facilities.

CICS recovery

CICS recognizes MQSeries for MVS/ESA as a non-CICS resource (or external resource manager), and includes MQSeries for MVS/ESA as an agent in any syncpoint coordination requests using the CICS resource manager interface (RMI). For more information about CICS recovery, see the *CICS for MVS/ESA Recovery and Restart Guide*. For information about the CICS resource manager interface, see the *CICS for MVS/ESA Customization Guide*.

IMS recovery

IMS/ESA recognizes MQSeries for MVS/ESA as an external subsystem and as a participant in syncpoint coordination. IMS recovery for external subsystem resources is described in the *IMS/ESA Version 4 Customization Guide: System*.

Backup and recovery with DFHSM

The data facility hierarchical storage manager (DFHSM) does automatic space and data availability management among storage devices in your system. If you use it, you need to know that it moves data to and from MQSeries for MVS/ESA storage automatically.

DFHSM manages your DASD space efficiently by moving data sets that have not been used recently to less expensive storage. It also makes your data available for recovery by automatically copying new or changed data sets to tape, or DASD, backup volumes. It can delete data sets, or move them to another device. Its operations occur daily, at a specified time, and allow for keeping a data set for a predetermined period before deleting or moving it.

All DFHSM operations can also be performed manually. The *Data Facility Hierarchical Storage Manager User's Guide* explains how to use the DFHSM commands.

If you use DFHSM with MQSeries for MVS/ESA, be aware that DFHSM:

- Uses cataloged data sets
- Operates on page sets and logs

Using Extended Recovery Facility (XRF)

MQSeries for MVS/ESA can be used in an Extended Recovery Facility (XRF) environment. All MQSeries for MVS/ESA-owned data sets (executable code, BSDSs, logs, and page sets) must be on DASD shared between the active and alternate XRF processors. If you use XRF for recovery, you must stop MQSeries for MVS/ESA on the active processor and start it on the alternate. For CICS, this can be done using the command list table (CLT) provided by CICS, or manually by the system operator. For IMS, this is a manual operation and must be done after the coordinating IMS system has completed the processor switch. MQSeries for MVS/ESA utilities must be completed or terminated before MQSeries for MVS/ESA

can be switched to the alternate processor. Consider the effect of this potential interruption carefully when planning your XRF recovery plans.

Take care to prevent MQSeries for MVS/ESA starting on the alternate processor before the MQSeries for MVS/ESA system on the active processor terminates. A premature start can cause severe integrity problems in data, the catalog, and the log. Using global resource serialization (GRS) helps avoid the integrity problems by preventing simultaneous use of MQSeries for MVS/ESA on the two systems. The BSDS must be included as a protected resource, and the active and alternate XRF processors must be included in the GRS ring.

Preparing for disaster recovery

In the case of a total loss of an MQSeries for MVS/ESA computing center, you can recover on another MQSeries for MVS/ESA system at a recovery site. To be able do this, you must regularly back up the data sets and logs and provide the means to transfer them to the recovery site if necessary.

General tips for backup and recovery

The MQSeries for MVS/ESA restart process recovers your data to a consistent state by applying log information to the page sets. If your page sets are damaged or unavailable, you can resolve the problem using your backup copies of your page sets (provided that all the logs are available). If your log data sets are damaged or unavailable, it might not be possible to recover completely. It is recommended that you do the following:

- Periodically take backup copies
- Use dual logging for your active log, archive log, and bootstrap data sets
- Keep archive logs you might need
- Retain the DD name or page set association

These are described in more detail below.

Periodically taking backup copies

A *point of consistency* is the term used to describe a set of backup copies of MQSeries for MVS/ESA page sets and the corresponding log data sets required to recover these page sets. These backup copies provide a potential restart point in the event of page set loss (for example, page set I/O error). If MQSeries for MVS/ESA were to be restarted using these backup copies, the data in MQSeries for MVS/ESA would be consistent up to the point that these copies were taken. Providing that all logs are available from this point, MQSeries for MVS/ESA can be recovered to the point of failure. See the *MQSeries for MVS/ESA System Management Guide* for more information about points of consistency.

The more recent your backup copies the quicker MQSeries for MVS/ESA can recover the data in the page sets. The recovery of the page sets is dependent on all the necessary log data sets being available.

In planning for recovery, you need to determine how often to take backup copies and how many complete backup cycles to keep. These values tell you how long you must keep your log data sets and backup copies of page sets for MQSeries for MVS/ESA recovery.

In deciding how often to take backup copies, consider the time needed to recover a page set. It is determined by:

- The amount of log to traverse
- The time it takes an operator to mount and remove archive tape volumes
- The time it takes to read the part of the log needed for recovery
- The time needed to reprocess changed pages

In general, the more often you make backup copies, the less time recovery takes; but, of course, the more time is spent making copies.

You should keep at least two copies of each cycle of page set backup. This reduces the risk involved if one backup copy is lost or damaged.

Using dual logging for your log data sets

This increases the chances of recovering from all problems. However, it also increases the processing overhead and will affect the performance of your system. You need to consider the gains and losses caused by dual logging.

If you use dual logging, keep the dual logs separate.

Keeping archive logs you might need

MQSeries for MVS/ESA can use archive logs during restart. You must keep sufficient archive logs so that the system can be fully restored. MQSeries for MVS/ESA might need to use an archive log to recover a page set from a restored backup copy. If you have discarded that archive log, MQSeries for MVS/ESA will not be able to restore the page set to its current state and you will have to resolve the problem manually. See the *MQSeries for MVS/ESA System Management Guide* guide for details of how recovery is effected using the archive log.

Retaining the DD name or page set association

MQSeries for MVS/ESA associates page set number 00 with DD name CSQP0000, page set number 01 with DD name CSQP0001, and so on up to CSQP0099. MQSeries for MVS/ESA writes recovery log records for a page set based on the DD name the page set is associated with. For this reason, you must not move or rename page sets that have already been associated with a page data set ID (PSID) DD name; otherwise, MQSeries for MVS/ESA tries to recover the page sets with the wrong log data.

Chapter 22. Security planning for MQSeries for MVS/ESA

Because MQSeries for MVS/ESA handles information passing between subsystems, it needs the safeguard of a security system to ensure that the resources it owns and manages are protected and secure from unauthorized access. It needs to ensure that the following are not accessed or changed by any unauthorized person or process:

- Connections to the MQSeries for MVS/ESA subsystem
- Resources such as queues, processes, and namelists
- MQSeries links to remote queue managers
- MQSeries system control commands
- MQSeries messages
- Context information in messages

To provide the necessary security, MQSeries for MVS/ESA uses the MVS System Authorization Facility (SAF) to route authorization requests to an external security manager (ESM), such as the Resource Access Control Facility (RACF). MQSeries does no verification of its own.

This book assumes that you are using RACF. If you are using a different ESM, you might need to modify the techniques mentioned in this book.

The decision to allow access to an object is made by the ESM. MQSeries follows the decision made by the ESM. If the ESM cannot make a decision, MQSeries does not allow access to the object.

If you are planning a distributed system, you must agree on the levels of security checking that are provided and used by the various administrative domains that exist across these distributed systems. Such agreement must also cover network security aspects and access by clients.

This chapter includes basic information about:

- “Security overview” on page 114
- “Security exits” on page 115
- “Things to consider” on page 115

For further information about security facilities in MQSeries for MVS/ESA, refer to the *MQSeries for MVS/ESA System Management Guide*.

Security overview

MQSeries for MVS/ESA provides you with a range of choices when specifying your security requirements. The choice ranges from no security checking to full checking.

Subsystem security

Subsystem security allows you to control whether any security checking is done on the whole subsystem. You can set security checking according to how secure you decide your users are in a particular subsystem. For example, you might apply full security checking on your production system, but have no checking on your test system.

If you decide you want subsystem security, you also need to consider the following:

- Connection security

  Do you want to control who and what connects to your MQSeries for MVS/ESA subsystem?

- API resource security

  Resources can be security checked when a user issues MQOPEN or MQPUT1 calls to the API. The security checking of the API resources is subdivided into the following:

  – Queue security

    Queue security allows you to control who is allowed to open a queue and how they are allowed to access it.

  – Process security

    Process security allows you to control who is allowed to open a process.

  – Namelist security

    Namelist security allows you to control who is allowed to open a namelist.

  – Context security

    Context information consists of two parts, an identity section and an origin section. The identity section specifies who the message came from; the origin section specifies where the message came from and when it was put to the queue.

    Context security allows you to control who is allowed to use the context-related options.

  – Alternate user security

    Alternate user security is used to control whether a particular user can use another user ID's authority to open an object on that user ID's behalf.

- Reslevel security

  Reslevel security is not a type of security but an option that allows you to control the number of user IDs checked for MQSeries for MVS/ESA API resource security checking (where applicable).

- Command security

  Command security allows you to control who is allowed to issue an MQSeries for MVS/ESA command.

- Command resource security

  Command resource security allows to you to control the resources a command can access. Command resource security is independent of API security. For example, user A might be allowed to define a local queue but might not be allowed to get messages from that queue.

Security classes

MQSeries for MVS/ESA uses its own special RACF classes to hold its security information. These special classes are described in the *MQSeries for MVS/ESA System Management Guide*.

Security exits

The message channels that are used for distributed queuing and the MQI channels that are used between clients and servers, have security exit facilities that can invoke programs supplied by you. Examples of the types of function for which these exit facilities are intended include:

- Verification that the partners at the ends of the channel are genuine, and have the appropriate security authorizations to take part in the exchange
- Encryption and decryption of messages

For more information about security exits, refer to the *MQSeries Intercommunication* manual.

Things to consider

In order to help you set up your security you need to consider the following:

- Decide what levels of security you require (from above list).
- Decide what types of resources are going to be used and, preferably, set up a naming convention for them (for example, queues and their names).
- Decide who will be using the system and the resources you have. Consider which users and user IDs will use which resources and so on.
- Decide how they are going to be using the system and what access levels each user requires for each resource they use.
- Decide any grouping of users and resources that might be required.
- Decide which users will be accessing or connecting to MQSeries for MVS/ESA and how they will be doing so; for example, batch, CICS, IMS, and so on.
- Decide on how to manage network security, for example to ensure that all incoming messages are from authorized applications or users and that all outgoing messages are from authentic users and are secure.

For details on how the steps can be carried out to satisfy your security requirements, refer to the *MQSeries for MVS/ESA System Management Guide*.

Chapter 23. Administration of MQSeries for MVS/ESA

This chapter is a summary of the administration facilities provided by MQSeries for MVS/ESA. It has the following sections:

Details of the administration facilities provided by MQSeries for MVS/ESA are given in the *MQSeries Command Reference*, and in the *MQSeries for MVS/ESA System Management Guide*.

Managing objects

It is the administrator's job to monitor MQSeries for MVS/ESA and make any changes that might be necessary. To do this, the administrator needs to know where each MQSeries object resides, what its characteristics are, and who has access to it. The administrator can manage and monitor the resources by use of the commands, the operations and control panels, or the utility programs supplied with MQSeries for MVS/ESA.

If there are sets of commands that are issued regularly on your system, you can run a supplied utility program, or write a program that constructs commands and places them on the command queue. Details of how to write these administration programs are contained in the *MQSeries for MVS/ESA System Management Guide*.

Commands on MQSeries for MVS/ESA

MQSeries for MVS/ESA supports MQSC commands, which can be issued from the following sources:

- The MVS console.
- The initialization input data set, CSQINP1, to be processed before the restart phase of MQSeries for MVS/ESA initialization.
- The initialization input data set, CSQINP2, to be processed after the restart phase of MQSeries for MVS/ESA initialization.
- The initialization input data set, CSQINPX, to be processed whenever the channel initiator is started.
- The supplied batch utility, CSQUTIL, processing a list of commands in a sequential data set.
- A suitably authorized application, by sending a command as a message to the command queue. This can be either:
  - A batch region program
  - A CICS application
  - An IMS application
  - A TSO application
  - An application program or utility on another MQSeries system

Much of the functionality of these commands is available in a user-friendly way from the MQSeries for MVS/ESA operations and controls panels.

Information on using commands, and on using the operations and controls panels is given in the *MQSeries for MVS/ESA System Management Guide*.

MQSeries for MVS/ESA can access security checks to ensure that the user is authorized to issue particular commands for particular resources.

Changes made to the resource definitions of a queue manager using the commands (directly or indirectly) are preserved across restarts of the MQSeries subsystem.

MQSeries for MVS/ESA does not support the PCF commands.

Managing communications

Part of the administrator's role is to ensure that the required communications links are activated, and to monitor the status of these links as required by your enterprise. You can find information describing these tasks in the *MQSeries Intercommunication* manual.

Remote administration

There are two aspects to the MQSeries remote administration facilities:

- MQSeries for MVS/ESA can be used to manage remote systems
- Other remote products can be used to manage an MQSeries for MVS/ESA system

Managing remote systems

Your administrator can manage remote MQSeries for MVS/ESA systems using facilities provided by MQSeries for MVS/ESA.

Because of the differences between the MQSeries products, it is not always possible to manage remotely the same set of objects or attributes that you can manage locally.

If you wish to manage any other MQSeries product, you can write an application program to send the appropriate commands to the queue manager command queue. However, some MQSeries products do not have a command queue, and they cannot accept commands from local or remote application programs.

Managing MQSeries from remote systems

MQSeries for MVS/ESA can be managed from a remote MQSeries system, by an administrator using the facilities provided by the following products:

- MQSeries for Digital OpenVMS
- MQSeries for MVS/ESA
- MQSeries for OS/2 Warp
- MQSeries for Tandem NSK
- MQSeries on UNIX systems
- MQSeries for Windows NT

Managing accounting information

MQSeries for MVS/ESA provides an accounting trace to enable you to collect information that you can use to charge your customers for their use of your MQSeries for MVS/ESA subsystem. This facility is described in the *MQSeries for MVS/ESA System Management Guide*.

Using the utilities

MQSeries for MVS/ESA supplies a set of utility programs to help you perform various administrative tasks. These utilities:

- Perform backup, restoration, and reorganization tasks
- Issue commands and process object definitions
- Generate data-conversion exits
- Modify the bootstrap data set
- List information about the logs
- Print the logs

For more information about all these utilities, see the *MQSeries for MVS/ESA System Management Guide*.

The CSQUTIL utility

The CSQUTIL utility program is provided with MQSeries for MVS/ESA to help you perform backup, restoration, and reorganization tasks, and to issue commands and process object definitions. Through this utility program, you can invoke the following functions:

COMMAND To issue any of the MQSC commands described in the *MQSeries Command Reference* manual, to record object definitions, and to make client-channel definition files.

COPY To read the contents of a named MQSeries for MVS/ESA message queue or the contents of all the queues of a named page set, and put them into a sequential file and retain the original queue.

COPYPAGE To copy whole page sets to larger page data sets.

EMPTY To delete the contents of a named MQSeries for MVS/ESA message queue or the contents of all the queues of a named page set, retaining the definitions of the queues.

FORMAT To format MQSeries for MVS/ESA page sets.

LOAD To restore the contents of a named MQSeries for MVS/ESA message queue or the contents of all the queues of a named page set from a sequential file created by the COPY function.

RESETPAGE To copy whole page sets to other page set data sets and reset the log information in the copy.

SCOPY To copy the contents of a queue to a data set while the queue manager is offline.

SDEFS To produce a set of define commands for objects while the queue manager is offline.

The data conversion exit utility

The MQSeries for MVS/ESA data conversion exit utility (CSQUCVX) runs as a stand-alone utility to create data conversion exit routines.

The change log inventory utility

The MQSeries for MVS/ESA change log inventory utility program (CSQJU003) runs as a stand-alone utility to change the bootstrap data set (BSDS). The utility can be used to:

- Add or delete active or archive log data sets
- Supply passwords for archive logs

The print log map utility

The MQSeries for MVS/ESA print log map utility program (CSQJU004) runs as a stand-alone utility to list the following information:

- Log data set name and log RBA association for both copies of all active and archive log data sets

  Note: If dual logging is not active, then there is only one copy of the data sets.

- Active log data sets available for new log data
- Contents of the queue of checkpoint records in the bootstrap data set (BSDS)
- Contents of the archive log command history record
- System and utility time stamps

The log print utility

The log print utility program (CSQ1LOGP) is run as a stand-alone utility. You can run the utility specifying:

- A bootstrap data set (BSDS)
- Active logs (with no BSDS)
- Archive logs (with no BSDS)

Chapter 24. Storage planning for MQSeries for MVS/ESA

This chapter tells you how to plan the type and amount of storage you require when you include MQSeries for MVS/ESA in your system. It contains information about:

MQSeries for MVS/ESA is an MVS/ESA subsystem running in its own address space. The virtual storage usage therefore falls into two categories: the storage used by the subsystem address space itself and the additional storage used by applications requesting MQSeries for MVS/ESA services.

Address space storage

Each MQSeries for MVS/ESA subsystem has the following storage requirements:

CSA 48 KB
ECSA 1.3 MB

In addition, each concurrent MQSeries task requires about 1500 bytes of ECSA. When a task ends, this storage can be reused by other MQSeries tasks. MQSeries does not release the storage until the queue manager is shut down, so the maximum amount of ECSA required can be calculated by multiplying the maximum number of concurrent tasks by 1500 bytes.

Concurrent tasks consist of the following:

- The number of Batch, TSO or IMS regions that have connected to MQSeries, but not disconnected
- The number of CICS transactions that have issued an MQSeries request, but have not terminated

The trace table also resides in the ECSA; you should use the TRACTBL parameter of the CSQ6SYSP macro to determine the size of the resident trace table. This macro is described in the *MQSeries for MVS/ESA System Management Guide*.

Every channel uses approximately 140 KB of extended user region in the channel initiator (CHINIT) address space, plus a further 1 KB below the 16 MB line (12 KB if LE/370 APAR PQ03507 has not been applied). Storage is increased if messages larger than 32 KB are being transmitted.

Logs and archive storage

Active log data sets record significant events and data changes. They are periodically off-loaded to the archive log. Consequently, the space requirements for your active log data sets depend on the volume of messages that your MQSeries for MVS/ESA handles and how often the active logs are off-loaded to your archive data sets. MQSeries for MVS/ESA provides optional support for dual logging; if you use this your log storage requirement will be doubled.

If you decide to place the archive data sets on direct access storage devices (DASD), you need to reserve enough space on the devices. Space should also be reserved for the bootstrap data sets (BSDS). The functions of these data sets are described in Chapter 20, “Data sets used by MQSeries for MVS/ESA” on page 105. The above are all separate data sets and should, preferably, be allocated space on different volumes and strings to minimize DASD contention and problems caused by any defects on the physical devices.

As each change to the system is logged, the size of storage required can be estimated from the size and expected throughput of messages. You must add to this a small overhead for the header information in the data sets.

To arrive at the size of the log extents, an algorithm can be developed which depends on various factors including the message rate, the message size in bytes, and how often you want to switch the log.

Below is shown an approximate calculation for the number of records to specify in the cluster for the log data set.

```
Number of records = (a * log switch interval required in seconds) / 4096

 where a =  (Number of puts/sec * Average message size)+440
          + (Number of gets/sec * 72)
          + (Number of units of recovery started  * 100)
          + (Number of syncpoints per second * 196)
```

Storage for page data sets and messages

The amount of storage needed for page data sets depends on the sizes of the messages that your applications will exchange, on the numbers of these messages, and on the rate at which they are created or exchanged. You can calculate the amount of storage needed for page data sets using the algorithm given in the *MQSeries for MVS/ESA System Management Guide*.

Storage for bootstrap data sets (BSDS)

Each BSDS requires 500 KB. You must have one BSDS but, optionally, can choose to have dual BSDSs. Further information about choosing single or dual BSDSs is given in Chapter 20, “Data sets used by MQSeries for MVS/ESA” on page 105.

Planning your library storage

You need to allocate storage for the product libraries. The exact figures depend on your configuration, but an estimate of the space required by the distribution libraries is 35 MB. The target libraries also require about 35 MB. Additionally, you require space for the SMP/E libraries.

You should refer to the program directory supplied with MQSeries for MVS/ESA for information about the required libraries and their sizes.

Further information

This chapter has shown some of the main considerations when planning the storage required for MQSeries for MVS/ESA. For more information about the library sizes, refer to the *MQSeries for MVS/ESA Program Directory.* For more information about storage variables to be defined during customization, refer to the *MQSeries for MVS/ESA System Management Guide*.

Chapter 25. Performance of MQSeries for MVS/ESA

This chapter introduces the performance considerations that you must review when planning to include MQSeries for MVS/ESA in your system. These items include:

Impact of logging

The performance of MQSeries for MVS/ESA is sensitive to logging activities. MQSeries for MVS/ESA provides and maintains data integrity by the use of logs to record messaging events. If no logging were to take place, the performance of the system would depend primarily on processor availability.

Significant logging takes place mainly when using persistent messages. Hence, performance is affected by the number, and size, of persistent messages in the system, and also by the frequency of syncpointing and DASD performance for the log data sets.

Figure 10 shows the response overhead added by logging for various message sizes in a typical CICS for MVS/ESA 3.3 region. You will see that the response time increases by about 15%.

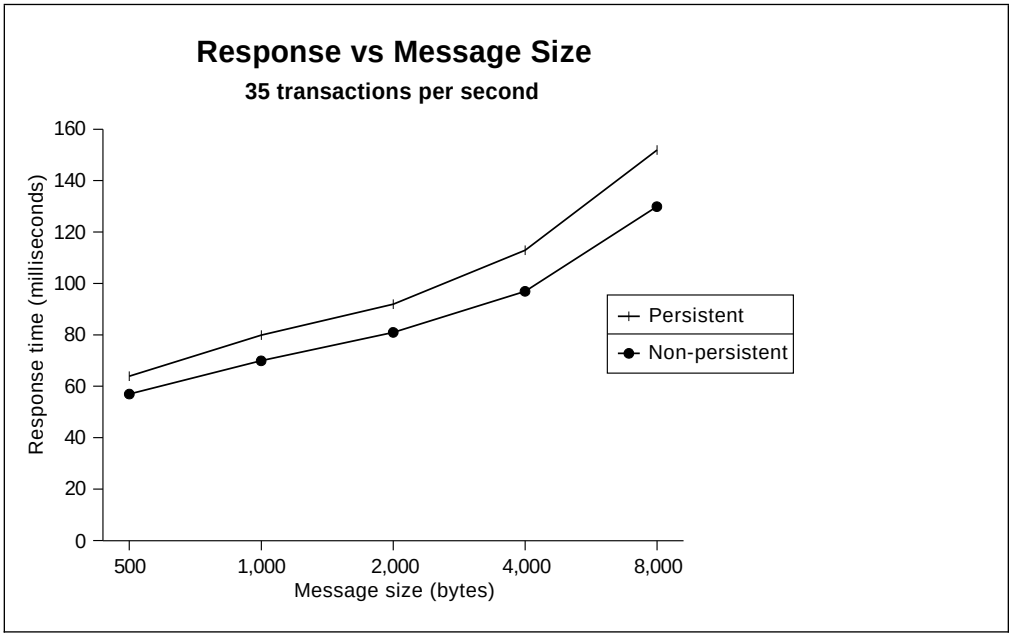

Figure 10. Impact of logging on response time

To achieve the best performance:

- Put active logs on the fastest DASD (for example, use DASD fast write).
- Ensure that when an active log is in use it is the only data set on that volume.

Impact of dual logging

MQSeries for MVS/ESA offers the capability of having dual logs. These logs should be placed on separate volumes. This, as well as giving more protection against unexpected corruption of data on a volume, ensures the log writes are not competing for disk accesses on the same volume and are therefore faster. Figure 11 shows the overhead in terms of transaction response times in a CICS region with and without dual logging.

Each transaction in Figure 11 consists, on average, of five put or get messages, and covers a range of about 300 to 500 log events per second.

Note: The fast write facility is turned on in this example. For an explanation see “Fast write for logging” on page 127.

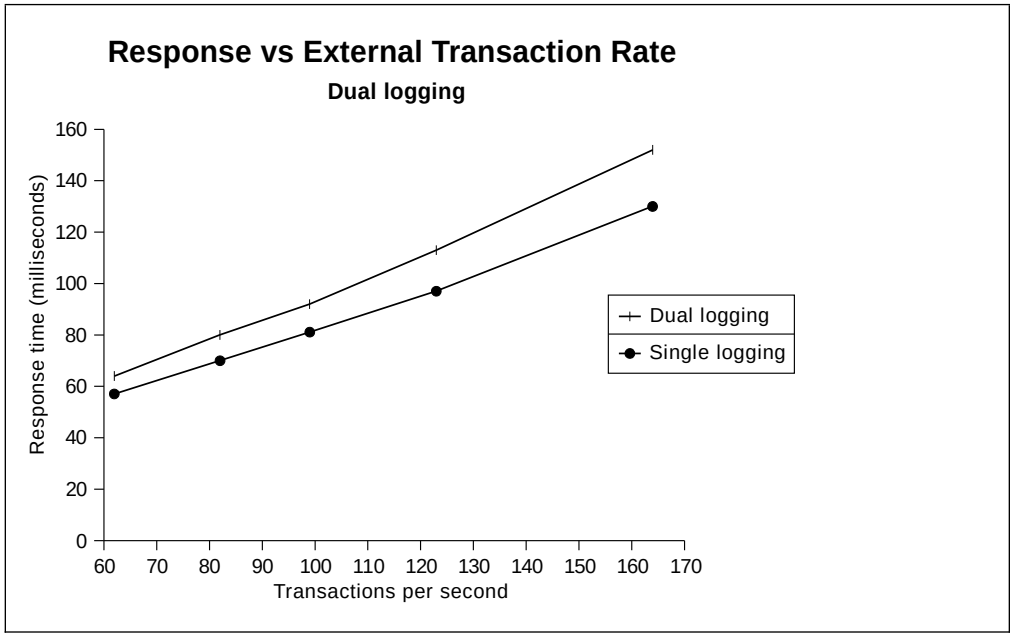

Figure 11. Impact of dual logging on response time

Fast write for logging

Fast write is a facility provided on some DASD controllers. It is a cache that is used to buffer write operations to DASD. Data can be written into the cache at channel speeds and is not held up by seek, rotational, and other delays in the DASD hardware.

Throughput gains can be made by directing the logs to a fast write facility if it is available. The facility significantly reduces I/O response times, as shown in Figure 12. The figure shows that if fast write is in use, a high throughput can be sustained beyond the point where, without fast write caching, the I/O time to complete would cause degradation in the transaction response time.

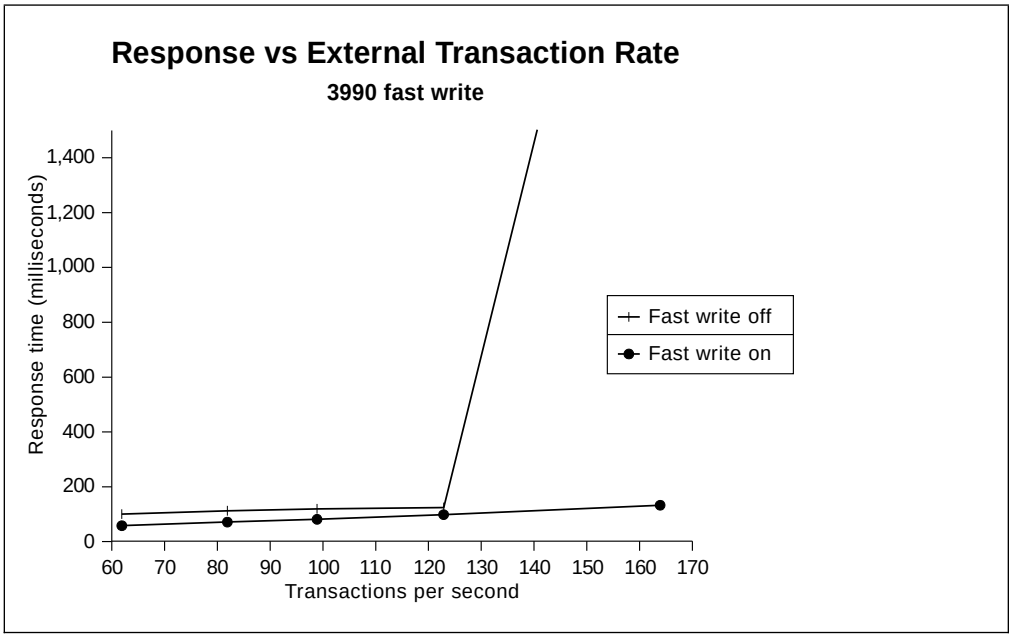

Figure 12. Impact of using 3990 fast write on response time

Causes of I/O to log

The active log buffer is written to DASD when a syncpoint is taken or when the threshold value set for flushing the buffer to DASD is reached. The log buffer is also written to DASD when a persistent message is placed on a queue and that message is not operating within a syncpoint.

Checkpointing

The number of messages written to the log buffer before a checkpoint is taken is defined when MQSeries for MVS/ESA is customized. Every two checkpoints, all pages that have not been written to the page data set since the last checkpoint are written to the page data set. In this way, no page can be older than two checkpoints. Two consequences of this are that the greater the number of messages held in the log buffer, the less paging is done, but the further back the checkpoint will be in the log. The further back in the log that the checkpoint is, the more of the log will have to be scanned during a recovery. In other words, the higher the number of messages held in the log buffer, the longer any recovery will take.

MQSeries page set I/O

Page set I/O is another activity that impacts performance. Hence, the more it can be avoided, the better the performance. You should specify enough buffers to contain the expected workload, so reducing excessive I/O. If the number you specify is insufficient, the I/O rate of the MVS/ESA system will increase.

At initialization you need to define the number of data buffers that are to be made available for use by queues. You can define between one and four buffer pools. Each buffer pool can hold between 100 and 125 000 buffers. You must define how many buffers you will need. Each buffer pool is also defined to a particular page data set ID (PSID).

You must also decide where the page data sets will be placed. When you have defined more than one PSID, it is best to locate the data sets on different physical volumes.

You will find more information about paging and how to define the parameters introduced above, in the *MQSeries for MVS/ESA System Management Guide* and in the *MQSeries Command Reference*.

Buffer pools, page sets, storage classes, and queues

- Buffer pools are areas of MQSeries for MVS/ESA virtual storage reserved to satisfy the buffering requirements for one or more queues.
- Page sets are VSAM linear datasets and are each associated with a buffer pool.
- Storage classes are used to provide a mapping between queues and page sets.
- A queue is associated with a page set, which is a VSAM data set that is used when MQSeries for MVS/ESA moves data (for example, queues and messages) from buffers in main storage to permanent backing storage (DASD).

This gives a hierarchy which provides the following mappings:

- A queue (via its storage class) is associated with a single page set
  - Many queues can map to the same page set
- A page set is associated with a single buffer pool
  - Many page sets can map to the same buffer pool

In general, it is best to allocate as much storage as possible for the buffer pools. The larger the pool, the more likely that a message put to a queue will stay in storage long enough for it to be retrieved, thus saving I/O operations to the backing page set.

You should try to separate short-lived messages from messages that exist for a long time (for example half an hour). This means using different queues for the different messages, allocating the queues to different page sets, and allocating the different page sets to different buffer pools.

For short-lived messages you should allocate a large buffer pool. Long lived messages are written to the page set when the buffer pool fills up, or the messages have existed over two checkpoints. They are written to disk to reduce the startup time if the queue manager ends abnormally. Having a large buffer pool causes a lot of activity at a checkpoint, when the queue manager writes out the ‘old’ messages. Having a small buffer pool causes the data to be written to the page set at a steady rate, and reduces the impact at a checkpoint.

Refer to the *MQSeries for MVS/ESA System Management Guide* for more information about tuning your system.

Monitoring performance

A number of tools to monitor the performance of an MQSeries for MVS/ESA subsystem are provided. These include:

- Trace
- Display commands
- CICS adapter statistics

Further tools are provided by MVS/ESA:

- System Management Facility (SMF)
- Generalized Trace Facility (GTF)

You can also use other IBM licensed programs:

- Service Level Reporter (SLR)
- Resource Management Facility (RMF)
- CICS Monitoring facility

The use of these tools to monitor MQSeries for MVS/ESA performance is described in the *MQSeries for MVS/ESA System Management Guide.*

In addition to using these tools, with MQSeries for MVS/ESA you can write your own application programs, to use the event messages generated by the instrumentation events facility to produce performance reports.

More information on using instrumentation events is given in the *MQSeries Programmable System Management* manual.

Where to find more information

Information about MQSeries performance is available on the Internet at:

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

Chapter 26. Measured usage license charges with MQSeries for MVS/ESA

Measured Usage License Charges (MULC) is a particular way of charging you for an IBM product that runs on an MVS/ESA system, based on how much use you make of the product. To determine the product usage, the MVS/ESA system records the amount of processor time that is used by the product when it executes.

MVS/ESA can measure how much processing time is spent in doing work on behalf of the MQSeries queue manager which is handling MQI calls, executing MQSeries commands, or performing some other action to support the messaging and queuing functions used by your application programs. The amount of processing time is recorded in a file at hourly intervals, and the hourly records are totalled at the end of a month. In this way, the total amount of time that has been used by the MQSeries for MVS/ESA product on your behalf is computed, and used to determine how much you should pay for your use of the MQSeries for MVS/ESA product that month.

MULC is implemented as follows:

- When MQSeries for MVS/ESA is installed, it identifies itself to MVS/ESA, and requests that the *System Management Facilities (SMF)* mechanism within MVS/ESA is to automatically measure how much processor time is used by the MQSeries for MVS/ESA product.
- When enabled, the MVS usage measurement facility collects usage figures for each hour of the day, and generates usage records that are added to a report file on disk.
- At the end of one full month, these usage records are collected by a program, which generates a report of product usage for the month. This report is used to determine the charge for the MQSeries for MVS/ESA product.

More details on MULC can be found in *MVS/ESA Support for Measured License Charges, GC28-1098*.

Part 5. Planning for MQSeries for OS/2 Warp and Windows

Chapter 27. Introduction to MQSeries for OS/2 Warp and Windows NT

The MQSeries for OS/2 Warp and MQSeries for Windows NT products are in two parts, the *server* and the *clients*. The server runs on a machine that is capable of running OS/2 Warp or Windows NT; the clients provided with the product are for AIX, DOS, HP-UX, OS/2 Warp, Sun Solaris, Windows NT, Windows 3.1, and Windows 95.

Do not confuse MQSeries for Windows NT Version 5.0 with MQSeries for Windows. (Version 2.1 of the MQSeries for Windows product also runs on Windows NT.) This 'lightweight' messaging and queuing product is described in Chapter 32, "Introduction to MQSeries for Windows" on page 151.

MQSeries provides the MQI programming interface for use by application programs that are running on the server or the client processor. More detail on clients and servers is given in Chapter 7, "Introduction to MQSeries clients and servers" on page 47.

This chapter includes basic information about:

- "Planning for MQSeries"
- "Support for Lotus Notes" on page 138
- "MQSeries and R/3" on page 138
- "Migration from MQSeries Version 2" on page 139

For more information on the hardware and software environments needed by MQSeries, see "MQSeries for OS/2 Warp" on page 228 and "MQSeries for Windows NT" on page 252.

Planning for MQSeries

This chapter helps you to plan for the introduction of MQSeries into your enterprise, and introduces the items that you need to consider when doing this planning.

There are several stages in planning for the use of MQSeries that you must go through. They are:

1. Preparing your applications for the use of MQSeries
2. Planning to include MQSeries in a network
3. Preparing to install MQSeries
4. Planning to set up MQSeries

A prime requirement for a message delivery system is that it must be reliable. Many functions are built into MQSeries to ensure that messages are not lost despite system failures, and are not delivered more than once. MQSeries uses logging and other facilities to support these functions.

You must also plan for the operation and administration of MQSeries in your enterprise and consider the implementation of an appropriate set of security facilities. Brief outlines of these planning operations are included in this chapter.

Preparing your applications for the use of MQSeries

MQSeries brings the Message Queue Interface (MQI) to your applications. This interface allows you to modify existing applications and to write new applications. The MQI removes much of the need to understand the network and communication systems that you use. Thus you can expect to generate applications more speedily than before. However, you must plan to take advantage of the MQI by planning its use in your applications and by understanding the ways in which it assists you.

You can find more information on how to use the MQI in your applications by referring to the *MQSeries Application Programming Guide*.

Interfacing with CICS

With MQSeries you can create application programs for the CICS for OS/2 or CICS for Windows NT transaction environment. These applications can use the MQI to communicate with CICS or non-CICS programs in any of the environments supported by the MQSeries products.

Planning to use MQSeries in a network

MQSeries uses the distributed queuing facility to exchange messages between MQSeries platforms, using either the SNA LU 6.2, TCP/IP, SPX, or NetBIOS transmission protocols.

You must consider how you will attach MQSeries to a network, and how you will define the channels that will be used to exchange messages.

According to the way that you have set your systems up, security checks can be performed at various times. You can provide your own security services, using the exit facilities provided by MQSeries. MQSeries also provides exits relating to DCE security.

“MQSeries interoperability summary” on page 210 shows the links that are possible to other MQSeries products, and the transmission protocols that can be used.

For further information about distributed queuing, refer to the *MQSeries Intercommunication* manual.

Installing MQSeries for OS/2 Warp

MQSeries for OS/2 Warp is installed by using Software Installer/2 (SI/2), an IBM product. Although this is a separate licensed program, an SI/2 module is provided with your copy of MQSeries for OS/2 Warp that requires no additional license to be purchased. The supplied version of SI/2 is restricted in such a way that it can only be used for the installation of MQSeries for OS/2 Warp.

To prepare for the actual installation, you need to plan how much disk space you require in your OS/2 Warp system to accommodate MQSeries for OS/2 Warp. Assistance is given in Chapter 31, “Storage planning for MQSeries for OS/2 Warp and Windows NT” on page 149 to help you plan the amount of space required.

You can find more information to help you with the installation of MQSeries for OS/2 Warp in the *MQSeries for OS/2 Warp V5.0 Quick Beginnings* booklet.

When you have installed MQSeries, you can run the supplied installation verification test. This is described in the *MQSeries for OS/2 Warp V5.0 Quick Beginnings* booklet.

Installing MQSeries for Windows NT

Information on installing MQSeries for Windows NT is given in the *MQSeries for Windows NT V5.0 Quick Beginnings* booklet.

When you have installed MQSeries, you can run the supplied installation verification test. This is described in the *MQSeries for Windows NT V5.0 Quick Beginnings* booklet.

Setting up MQSeries

After installation, MQSeries needs to be set up, and customized for your own use. This ensures that the appropriate operating system facilities are made available to MQSeries, and that your MQSeries system is correctly initialized and ready to work with your applications.

MQSeries uses configuration files to hold the product configuration information used for logging, communications protocols and installable components. After installing the product, you can edit these files to tailor the operation of the product to meet the requirements of your installation.

In addition, you need to do the following:

- Define queues
  - Consider your naming conventions for queues
- Define trigger processes
- Define remote links
  - Define associated transmission queues
  - Consider your naming conventions for remote queues
  - Consider your naming conventions for channels

Note: The characters within the names given to all MQSeries objects are case sensitive. Therefore, be very careful when defining the names of objects, to select the appropriate uppercase or lowercase characters.

You can find more information on the setting up and customizing processes for MQSeries in the *MQSeries System Administration* manual.

Planning recovery services

MQSeries provides logging services to allow backup and recovery of the messaging system. Chapter 28, “Backup and recovery planning for MQSeries for OS/2 Warp and Windows NT” on page 141 introduces you to these facilities, and to the items that you need to consider in order to include MQSeries in your backup and recovery plans.

You can find more information on the backup and recovery facilities provided by MQSeries in the *MQSeries System Administration* manual.

Planning data security

MQSeries provides a number of security facilities for use by your applications. An introduction to these facilities is given in Chapter 29, “Security planning for MQSeries for OS/2 Warp and Windows NT” on page 145.

You can find more information on the security facilities provided by MQSeries in the *MQSeries System Administration* manual.

Administration

A summary of the administration facilities provided by MQSeries are given in the *MQSeries System Administration* manual.

Support for Lotus Notes

MQSeries provides a Lotus Notes server add-in task that gives Lotus Notes applications access to MQSeries messaging. This allows Lotus Notes users to communicate with other systems connected by MQSeries.

Lotus Notes is a networked application that users can use to share information. Lotus Notes has two main components, the server and the client. The Lotus Notes server provides services to Lotus Notes clients and to other servers. The services provided include storage and replication of shared databases and mail routing. Lotus Notes clients connect to a Lotus Notes server to use shared databases, and also to read and send mail. The Lotus Notes server add-in task runs in either an MQSeries server or an MQSeries client on that Lotus Notes server.

The basic units of information in a Lotus Notes system are databases and the documents that they contain. A database can be used by one person, or shared among users who have common data requirements. Most databases in Lotus Notes reside on a Lotus Notes server.

MQSeries provides a Lotus Notes server add-in task that recognizes and interprets:

- Data from documents that Lotus Notes wants to send to MQSeries
- Messages from MQSeries sent in reply and used to update a Lotus Notes document

The mobile or remote Notes user can access MQSeries applications and data.

MQSeries and R/3

The MQSeries link for R/3 for Windows NT product provides an interface that enables you to integrate your R/3 application with applications running in other environments (including those on R/3 and R/2 environments).

The R/3 link works with the Application Link Enabling (ALE) layer of the R/3 system to transmit Intermediate Documents (IDocs) into and out of your R/3 system, using MQSeries messages and queues to carry the information. It extends the scope of your business by allowing you to link your R/3 applications to any other application that you can access through MQSeries, even when those applications require different data formats.

For more information, see the *MQSeries link for R/3 User’s Guide*.

Migration from MQSeries Version 2

When you have migrated from MQSeries Version 2 to MQSeries Version 5 you will be unable to revert to Version 2. You should back up your system before installing the new version. This will enable you to back off the upgrade if necessary. If you do this however, you will not be able to recover the work performed by MQSeries Version 5.

With MQSeries Version 5, the system default objects are created automatically when you use the crtmqm command to create a queue manager. The sample MQSC definition file, AMQSCOMA.TST, is no longer provided. If you have used AMQSCOMA.TST to customize your settings for MQSeries Version 2, and you want to use the same settings with Version 5, save your version of the file before you install MQSeries Version 5. You can then use this file to create the Version 2 default objects. Alternatively, you can generate a new MQSC definition file if required.

A list of the system default objects for MQSeries Version 5 is provided in the *MQSeries System Administration* manual.

Chapter 28. Backup and recovery planning for MQSeries for OS/2 Warp and Windows NT

This chapter describes the background concepts of recovery and restart. It contains the following sections:

More details of the logging and recovery facilities are given in the *MQSeries System Administration* manual.

Logging

The basic premise of a messaging system is that messages entered into the system are assured of delivery to the destination. One of the ways of ensuring that messages are not lost, is to maintain a record of the activities of the queue manager that handles the receipt, transmission, and delivery of messages.

MQSeries does this by recording all the significant changes to the data controlled or managed by the queue manager in a log. This process is called *logging*. The data changes that are logged include the puts and gets of persistent messages to and from queues, changes to queue attributes, and channel activities.

The purpose of logging is to create and maintain a log that:

- Keeps records of queue manager changes
- Keeps records of queue updates for use by the restart process
- Is a source for restoration of data should there be a hardware or software failure

Each MQSeries log consists of a log control file, together with one or more log files for the storage of data.

The log control file is, as its name implies, used to control and monitor the use of the log files. It contains information relating to the size, location, next available file, and other data related particularly to the log files themselves. All the log files within one log are the same size; there is a default value for this size, but you can override this value when you set up the log.

MQSeries provides a log dump facility (DMPMQLOG). This enables you to format and display the contents of the log when doing problem determination.

Types of logging

In MQSeries, there are two approaches to maintaining records of queue manager activities:

- Circular logging
- Linear logging

Each type of logging stores the recorded data in a set of files. The differences between the two types of logging are the contents, and the way that the files are linked together.

With circular logging, the set of log files are effectively linked together so as to form a ring. When data is collected, it is written sequentially into the files in such a way as to reuse the log files in the ring. You can use circular logging for:

- Crash recovery - that is, after a system failure of some kind has stopped the queue manager unexpectedly
- Restart recovery - after a planned close down of the system

With linear logging, the log is maintained as a continuous sequence of files. When data is collected, it is written sequentially into the log files; the space in the files is not reused, so that you can always retrieve any record from the time that the queue manager was created.

Because disk space is finite, you might have to plan for some form of archiving. Also, if you are handling a high volume of persistent messages, all your log files will eventually be filled. This will result in operator messages being written to an error log file, and some action will need to be taken by the system administrator to make more log space available, or to reuse the existing space. You can use linear logging for:

- Crash recovery
- Restart recovery
- Media recovery - to recreate lost or damaged data after a media failure by replaying the contents of the log

Selecting a logging method

You must base your selection of log type on your requirements for recovery.

Both types of logging can cope with unexpected power outages in the absence of hardware failure. If you accept that only crash or restart recovery is required, circular logging might be adequate. If media recovery is important to you, select linear logging.

With each type of logging, you will need to decide on the number of files to use in the log, and their size. The total amount of space needed will depend on the amount of data to be recorded, which depends on various parameters, including:

- The size of messages
- The number of gets and puts to queues
- The number of messages being transmitted by the message channel agents

Resource management

MQSeries supports the coordination of transactions by an external transaction manager that uses the X/Open XA interface. MQSeries can also act as a transaction manager, coordinating updates made by external resource managers.

MQSeries as a resource manager

MQSeries supports the coordination of transactions by an external transaction manager that uses the X/Open XA interface.

In an XA configuration, the MQSeries queue manager acts as an XA resource manager, managing message queues. The XA transaction manager coordinates the operations of the queue manager, and any other resource managers, to synchronize the commit or backout of transactions. This ensures that updates to MQSeries message queues are coordinated with the updates to all the other types of resource being managed.

CICS for Windows NT operates as an XA transaction manager, so the XA resource management capabilities of MQSeries for Windows NT can be used. MQSeries for Windows NT can also use ENCINA or TUXEDO as a transaction manager.

CICS for OS/2 Warp does not operate as an XA transaction manager, so the XA resource management capabilities of MQSeries for OS/2 Warp and cannot be used. However, CICS does provide a single-phase commit process (as opposed to the two-phase XA coordination), which MQSeries takes part in.

This support is available only on the MQSeries server, and is not available to client applications. See the information on CICS transactions in the *MQSeries System Administration* manual for more details.

MQSeries as a transaction manager

MQSeries can act as a transaction manager and coordinate updates made by external resource managers within MQSeries units of work. These external resource managers must comply to the X/Open XA interface. MQSeries can act as a transaction manager for DB2.

Recovering from problems

MQSeries will recover from both communications failures and power loss incidents. In addition, it is sometimes possible to recover from other types of problem with the MQSeries data, such as inadvertent deletion of a file.

In the case of a communications failure, messages simply remain on queues until they are removed from the queues by a receiving application. If the message is being transmitted, it remains on the transmission queue until it can be successfully transmitted. To recover from a communications failure, it will normally be sufficient simply to restart the channels using the link that failed.

On a restart after your system has lost power, the queue manager will restore all the persistent messages on the queues to the state that existed just before the power failure, so that no persistent messages are lost; nonpersistent messages are discarded.

There are ways in which an MQSeries object can become unusable, for example due to inadvertent damage. In such situations, you will have to take steps to recover either your complete system or some part of it. The action required depends on when the damage is detected, whether the log method selected supports media recovery, and which object or objects are damaged.

Chapter 29. Security planning for MQSeries for OS/2 Warp and Windows NT

In general, each MQSeries product provides security facilities by building on those provided by the platform for which the product was designed.

This chapter describes the security facilities provided by MQSeries. The chapter contains these sections:

- "Setting user IDs with MQSeries for OS/2 Warp"
- "Security exits"

Setting user IDs with MQSeries for OS/2 Warp

Note: This is not applicable to MQSeries for Windows NT.

MQSeries for OS/2 Warp provides the *User ID* installable component. This can be used to generate a user ID value, which the queue manager places in the message context portion of the message descriptor of all messages that are generated by application programs running on the MQSeries for OS/2 Warp client or server.

This user ID can be used by the programs that receive the messages, to verify that the messages have come from an authorized user.

Details of the User ID installable component can be found in the *MQSeries Programmable System Management* book.

Security exits

The message channels that are used for distributed queuing, and the MQI channels that are used between clients and servers, both have security exit facilities that can invoke programs you have supplied.

For more information on these security exit programs, see the *MQSeries Intercommunication* manual.

Chapter 30. Administration of MQSeries for OS/2 and Windows NT

This chapter is a summary of the administration facilities provided by MQSeries. It has the following sections:

Details of the commands, command interfaces, and utilities that are provided by MQSeries are given in the *MQSeries System Administration* manual.

Managing objects

It is the administrator's job to monitor MQSeries and make any changes that might be necessary. To do this, the administrator needs to know where each MQSeries object resides, what its characteristics are, and who has access to it.

The administrator can manage and monitor the resources using MQSeries commands (MQSC), or, if there are sets of commands that are issued regularly, by writing an application program that places them on the command queue.

Commands

MQSeries supports the following administration commands and facilities:

- MQSeries provides control commands that you can enter through the OS/2 Warp or Windows NT command line
- You can use the **runmqsc** control command to cause MQSC commands from standard input to be executed
- Any local or remote MQSeries application program can generate PCF commands in messages, and put them to the command queue, SYSTEM.ADMIN.COMMAND.QUEUE, to be processed by the MQSeries command server

More information on how to use all these facilities is given in the *MQSeries System Administration* manual.

Changes made to the resource definitions of a queue manager using the commands (directly or indirectly) are preserved across restarts of the MQSeries system.

Managing communications

Part of the administrator's role is to ensure that the required communications links are activated, and to monitor the status of these links as required by your enterprise. You can find information describing these tasks in the *MQSeries Intercommunication* manual.

Remote administration

There are two aspects to the MQSeries remote administration facilities:

- MQSeries for OS/2 Warp and MQSeries for Windows NT can be used to manage remote systems
- Other remote products can be used to manage an MQSeries for OS/2 Warp or MQSeries for Windows NT system

Managing remote systems

Facilities are provided by MQSeries to allow an administrator to manage the following remote systems:

- MQSeries for AS/400
- MQSeries for Digital OpenVMS
- MQSeries for MVS/ESA
- MQSeries for OS/2 Warp
- MQSeries for Tandem NSK
- MQSeries on UNIX systems
- MQSeries for Windows NT

Because of the differences between the MQSeries products, it is not always possible to manage remotely the same set of MQSeries objects or attributes that you can manage locally.

If you wish to manage any other MQSeries product, you can write an application program to send the appropriate commands to the command queue for that product. However, some MQSeries products do not have a command queue, so they cannot accept commands from local or remote application programs.

Managing MQSeries from remote systems

MQSeries for OS/2 Warp and MQSeries for Windows NT can be managed from a remote MQSeries system, by an administrator using the facilities provided by the following products:

- MQSeries for AS/400
- MQSeries for Digital OpenVMS
- MQSeries for MVS/ESA
- MQSeries for OS/2 Warp
- MQSeries for Tandem NSK
- MQSeries on UNIX systems
- MQSeries for Windows NT

Note: You can manage MQSeries for OS/2 Warp and MQSeries for Windows NT from MQSeries for MVS/ESA by writing an application program to send the appropriate PCF commands to the command queue.

Chapter 31. Storage planning for MQSeries for OS/2 Warp and Windows NT

This chapter tells you how to plan the type and amount of storage you require when you include MQSeries in your system. It has the following sections:

- "RAM considerations"
- "Disk space considerations"
- "Capacity planning and performance figures" on page 150

RAM considerations

The processor memory (RAM) is used by MQSeries in the execution of the product modules, and as a paging area for the messages that are being processed. Parts of the paging area are written to, and read from, disk as necessary.

The minimum amount of RAM required to run the MQSeries server on OS/2 or Windows NT is 24 MB; if more RAM is available, the performance of the message processing improves. Allow 0.5 MB for each MQSeries client connected.

The amount of RAM required on each client system for an MQSeries client is small compared to that required for the operating system on each of the platforms.

Disk space considerations

Disk space is used by MQSeries for the following:

- Product modules - client and server executable modules, the toolkit, and the online documentation
- Paging space - server only
- Message queues - server only
- Logs - server only

Product modules

The disk space that you require for the product modules depends on the options that you decide to install: the options are described in the *MQSeries System Administration* manual.

Space might be required for client and server executable program modules, the toolkit, and the online documentation. If all options are selected, 66.5 MB is required.

The product modules can be stored on a LAN server, as an alternative to them being stored on a disk attached to the MQSeries client or server. MQSeries loads the modules from the LAN server when required.

Message queues

In order to estimate the total amount of storage that you will need for queues, you need to know:

- The number of queues that you have.
- The maximum number of messages there will be on each of the queues at any one time.
- The average size of message on each of the queues. The amount of storage required for one message varies. It is based on the size of the message data plus the size of the message header (456 bytes), rounded up the nearest 512-byte block. If you are using distribution lists, or grouped or segmented messages, the size of the header increases for the transmission queue.

Given these values, you can calculate the total amount of space required for queues. However, this value is likely to be an approximate value only, and it is advisable to add a contingency value, to avoid the situation where there is no space left for messages on the queues when your application is running.

Log files

Significant events and data changes can be logged in circular or sequential logs. In particular, the logs are used for recording persistent messages.

All the log files in a log are of the same size. By default, this size is 4 MB, but this value can be changed by the system administrator when the log is defined.

For a circular log, the system administrator needs to specify how many files should be included in the log. For a sequential log, the number of files will increase over time, until the system administrator archives some of the files, and disposes of them. You need to plan for the permanent storage (diskettes, tape, or other media supported in your enterprise) that is to be used for these archived files.

Capacity planning and performance figures

Information about MQSeries performance and capacity planning is available on the Internet at:

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

Chapter 32. Introduction to MQSeries for Windows

MQSeries for Windows is a lightweight messaging and queuing product that provides MQSeries functions on workstations that run on the Microsoft Windows platform. It uses significantly fewer resources than other MQSeries products, so it is a good choice to use as a single-user queue manager running on a small or medium-sized personal computer.

- MQSeries for Windows Version 2.0 is a 16-bit product that runs on Microsoft Windows 3.1 and Windows 95.
- MQSeries for Windows Version 2.1 is a 32-bit product that runs on Windows 95 and Windows NT Version 4.0.

For information about these products, see:

- *MQSeries for Windows Version 2.0 User's Guide*
- *MQSeries for Windows Version 2.1 User's Guide*

Do not confuse MQSeries for Windows with MQSeries for Windows NT Version 5. This product is described in Chapter 27, "Introduction to MQSeries for OS/2 Warp and Windows NT" on page 135.

MQSeries for Windows is particularly well suited to users of messaging applications who want to use a standard configuration. It uses definition files that automatically create and start the messaging components the users need, and it can automatically start components when the users start their workstations. These features reduce the need for users of applications to be aware of the messaging product and allow them to concentrate on the applications they want to use.

This chapter contains the following sections:

Where to use MQSeries for Windows

MQSeries for Windows is designed for use as a *leaf node* in a network of queue managers; that is, it is intended for use by a single user on a workstation that is connected to only one other computer in an MQSeries network of computers (see Figure 13 on page 153).

There are important differences between a leaf-node queue manager, an MQSeries client, and a server-node queue manager:

- A leaf-node queue manager is a lightweight product that connects to a network of one or more larger servers. It manages its own queues, so an application that runs on a leaf-node queue manager can continue to work, even if there is a failure in the messaging network or if the user decides to work in standalone (disconnected) mode (for example, away from their own office or in a branch office that does not currently have a connection to a server).

  A leaf-node queue manager is ***not*** intended for use as an intermediate queue manager that passes messages from one queue manager to another or one that serves many users. For this reason it does not support MQSeries clients. It ***is*** intended for a single user working on their own workstation.
- An MQSeries client provides no queue manager functions and it has no queues. It is dependent on an MQSeries server (of the type that supports MQSeries clients). The server owns the queues that the client uses, so if the communication link between the client and the server is broken, the client cannot use those queues.
- A server-node queue manager is a product (such as MQSeries for Windows NT) that manages the queues and communication channels required to support the transfer of messages between queue managers. The computer on which the server-node queue manager runs is large enough to manage the volume of messages such a server might be required to process, and it might also support MQSeries clients. Such a queue manager is likely to be used by a network administrator.

MQSeries for Windows typically runs on workstations that are not powerful enough to act as a server. Like a server though, MQSeries for Windows manages its own queues and the channels to communicate with other queue managers. However, because it is intended to be a leaf node, MQSeries for Windows does not provide all the server functions available on other MQSeries queue managers; these include media recovery, two-phase commit, instrumentation events, and MQSeries client support. For a full list of the MQSeries features that MQSeries for Windows does not support, see “How MQSeries for Windows differs from the other MQSeries products” on page 155.

MQSeries for Windows is designed to run in the Windows environment, so it provides Windows programs that help to make the queue manager easier to use. These programs are not provided by other MQSeries products.

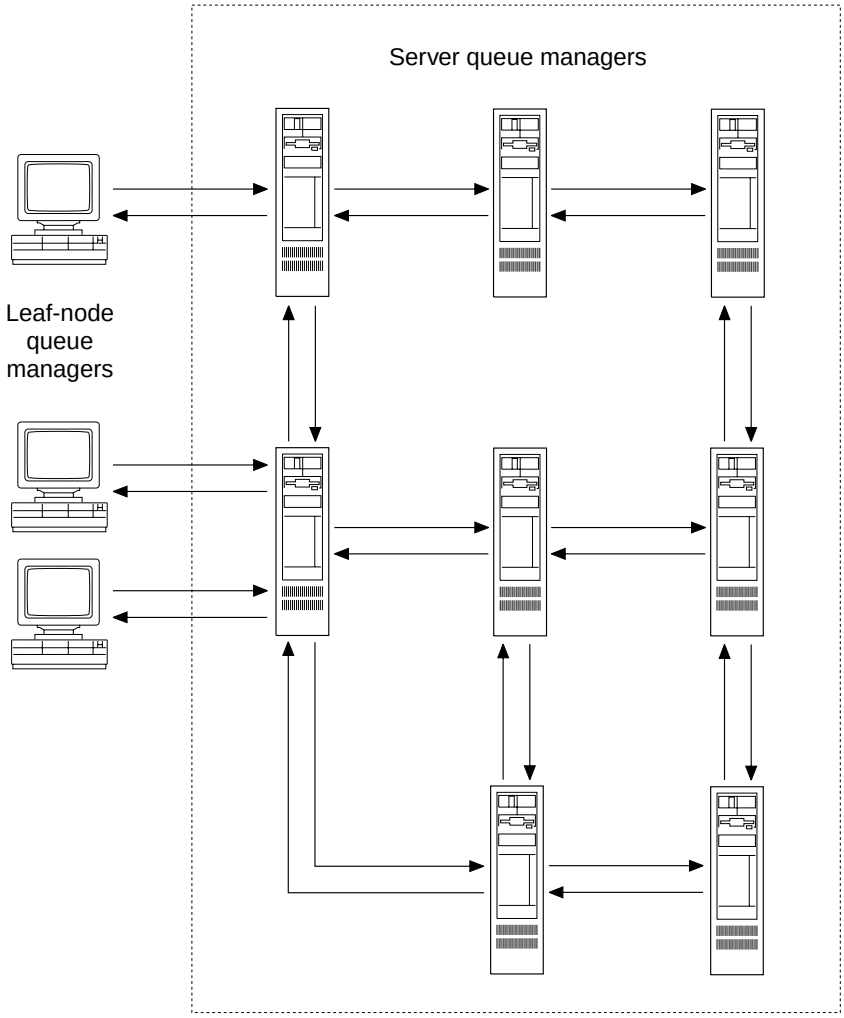

Figure 13. A network of server queue managers and three leaf-node queue managers. The leaf node queue managers run on Windows; they each connect to only one server. The server queue managers run on any other MQSeries platform; they can each have many connections.

The features of MQSeries for Windows

MQSeries for Windows provides existing MQSeries features, but on the Windows operating system:

- A small footprint queue manager that runs on Windows
- The MQSeries Message Queue Interface (MQI) for application development on Windows
- Application development support for the C and Visual Basic programming languages
- Communication between queue managers using TCP/IP
- Standard MQSeries message types and formats
- *Persistent messages* (which survive restarts of the workstation) and nonpersistent messages
- Standard MQSC commands to create, alter, or delete MQSeries objects (but MQSeries for Windows does not support all the commands)
- Enablement for automatic installation.
- Report generation, including confirm on arrival (COA), confirm on delivery (COD), and message expiry
- Remote administration using PCF commands and MQ events (Version 2.1 only)

In addition, MQSeries for Windows provides these features:

- To help users of applications to get started quickly and easily the first time they use the product, MQSeries for Windows uses definition files that automatically create and start the messaging components the users need.
- To help users of applications to get started when they start their workstations, MQSeries for Windows can automatically start its components.
- To help you to set up and work with your MQ components:
  - Version 2.0 of MQSeries for Windows provides utilities you can access from the MQSeries for Windows program group in the Windows Program Manager.
  - Version 2.1 of MQSeries for Windows provides an MQSeries Properties dialog box. You can open this from the Windows taskbar and Control Panel.

  These facilities are supported by extensive online help.
- To make it easier to work with the message channels that you must use to send messages between queue managers, MQSeries for Windows provides channel groups. A channel group is a collection of channels that you start and stop at the same time.
- To make it easier to work with dial-up devices (such as modems) when you connect two queue managers:
  - Version 2.0 of MQSeries for Windows provides transport links which can help you to control the duration (and hence the cost) of such a connection.
  - Version 2.1 of MQSeries for Windows uses the dial-up networking connections provided by the operating system.

Comparing queue managers, clients, and servers

If you already use MQSeries clients, see Table 14 for a summary of the differences between an MQSeries for Windows queue manager, an MQSeries client, and an MQSeries server.

Table 14. Comparison of supported features on MQSeries for Windows

Feature	MQSeries for Windows	MQSeries client on Windows	MQSeries for OS/2 Warp
Independent operation	Yes	No	Yes
Queue manager	Yes	No	Yes
Queues	Yes	No	Yes
Message channels	Yes	No	Yes
Run MQSC commands	Utility (V2.0), interactive (V2.1), or command file	No	Command line or command file
Persistence of MQSeries objects	Yes	All objects are on the server	Yes
Logging and media recovery	No	All objects are on the server	Yes
Automatic installation	Yes	Yes	Yes
Automatic start up	Yes	No	No
Supports MQSeries clients	No	Not applicable	Yes

How MQSeries for Windows differs from the other MQSeries products

MQSeries for Windows is a leaf-node queue manager for the Microsoft Windows platform. It is designed to minimize system requirements so that workstations with relatively modest specifications can use commercial messaging. This section summarizes the differences between MQSeries for Windows and the other workstation products in the MQSeries family. The features are listed in alphabetic order.

Attributes of queues and queue managers

MQSeries for Windows does not support all the attributes of queues and queue managers (for example, it does not support those related to triggering). If you use an unsupported attribute in a command or an MQI call, MQSeries for Windows returns a value to show that the attribute is not supported.

Authority checking on the MQOPEN call

MQSeries for Windows does not support the SETMQAUT and DSPMQAUT commands.

Command Server

Version 2.0 of MQSeries for Windows does not support the MQSeries Command Server, so it does not support any MQSeries feature that

uses the command server (for example, PCF commands and remote administration).

Version 2.1 provides a command server.

Context passing
MQSeries for Windows does not copy context information from messages it receives from other queue managers. This is because MQSeries for Windows is intended to be a leaf node; it is not intended to be an intermediate node in a network of queue managers, where messages received from one queue manager are passed on to another.

Control commands
In other MQSeries products, you can issue control commands from the command line. MQSeries for Windows provides a user interface to enable you to perform the functions of some of these commands (for example, starting and stopping a queue manager). For a comparison with the MQSeries control commands, see the *MQSeries for Windows User's Guide*.

Data conversion
When an MQSeries for Windows queue manager receives data from a queue manager running on a different platform, it cannot convert the machine encoding, integer representation, or coded character set of the application data. Also, it cannot run data conversion exits. This means that any data conversion that is required must be performed by the other queue manager.

Dead-letter queues
MQSeries for Windows does not support dead-letter queues. A dead-letter queue is a queue to which a queue manager or application sends messages it cannot deliver to their correct destination. It is also known as an undelivered-message queue.

An MQSeries for Windows queue manager does not need a dead-letter queue because, being a leaf node, it is always on the edge of a network of queue managers. This means that it does not have to store messages for onward transmission to other queue managers.

Distributed Computing Environment (DCE) directories
MQSeries for Windows does not support DCE directories.

Distributed Computing Environment (DCE) security
MQSeries for Windows does not support exits relating to DCE security.

Events See *instrumentation events*.

Installable services
MQSeries for Windows does not support MQSeries installable services. These are additional functions provided in other MQSeries products as several independent components.

Instrumentation events
Instrumentation events are facilities you can use to monitor the operation of queue managers and channels in a network of MQSeries systems. Version 2.1 of MQSeries for Windows generates most of the MQSeries instrumentation events. Version 2.0 does not generate instrumentation events.

Media recovery and logging

MQSeries for Windows does not support the creation of a sequence of log records that contain an image of an object. Other MQSeries products allow you to create such records and re-create objects from this image.

Message Queue Interface (MQI)

MQSeries for Windows supports a subset of the MQI.

To understand those features of the MQI that MQSeries for Windows does not support, see the *MQSeries for Windows User's Guide*.

Message retry exit

MQSeries for Windows does not support a message retry exit.

An MQSeries for Windows queue manager does not need a message retry exit because, being a leaf node, it is always on the edge of a network of queue managers. This means that it does not have to store messages for onward transmission to other queue managers.

MQI channels

MQSeries for Windows does not support MQI channels. These are client connection and server connection channels. These are used with MQSeries clients only, so MQSeries for Windows does not support them.

MQSC commands

MQSeries for Windows supports a subset of the MQSC commands. To see which commands it supports, see the *MQSeries for Windows User's Guide*.

However, MQSeries for Windows provides facilities that allow you to type MQSC commands in a window (and test and reissue them) and run MQSC command files. This is described in the *MQSeries for Windows User's Guide*.

MQSeries client and server support

You cannot use an MQSeries for Windows queue manager as an MQSeries client, nor can you use it to support its own MQSeries clients.

Network support

MQSeries for Windows supports TCP/IP only.

Object Authority Manager (OAM)

MQSeries for Windows does not provide a security manager. It does not support the SETMQAUT and DSPMQAUT commands.

Process definitions

Other MQSeries products use process definitions for setting up the automatic triggering of applications. MQSeries for Windows does not support triggering or process definitions.

Programmable Command Formats (PCFs)

Version 2.0 of MQSeries for Windows does not support PCFs.

Version 2.1 supports many of the PCF commands. To see which commands, see the *MQSeries for Windows User's Guide*.

Queue manager

MQSeries for Windows supports multiple queue manager definitions, but it allows only one queue manager to run at any time.

Queue manager quiescing

MQSeries for Windows does not support the quiescing of a queue manager. This is the ability to allow applications to finish processing before the queue manager is stopped, and to prevent any further applications starting.

Sample programs

MQSeries for Windows provides Windows versions of some of the MQSeries sample programs. The MQSeries for Windows samples are described in the *MQSeries for Windows User's Guide*.

Security manager

See *object authority manager*.

Signaling

Version 2.0 of MQSeries for Windows does not support signaling.

Version 2.1 supports signalling, so you can use the MQGMO_SET_SIGNAL option with the MQGET call on Windows applications.

Triggering

MQSeries for Windows does not support triggering, so it does not allow a queue manager to start an application automatically when predetermined conditions on a queue are satisfied. The following features of triggering are also not supported:

- Initiation queues
- Process definitions
- Trigger monitors

Two-phase commit

MQSeries for Windows does not support two-phase commit. This is a protocol for the coordination of changes to recoverable resources when more than one resource manager is used by a single transaction.

However, MQSeries for Windows does allow the queue manager to commit or back out units of work.

Part 6. Planning for MQSeries for Tandem NSK

Chapter 33. Introduction to MQSeries on Tandem NSK

MQSeries for Tandem NSK runs on a machine that is capable of running the Tandem NSK operating system.

MQSeries provides the MQI programming interface for use by application programs that are running on Tandem NSK.

For information on the hardware and software environments, see “MQSeries for Tandem NSK” on page 239.

Planning for MQSeries

This chapter helps you to plan for the introduction of MQSeries for Tandem NSK into your enterprise, and introduces the items that you need to consider when doing this planning.

There are several stages in planning for the use of MQSeries for Tandem NSK that you must go through. They are:

1. Preparing your applications for the use of MQSeries for Tandem NSK
2. Planning to include MQSeries for Tandem NSK in a network
3. Preparing to install MQSeries for Tandem NSK
4. Planning to set up MQSeries for Tandem NSK

A prime requirement for a message delivery system is that it must be reliable. Functions are built into MQSeries for Tandem NSK to ensure that:

- Messages are not lost despite system failures
- Messages are not delivered more than once
- Messages are not accessed by, or delivered to, unauthorized persons or applications

MQSeries uses Tandem NSK facilities to support these functions.

You must also plan for the operation and administration of MQSeries for Tandem NSK in your enterprise, and consider the implementation of an appropriate set of security facilities. Brief outlines of these planning operations are included in this chapter.

Preparing your applications for use with MQSeries

MQSeries for Tandem NSK brings the Message Queue Interface (MQI) to your applications. This interface allows you to modify existing applications and to write new applications. The MQI removes much of the need to understand the network and communication systems that you use. Thus you can expect to generate applications more speedily than before. However, you must prepare to take advantage of the MQI by planning its use in your applications and by understanding the ways in which it assists you.

You can find more information on how to use the MQI in your applications by referring to the *MQSeries Application Programming Guide*.

Planning to use MQSeries in a network

MQSeries for Tandem NSK uses distributed queuing to exchange messages between MQSeries platforms, using either the SNA LU 6.2, or TCP/IP transmission protocols.

You must consider how you will attach MQSeries for Tandem NSK to a network, and how you will define the message channels that will be used to exchange messages.

According to the way that you have set your systems up, security checks can be performed at various times. Various exits are provided that can be used by your applications to provide these facilities.

"MQSeries interoperability summary" on page 210 shows the links that are possible to other MQSeries products, and the transmission protocols that can be used.

For further information about distributed queuing, refer to the *MQSeries Intercommunication* manual.

Installing MQSeries

MQSeries for Tandem NSK is installed with the **instmqm** control command. For information about installing MQSeries, see the *MQSeries for Tandem NonStop Kernel System Management Guide*.

Setting up MQSeries

After installation, MQSeries for Tandem NSK needs to be set up, and customized for your own use. This ensures that the appropriate Tandem NSK facilities are made available to MQSeries, and that your MQSeries system is correctly initialized and ready to work with your applications.

You need to do the following:

- Define queues
  - Consider your naming conventions for queues
- Define trigger processes
- Define remote links
  - Define associated transmission queues
  - Consider your naming conventions for remote queues
  - Consider your naming conventions for channels

Note: The characters within the names given to all MQSeries objects are case sensitive. Therefore, be very careful when defining the names of objects, to select the appropriate uppercase or lowercase characters.

You can find more information on the setting up and customizing processes for MQSeries for Tandem NSK in the *MQSeries for Tandem NonStop Kernel System Management Guide*.

Planning recovery services

MQSeries for Tandem NSK provides facilities to allow backup and recovery of the messaging system. Chapter 34, “Backup and recovery planning for MQSeries for Tandem NSK” on page 165 introduces you to these facilities, and to the items that you need to consider in order to include MQSeries for Tandem NSK in your backup and recovery plans.

You can find more information on the backup and recovery facilities provided by MQSeries for Tandem NSK in the *MQSeries for Tandem NonStop Kernel System Management Guide*.

Planning data security

MQSeries for Tandem NSK uses the facilities of the MQSeries *object authority manager (OAM)* installable component to control access to the various different types of queue manager resource (queues, process definitions, channels, and queue managers).

You can find more general information on authorization installable components in Chapter 8, “Introduction to the MQSeries Framework” on page 53.

Chapter 35, “Security planning for MQSeries for Tandem NSK” on page 167 introduces you to the security facilities provided by MQSeries for Tandem NSK and to some of the items you need to consider when planning for security.

You can find more information on the security facilities provided by MQSeries for Tandem NSK in the *MQSeries for Tandem NonStop Kernel System Management Guide*.

Administration

Information about administering MQSeries is given in “MQSeries product administration facilities” on page 36.

A summary of the administration facilities provided for Tandem NSK is given in Chapter 36, “Administration of MQSeries for Tandem NSK” on page 171. Full details of these facilities can be found in the *MQSeries for Tandem NonStop Kernel System Management Guide*.

Migration from MQSeries for Tandem NSK Version 1.5.1

If you are a user of MQSeries for Tandem NSK V1.5.1, you can convert your existing MQSeries configuration files and messages to work with MQSeries for Tandem NSK V2.2 using the following two conversion utilities:

CNV1520 Converts MQSeries for Tandem NSK Version 1.5.1 queue and channel definitions into MQSC scripts.

CNVMSGS Transfers messages from MQSeries for Tandem NSK Version 1.5.1 message queues to Version 2.2 message queues after the queue definitions have been established using CNV1520.

Both utilities reside in the ZMQSEXE subvolume.

Migrating applications

To migrate your MQSeries for Tandem NSK V1.5.1 applications you must recompile and rebind them with V2.2 header files and libraries. Stubs have been provided for MQI calls that are not present or required in MQSeries for Tandem NSK V2.2, so code changes relating to MQSeries (other than including the correct header files) are not required. However, MQSeries for Tandem NSK V2.2 requires that you compile C programs with the WIDE model. MQSeries for Tandem NSK V1.5.1 required LARGE; if your programs contain code that relies on LARGE data representations, the code might have to be changed before it functions correctly under the WIDE model.

Chapter 34. Backup and recovery planning for MQSeries for Tandem NSK

This chapter describes the background concepts of recovery and restart. More information about the recovery facilities of MQSeries for Tandem NSK is given in the *MQSeries for Tandem NonStop Kernel System Management Guide*.

Recovery facilities

MQSeries for Tandem NSK ensures that messages are not lost by using the Tandem NonStop Transaction Manager (TM/MP). TM/MP provides transaction protection, queue-file consistency, and queue-file recovery.

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; nonpersistent messages are lost during the process.

Recovering from problems

If you properly configure MQSeries and your NSK system software and hardware, the failure of any single hardware of software component does not result in the loss of any data or system functions. MQSeries can recover from a single point of failure while maintaining data integrity.

For more information about recovery from failures, see the *MQSeries for Tandem NonStop Kernel System Management Guide*.

Backing up and restoring

Periodically, you might want to make a backup of your queue manager data to provide protection against possible corruption due to hardware failures. See the *MQSeries for Tandem NonStop Kernel System Management Guide* for information about backing up and restoring MQSeries for Tandem NSK.

Chapter 35. Security planning for MQSeries for Tandem NSK

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, channels, and processes
- Commands for queue manager administration, including MQSC 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.

This chapter describes the access control security features in MQSeries for Tandem NSK. It contains these sections:

Full details of MQSeries for Tandem NSK security handling are given in the *MQSeries for Tandem NonStop Kernel System Management Guide*.

Controlling access to resources

By default, access to queue manager resources is controlled through an authorization service installable component. The authorization service component supplied with MQSeries for Tandem NSK is called the OAM and is automatically installed and enabled for each queue manager you create, unless you specify otherwise.

The OAM is an *installable component* of the authorization service. Because the OAM is an installable component, you can implement your own security controls in place of, or in addition to, those supplied by the OAM. (For more information on installable services and installable components, see Chapter 8, "Introduction to the MQSeries Framework" on page 53.)

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.

The OAM uses the user and group IDs and security features of the Tandem 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 NSK *groups*. The OAM maintains authorizations at the group level.

Using groups, rather than individual users, for authorization reduces the amount of administration required. Typically, a particular kind of access is required by more than one user. 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 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.

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.

Resources you can protect

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 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 NSK MQM group. This group should have been set up before you installed the product.

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 two commands that you can invoke from TACL to manage the authorizations of users. These are:

- **setmqaut** (Set or reset authority)
- **dspmqaut** (Display authority)

Details of these commands can be found in the *MQSeries for Tandem NonStop Kernel System Management Guide*.

Security exits

The message channels that are used for distributed queuing, and the MQI channels that are used between clients and servers, both have security exit facilities that can invoke programs that you have supplied.

For more information on these security exit programs, see the *MQSeries Intercommunication* manual.

Chapter 36. Administration of MQSeries for Tandem NSK

This chapter is a summary of the administration facilities provided by MQSeries for Tandem NSK. It has the following sections:

Details of the commands, command interfaces, and utilities that are provided by MQSeries for Tandem NSK are given in the *MQSeries for Tandem NonStop Kernel System Management Guide*. You need to arrange for users who need to be able to use these administration facilities to have the necessary authorizations, using the procedures given in the *System Management Guide*.

Managing objects

It is the administrator's job to monitor MQSeries for Tandem NSK and make any changes that might be necessary. To do this, the administrator needs to know where each MQSeries object resides, what its characteristics are, and who has access to it.

The administrator can manage and monitor the resources using MQSeries commands (MQSC), or, if there are sets of commands that are issued regularly, by writing an application program that places them on the command queue.

MQSeries can use the security features provided by the OAM, or by a security component that you have installed, to ensure that the user is authorized to issue particular commands for particular resources.

Commands

MQSeries for Tandem NSK supports the following administration commands and facilities:

- You can enter control commands on the command line.
- You can use the **runmqsc** control command to cause MQSC commands from standard input to be executed.
- Any local or remote MQSeries application program can generate PCF commands in messages and put them to the command queue SYSTEM.ADMIN.COMMAND.QUEUE, to be processed by the MQSeries for Tandem NSK command server.

In addition:

- Some TS/MP (Pathway) commands are used for administration purposes.
- The MQM (Message Queue Management) facility supports some administration tasks.

More information on how to use all these facilities is given in the *MQSeries for Tandem NonStop Kernel System Management Guide*.

Changes made to the resource definitions of a queue manager using the commands (directly or indirectly) are preserved across restarts of MQSeries for Tandem NSK.

Managing communications

Part of the administrator's role is to ensure that the required communications links are activated, and to monitor the status of these links as required by your enterprise. You can find information describing these tasks in the *MQSeries Intercommunication* manual.

Remote administration

There are two aspects to the MQSeries remote administration facilities:

- MQSeries for Tandem NSK can be used to manage remote systems
- Other remote products can be used to manage MQSeries for Tandem NSK

Managing remote systems

Facilities are provided by MQSeries for Tandem NSK to allow an administrator to manage the following remote systems:

- MQSeries for AS/400
- MQSeries for Digital OpenVMS
- MQSeries for MVS/ESA
- MQSeries for OS/2 Warp
- MQSeries for Tandem NSK
- MQSeries on UNIX systems
- MQSeries for Windows NT

Because of the differences in the MQSeries products, it is not always possible to manage remotely the same set of MQSeries objects or attributes that you can manage locally.

If you want to manage any other MQSeries product, you can write an application program to send the appropriate commands to the command queue for that product. However, some MQSeries products do not have a command queue, so they cannot accept commands from local or remote application programs.

Managing MQSeries from remote systems

MQSeries for Tandem NSK can be managed from a remote MQSeries system, by an administrator using the facilities provided by the following products:

- MQSeries for AS/400
- MQSeries for Digital OpenVMS
- MQSeries for MVS/ESA
- MQSeries for OS/2 Warp
- MQSeries for Tandem NSK
- MQSeries on UNIX systems
- MQSeries for Windows NT

Note: You can manage MQSeries for Tandem NSK from MQSeries for MVS/ESA by writing an application program to send the appropriate PCF commands to the command queue.

Part 7. Planning for MQSeries on UNIX systems

Chapter 37. Introduction to MQSeries on UNIX systems

MQSeries on UNIX systems

Don't forget: MQSeries on UNIX systems is the general term used in this book for UNIX operating systems that support MQSeries Level 2 products (for example: AIX, Sun Solaris, and HP-UX).

MQSeries products that run on UNIX operating systems comprise two parts, the *server* and the *clients*. The server runs on a machine that is capable of running an MQSeries queue manager on a UNIX system; the clients provided with the product are for OS/2 Warp, Windows 3.1, DOS, Windows 95, and your UNIX platform. On AIX, HP-UX, and Sun Solaris, clients are also provided for AIX, HP-UX, Sun Solaris, and Windows NT.

A variety of terminals (such as X-stations, non-programmable terminals, and portables) can access the server code. There is no MQSeries code on such terminals.

For more information on the hardware and software environments, see:

- "MQSeries for AIX" on page 216.
- "MQSeries for AT&T GIS UNIX" on page 219.
- "MQSeries for HP-UX" on page 222.
- "MQSeries for SINIX and DC/OSx" on page 233.
- "MQSeries for SunOS" on page 235.
- "MQSeries for Sun Solaris" on page 237.

In this section, you are often referred to other books for more information; the appropriate books are:

Table 15. MQSeries on UNIX systems: system administration manuals

UNIX platform	See these manuals
AIX	*MQSeries for AIX V5.0 Quick Beginnings*, *MQSeries System Administration*
AT&T GIS UNIX	*MQSeries for AT&T GIS UNIX System Management Guide*
HP-UX	*MQSeries for HP-UX V5.0 Quick Beginnings*, *MQSeries System Administration*
SINIX and DC/OSx	*MQSeries for SINIX and DC/OSx System Management Guide*
SunOS	*MQSeries for SunOS System Management Guide*
Sun Solaris	*MQSeries for Sun Solaris V5.0 Quick Beginnings*, *MQSeries System Administration*

Planning for MQSeries

This chapter helps you to plan for the introduction of MQSeries on UNIX systems into your enterprise, and introduces the items that you need to consider when doing this planning.

There are several stages in planning for the use of MQSeries on UNIX systems that you must go through. They are:

1. Preparing your applications for the use of MQSeries on UNIX systems
2. Planning to include MQSeries on UNIX systems in a network
3. Preparing to install MQSeries on UNIX systems
4. Planning to set up MQSeries on UNIX systems

A prime requirement for a message delivery system is that it must be reliable. Many functions are built into MQSeries on UNIX systems to ensure that:

- Messages are not lost despite system failures
- Messages are not delivered more than once
- Messages are not accessed by, or delivered to, unauthorized persons or applications

MQSeries on UNIX systems uses logging and other facilities to support these functions.

You must also plan for the operation and administration of MQSeries on UNIX systems in your enterprise, and consider the implementation of an appropriate set of security facilities. Brief outlines of these planning operations are included in this chapter.

Preparing your applications

MQSeries on UNIX systems brings the Message Queue Interface (MQI) to your applications. This interface allows you to modify existing applications and to write new applications. The MQI removes much of the need to understand the network and communication systems that you use. Thus you can expect to generate applications more speedily than before. However, you must prepare to take advantage of the MQI by planning its use in your applications and by understanding the ways in which it assists you.

You can find more information on how to use the MQI in your applications by referring to the *MQSeries Application Programming Guide*.

MQSeries on SunOS, DC/OSx and AT&T GIS UNIX does not support threads. MQSeries on SINIX, AIX, Sun Solaris, and HP-UX supports multithreaded applications, but there are limitations that are documented in the *MQSeries Application Programming Guide*. If the platform that you intend to use is not specifically mentioned, check with your IBM support center whether multithreaded applications are supported.

Interfacing with CICS

With MQSeries products on UNIX systems, you can create application programs for the appropriate CICS transaction environment. These applications can use the MQI to communicate with CICS or non-CICS programs in any of the environments supported by the MQSeries products.

Planning to use MQSeries in a network

MQSeries on UNIX systems uses the distributed queue management (DQM) facility to exchange messages between MQSeries platforms, using either the SNA LU 6.2 or TCP/IP transmission protocols.

You must consider how you will attach MQSeries on UNIX systems to a network, and how you will define the message channels that will be used to exchange messages.

According to the way that you have set your systems up, security checks can be performed at various times. MQSeries products on UNIX systems do not provide communications link authorization or data encryption on these links. Instead, various exits are provided that can be used by your applications to provide these facilities. On AIX, HP-UX, and Sun Solaris, MQSeries also provides exits relating to DCE security.

"MQSeries interoperability summary" on page 210 shows the links that are possible to other MQSeries products, and the transmission protocols that can be used.

For further information about distributed queue management, refer to the *MQSeries Intercommunication* manual.

Installing MQSeries

Before you can install one of the MQSeries products on a UNIX system, you must create both a group and user with the name **mqm**, which will own the directories and files that contain the various resources associated with the product.

To prepare for the actual installation, you need to plan how much disk space will be required in your UNIX system to accommodate MQSeries. Assistance is given in Chapter 41, "Storage planning for MQSeries on UNIX systems" on page 191 to help you plan the amount of space required.

You can find more information to help you with the installation of MQSeries on a UNIX system in the appropriate manual (see Table 15 on page 175).

When you have installed MQSeries on AIX, HP-UX, or Sun Solaris, you can run the supplied installation verification test. This is described in the appropriate *MQSeries Quick Beginnings* manual.

Setting up MQSeries

After installation, MQSeries needs to be set up, and customized for your own use. This ensures that the appropriate UNIX system facilities are made available to MQSeries, and that your MQSeries system is correctly initialized and ready to work with your applications.

MQSeries uses configuration files to hold the product configuration information used for logging, communications protocols and installable components. After installing

the product, you can edit these files to tailor the operation of the product to meet the requirements of your installation.

In addition, you need to do the following:

- Define queues
  - Consider your naming conventions for queues
- Define trigger processes
- Define remote links
  - Define associated transmission queues
  - Consider your naming conventions for remote queues
  - Consider your naming conventions for channels

Note: The characters within the names given to all MQSeries objects are case sensitive. Therefore, be very careful when defining the names of objects, to select the appropriate uppercase or lowercase characters.

You can find more information about the setting up and customizing processes for MQSeries on UNIX systems in the appropriate manual (see Table 15 on page 175).

Planning recovery services

MQSeries provides logging services to allow backup and recovery of the messaging system. Chapter 38, “Backup and recovery planning for MQSeries on UNIX systems” on page 181 introduces you to these facilities, and to the items that you need to consider in order to include MQSeries on UNIX systems in your backup and recovery plans.

You can find more information on the backup and recovery facilities provided by MQSeries in the appropriate manual (see Table 15 on page 175).

Planning data security

MQSeries uses the facilities of the MQSeries *object authority manager (OAM)* installable component to control access to the various different types of queue manager resource (queues, process definitions, channels, and queue managers).

You can find more general information on authorization installable components in Chapter 8, “Introduction to the MQSeries Framework” on page 53.

Chapter 39, “Security planning for MQSeries on UNIX systems” on page 185 introduces you to the security facilities provided by MQSeries and to some of the items you need to consider when planning for security.

You can find more information on the security facilities provided by MQSeries on UNIX systems in the appropriate manual (see Table 15 on page 175).

Administration

A summary of the administration facilities provided is given in Chapter 40, “Administration of MQSeries on UNIX systems” on page 189. Full details of these facilities can be found in the appropriate manual (see Table 15 on page 175).

Support for Lotus Notes

MQSeries for AIX, MQSeries for HP-UX, and MQSeries for Sun Solaris provide a Lotus Notes server add-in task that gives Lotus Notes applications access to MQSeries messaging. This allows Lotus Notes users to communicate with other systems connected by MQSeries.

Note: This MQSeries application is not available on other MQSeries UNIX platforms. However, some support packs to link to Lotus Notes for other UNIX platforms might be available. Refer to the MQSeries home page for details, see "MQSeries information available on the Internet" on page xvii.

Lotus Notes is a networked application that users can use to share information. Lotus Notes has two main components, the server and the client. The Lotus Notes server provides services to Lotus Notes clients and to other servers. The services provided include storage and replication of shared databases and mail routing. Lotus Notes clients connect to a Lotus Notes server to use shared databases, and also to read and send mail. The Lotus Notes server add-in task runs in either an MQSeries server or an MQSeries client on that Lotus Notes server.

The basic units of information in a Lotus Notes system are databases and the documents that they contain. A database can be used by one person, or shared among users who have common data requirements. Most databases in Lotus Notes reside on a Lotus Notes server.

MQSeries provides a Lotus Notes server add-in task that recognizes and interprets:

- Data from documents that Lotus Notes wants to send to MQSeries
- Messages from MQSeries sent in reply and used to update a Lotus Notes document

The mobile or remote Notes user can access MQSeries applications and data.

Support for R/3

The following products:

- MQSeries link for R/3 for AIX
- MQSeries link for R/3 for HP-UX, and
- MQSeries link for R/3 for Sun Solaris

provide an interface that enables you to integrate your R/3 application with applications running in other environments (including those on R/3 and R/2 environments).

The R/3 link works with the Application Link Enabling (ALE) layer of the R/3 system to transmit Intermediate Documents (IDocs) into and out of your R/3 system, using MQSeries messages and queues to carry the information. It extends the scope of your business by allowing you to link your R/3 applications to any other application that you can access through MQSeries, even when those applications require different data formats.

For more information, see the *MQSeries link for R/3 User's Guide*.

Migration from MQSeries Version 1

To use MQSeries application programs that were written for version 1 of MQSeries with version 2 of MQSeries, you need to do the following:

1. Redefine all message queues.
2. Redefine all message channels.
3. Recompile the application programs, using the MQSeries version 2 header files.

This might be a suitable time to consider whether you need to re-design any parts of your application, to take advantage of the additional function provided by version 2 of MQSeries.

One difference between version 2 of MQSeries on UNIX systems and the earlier products is that version 2 does its own queue storage management. It is not necessary to run a utility program to recover the space that was occupied by messages that have been removed from queues by MQGET calls; this is done automatically by MQSeries.

Migration from MQSeries Version 2

When you have migrated from MQSeries Version 2 to MQSeries Version 5 you will be unable to revert to Version 2. You should back up your system before installing the new version. This will enable you to back off the upgrade if necessary. If you do this however, you will not be able to recover the work performed by MQSeries Version 5.

With MQSeries Version 5, the system default objects are created automatically when you use the crtmqm command to create a queue manager. The sample MQSC definition file, AMQSCOMA.TST, is no longer provided. If you have used AMQSCOMA.TST to customize your settings for MQSeries Version 2, and you want to use the same settings with Version 5, save your version of the file before you install MQSeries Version 5. You can then use this file to create the Version 2 default objects. Alternatively, you can generate a new MQSC definition file if required.

A list of the system default objects for MQSeries Version 5 is provided in the *MQSeries System Administration* manual.

Chapter 38. Backup and recovery planning for MQSeries on UNIX systems

This chapter describes the background concepts of recovery and restart. It contains the following sections:

More details of the logging and recovery facilities of MQSeries on UNIX systems are given in the appropriate manual (see Table 15 on page 175).

Logging

The basic premise of a messaging system is that messages entered into the system are assured of delivery to the destination. One of the ways of ensuring that messages are not lost is to maintain a record of the activities of the queue manager that handles the receipt, transmission, and delivery of messages.

MQSeries on UNIX systems does this by recording all the significant changes to the data controlled or managed by the queue manager in a log. This process is called logging. The data changes that are logged include the puts and gets of persistent messages to and from queues, changes to queue attributes, and channel activity.

The purpose of logging is to create and maintain a log that:

- Keeps records of queue manager changes
- Keeps records of queue updates for use by the restart process
- Is a source for restoration of data should there be a hardware or software failure

Each MQSeries log consists of a log control file, together with one or more log files for the storage of data.

The log control file is, as its name implies, used to control and monitor the use of the log files. It contains information relating to the size, location, next available file, and other data related particularly to the log files themselves. All the log files within one log are the same size; there is a default value for this size, but you can override this value when you set up the log.

On some UNIX platforms, MQSeries provides a log dump facility (DMPMQLOG). This enables you to format and display the contents of the log when doing problem determination.

Types of logging

MQSeries on UNIX systems has two approaches to maintaining records of queue manager activities:

- Circular logging
- Linear logging

Each type of logging stores the recorded data in a set of files. The differences between the two types of logging are the contents, and the way that the files are linked together.

With circular logging, the set of log files are effectively linked together so as to form a ring. When data is collected, it is written sequentially into the files, in such a way as to reuse the log files in the ring. You can use circular logging for:

- Crash recovery - that is, after a system failure of some kind has stopped the queue manager unexpectedly
- Restart recovery - after a planned closedown of the system

With linear logging, the log is maintained as a continuous sequence of files. When data is collected, it is written sequentially into the log files; the space in the files is not reused, so that you can always retrieve any record from the time that the queue manager was created.

Because disk space is finite, you might have to plan for some form of archiving. Also, if you are handling a high volume of persistent messages, all your log files will eventually be filled. This will result in operator messages being written to an error log file; some action will need to be taken by the system administrator to make more log space available, or to reuse the existing space. You can use linear logging for:

- Crash recovery
- Restart recovery
- Media recovery - to recreate lost or damaged data after a media failure by replaying the contents of the log

Selecting a logging method

You must base your selection of log type on your requirements for recovery.

Both types of logging can cope with unexpected power outages in the absence of hardware failure. If you accept that only crash or restart recovery is required, circular logging might be adequate. If media recovery is important to you, select linear logging.

With each type of logging, you need to decide on the number of files to use in the log, and their size. The total amount of space needed depends on the amount of data to be recorded, which depends on various parameters, including:

- The size of messages
- The number of puts and gets from queues
- The number of messages being transmitted by the message channel agents

Resource management

MQSeries on UNIX systems supports the coordination of transactions by an external transaction manager that uses the X/Open XA interface. On some UNIX platforms, MQSeries can also act as a syncpoint coordinator, coordinating updates made by external resource managers.

MQSeries as a resource manager

In an XA configuration, the MQSeries queue manager acts as an XA resource manager, managing message queues. The XA transaction manager coordinates the operations of the queue manager, and any other XA-compliant resource managers, to synchronize the commit or backout of transactions. This ensures that updates to MQSeries message queues are coordinated with the updates to all the other types of resource being managed.

With MQSeries on UNIX systems, the XA transaction manager can be as follows:

CICS AIX, HP-UX, SINIX, and Sun Solaris

ENCINA AIX, HP-UX, Sun Solaris, and SINIX and DC/OSx

TUXEDO AIX, AT&T GIS UNIX[2], HP-UX, SINIX and DC/OSx, SunOS, and Sun Solaris

The MQSeries resources will be committed or backed out as directed by CICS, ENCINA, or TUXEDO. This support is available only on the MQSeries on UNIX systems server, and is not available to client applications.

MQSeries as a transaction manager

On some UNIX platforms, MQSeries can act as a transaction manager and coordinate updates made by external resource managers within MQSeries units of work. These external resource managers must comply to the X/Open XA interface.

MQSeries for AIX, MQSeries for HP-UX, and MQSeries for Sun Solaris can act as transaction managers for the following XA-compliant database managers:

- DB2
- Oracle

[2] This platform has become NCR UNIX SVR4 MP-RAS, R3.0

Recovering from problems

MQSeries can recover from communications failures and power loss incidents. In addition, it is sometimes possible to recover from other types of problem with the MQSeries data, such as inadvertent deletion of a file.

In the case of a communications failure, messages remain on the queues until they are removed by a receiving application. If the message is being transmitted, it remains on the transmission queue until it can be successfully transmitted. To recover from a communications failure, it is normally sufficient simply to restart the channels using the link that failed.

On a restart after your system has lost power, the queue manager restores all the persistent messages that were on the queues to the state that existed just before the power failure, so that no persistent messages are lost; nonpersistent messages are discarded.

There are ways in which an MQSeries object can become unusable, for example due to inadvertent damage. In such situations, you will have to take steps to recover either your complete system or some part of it. The action required depends on when the damage is detected, whether the log method selected supports media recovery, and which object or objects are damaged.

High availability

For those situations where high availability is a requirement, you might wish to consider using MQSeries for AIX with the AIX High Availability Cluster Multi-Processing/6000 (HACMP/6000) product.

HACMP/6000 offers an efficient way to recover from failures at the AIX server that is running your application programs. The product allows for servers to be configured to provide, for example, automatic transfer to a standby processor if the primary one fails.

Also on AIX, you can use *disk mirroring* to improve the availability of your application data on disk.

Chapter 39. Security planning for MQSeries on UNIX systems

This chapter describes the access control security features in MQSeries on UNIX systems. It contains these sections:

Full details of MQSeries on UNIX systems security handling are given in the *MQSeries System Administration* manual.

Controlling access to resources

With MQSeries on UNIX systems, access to queue manager resources is controlled through the *object authority manager (OAM)*, which is the default authorization installable component. Because the OAM is an installable component, you can implement your own security controls in place of, or in addition to, those supplied by the OAM. For more information on installable services and installable components, see Chapter 8, "Introduction to the MQSeries Framework" on page 53.

Users can access queue manager objects (queues, process definitions, channels, and queue managers) only if they have the required authority. The OAM manages a user's authorization to manipulate MQSeries objects, and provides a command interface through which you can grant or revoke access authority to an object for a specific group of users.

Managing access through user groups

In discussing security in a UNIX environment, we use the term *principal* rather than user ID. The reason for this is that authorities granted to a user ID can also be granted to other entities, for example, an application program that issues MQI calls, or an administration program that issues PCF commands. In these cases, the principal associated with a program is not necessarily the user ID that was used when the program was started.

Managing access permissions to MQSeries resources is based on *user groups*, that is, groups of principals. The OAM does not maintain authorizations at the level of individual principals. The mapping of principals to group names is carried out within the OAM, and operations are carried out at the group level.

The authorizations that a principal has are the union of the authorizations of all the groups of which it is a member, that is, its *group set*. Whenever a principal requests access to a resource, the OAM computes this union, and then checks the authorization against it.

Resources you can protect

Through MQSeries on UNIX systems you can control:

- **Access to queue manager objects through the MQI**

  When an application program attempts to access an object, the OAM checks to see if the principal making the request has the authorization (through its user group) for the operation requested. In this way, the queues, and the messages on queues, can be protected from unauthorized access.

- **Permission to use MQSeries commands**

  Only members of authorized user groups can execute queue manager administration commands.

  When a member of a user group attempts to execute a command, the OAM checks to see if the principal making the request has the authorization (through its user group) to use the command. These access control checks are performed irrespective of whether the commands are issued through the **runmqsc** (run MQSC commands) command, or through an application that uses PCFs.

Different groups of users can be granted different kinds of access authority to the same object. For example, one group might be allowed to read (MQGET) messages from a queue, but not to write (MQPUT) messages to that queue. Other groups might be given authority to put messages to and get messages from the same queue. Similarly, some groups might have get and put authority but not be allowed to create or delete queues.

Using groups for authorizations

Using groups for authorization, rather than individual principals, reduces the amount of administration required. Normally, 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 simply by adding their user ID to the appropriate group.

Try to keep the number of groups as small as possible. Dividing principals into one group for application users, and one for administrators, is a good place to start.

Using the security commands

The OAM provides two commands that you can use to manage the authorizations of users. These are:

- **setmqaut** - set or reset authority
- **dspmqaut** - view authority

When you start a queue manager, it uses a group ID of **mqm**, and a user ID of **mqm**. These must be defined before you can start a queue manager. After this, any principal belonging to the group **mqm** can issue **setmqaut** commands to change authorizations to resources.

Details of these commands can be found in the *MQSeries System Administration* manual.

Security exits

The message channels that are used for distributed queuing, and the MQI channels that are used between clients and servers, both have security exit facilities that can invoke programs that you have supplied.

For more information on these security exit programs, see the *MQSeries Intercommunication* manual.

Chapter 40. Administration of MQSeries on UNIX systems

This chapter is a summary of the administration facilities provided by MQSeries on UNIX systems. It has the following sections:

- "Managing objects"
- "Remote administration" on page 190

Details of the commands, command interfaces, and utilities that are provided by MQSeries on UNIX systems are given in the appropriate manual (see Table 15 on page 175).

You need to arrange for users who need to be able to use these administration facilities to have the necessary authorizations, using the procedures given in the appropriate manual (see Table 15 on page 175).

Managing objects

It is the administrator's job to monitor MQSeries on UNIX systems and make any changes that might be necessary. To do this, the administrator needs to know where each MQSeries object resides, what its characteristics are, and who has access to it.

The administrator can manage and monitor the resources using MQSeries commands (MQSC), or, if there are sets of commands that are issued regularly, by writing an application program that places them on the command queue.

MQSeries can use the security features provided by the OAM, or by a security component that you have installed, to ensure that the user is authorized to issue particular commands for particular resources.

Commands

MQSeries supports the following administration commands and facilities:

- You can enter control commands on the command line
- You can use the **runmqsc** control command to cause MQSC commands from standard input to be executed
- Any local or remote MQSeries application program can generate PCF commands in messages and put them to the command queue SYSTEM.ADMIN.COMMAND.QUEUE, to be processed by the MQSeries on UNIX systems command server

More information on how to use all these facilities is given in the appropriate manual (see Table 15 on page 175).

Changes made to the resource definitions of a queue manager using the commands (directly or indirectly) are preserved across restarts of MQSeries.

Managing communications

Part of the administrator's role is to ensure that the required communications links are activated, and to monitor the status of these links as required by your enterprise. You can find information describing these tasks in the *MQSeries Intercommunication* manual.

Remote administration

There are two aspects to the MQSeries remote administration facilities:

- The local MQSeries system can be used to manage remote systems
- Other remote products can be used to manage the local MQSeries system

Managing remote systems

Facilities are provided by MQSeries to allow an administrator to manage the following remote systems:

- MQSeries for AS/400
- MQSeries for Digital OpenVMS
- MQSeries for MVS/ESA
- MQSeries for OS/2 Warp
- MQSeries for Tandem NSK
- MQSeries on UNIX systems
- MQSeries for Windows NT

Because of the differences in the MQSeries products, it is not always possible to manage remotely the same set of MQSeries objects or attributes that you can manage locally.

If you want to manage any other MQSeries product, you can write an application program to send the appropriate commands to the command queue for that product. However, some MQSeries products do not have a command queue, so they cannot accept commands from local or remote application programs.

Managing MQSeries from remote systems

The local MQSeries system can be managed from a remote MQSeries system, by an administrator using the facilities provided by the following products:

- MQSeries for AS/400
- MQSeries for Digital OpenVMS
- MQSeries for MVS/ESA
- MQSeries for OS/2 Warp
- MQSeries for Tandem NSK
- MQSeries on UNIX systems
- MQSeries for Windows NT

Note: You can manage MQSeries on UNIX systems from MQSeries for MVS/ESA by writing an application program to send the appropriate PCF commands to the command queue.

Chapter 41. Storage planning for MQSeries on UNIX systems

This chapter tells you how to plan the type and amount of storage you require when you include MQSeries on UNIX systems in your network. It has the following sections:

RAM considerations

The processor memory (RAM) is used by MQSeries on UNIX systems in the execution of the product modules, and as a paging area for the messages that are being processed. Parts of the paging area are written to, and read from, disk as necessary.

The minimum amount of RAM required to run the MQSeries on UNIX systems server is 24 MB; if more RAM is available, the performance of the message processing improves.

The amount of RAM required on each client system for an MQSeries client is small compared to that required for the operating system on each of the platforms.

Disk space considerations

Disk space is used by MQSeries on UNIX systems for the following:

- Product modules - client and server executable modules, the toolkit, and the online documentation
- Paging space - server only
- Message queues - server only
- Logs - server only

Product modules

The disk space that you require for the product modules depends on the options that you decide to install: the options are described in the appropriate manual (see Table 15 on page 175).

Space might be required for client and server executable program modules and the toolkit. If all options are selected, 20 MB is required. You will need approximately 15 MB more than this if you install the online documentation.

The product modules can be stored on a LAN server, as an alternative to them being stored on a disk attached to the client or server. MQSeries loads the modules from the LAN server when required.

Message queues

In order to estimate the total amount of storage that you will need for queues, you need to know:

- The number of queues that you have.
- The maximum number of messages there will be on each of the queues at any one time.
- The average size of message on each of the queues. The amount of storage required for one message varies. It is based on the size of the message data plus the size of the message header (456 bytes), rounded up the nearest 512-byte block. If you are using distribution lists, or grouped or segmented messages, the size of the header will increase for the transmission queue.

Given these values, you can calculate the total amount of space required for queues. However, this value is likely to be an approximate value only, and it is advisable to add a contingency value, to avoid the situation where there is no space left for messages on the queues when your application is running.

Log files

Significant events and data changes can be logged in circular or sequential logs. In particular, the logs are used for recording persistent messages.

All the log files in a log are of the same size. By default, this size is 4 MB, but this value can be changed by the system administrator when the log is defined.

For a circular log, the system administrator needs to specify how many files should be included in the log. For a sequential log, the number of files will increase over time, until the system administrator archives some of the files, and disposes of them. You need to plan for the permanent storage (diskettes, tape, or other media supported in your enterprise) that is to be used for these archived files.

Capacity planning and performance figures

Information about MQSeries performance and capacity planning is available on the Internet at:

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

For AT&T GIS UNIX, SINIX and DC/OSx, and SunOS, use the information about MQSeries for AIX as a starting point.

Part 8. Planning for MQSeries Three Tier

Chapter 42. Introduction to MQSeries Three Tier

MQSeries Three Tier for OS/2 and MQSeries Three Tier for AIX (both referred to in this book as 3T) provide application design, application development, and application testing services to help you to create message queuing applications that run in a client/server environment. Such applications have the following characteristics:

- They need to access data held in departmental, corporate, or third-party databases
- They use windowing, event-driven user interfaces
- They can comprise modular, reusable components

Client/server applications usually perform three types of work:

- They interact with end users, presenting information and collecting it through a user interface. In 3T, this part of the application is known as the *presentation logic* (PL). It runs on a user's workstation.
- They manipulate the data they pull from a database or gather from the user. In 3T, this part of the application is known as the *business logic* (BL). It runs on a server.
- They retrieve and update data from one or more databases. In 3T, this part of the application is known as the *data logic* (DL). It runs on a server or a mainframe computer.

This is the *three-tier model* of client/server computing. 3T helps you to design and create these three types of logic, and make them communicate with each other. Figure 14 shows the relationship between the three tiers.

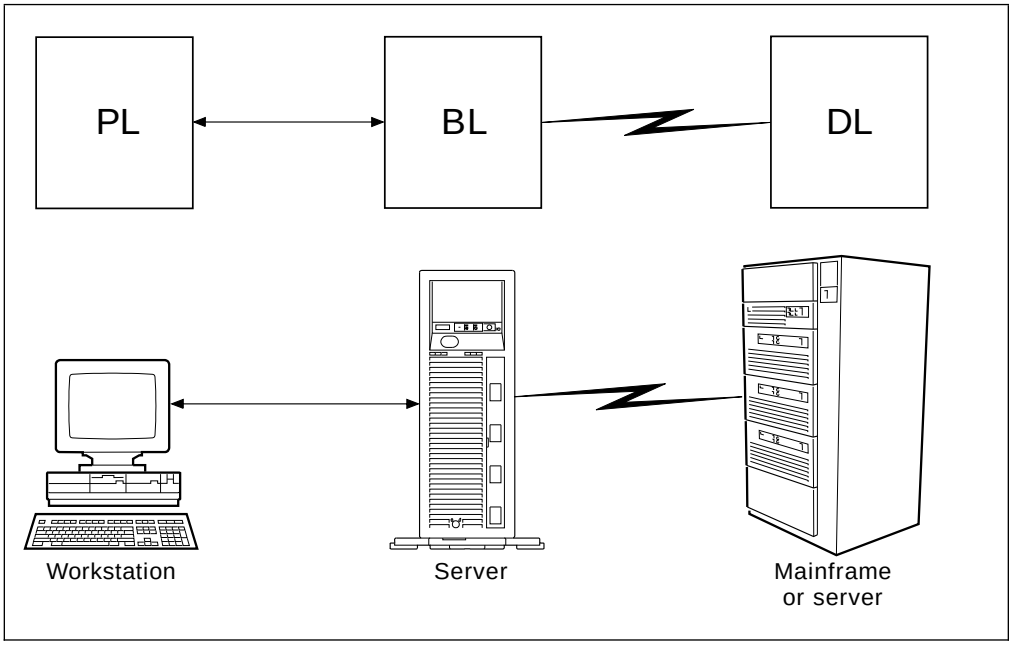

Figure 14. The three tiers of a 3T application

In a 3T application, each of the three tiers is further divided into *classes*, each corresponding to a business object that the application processes; for example, a customer or an account. So the application comprises a network of named classes that communicate with each other (see Figure 15). These classes can each run on separate computers; for this reason, such applications are often known as "distributed" applications.

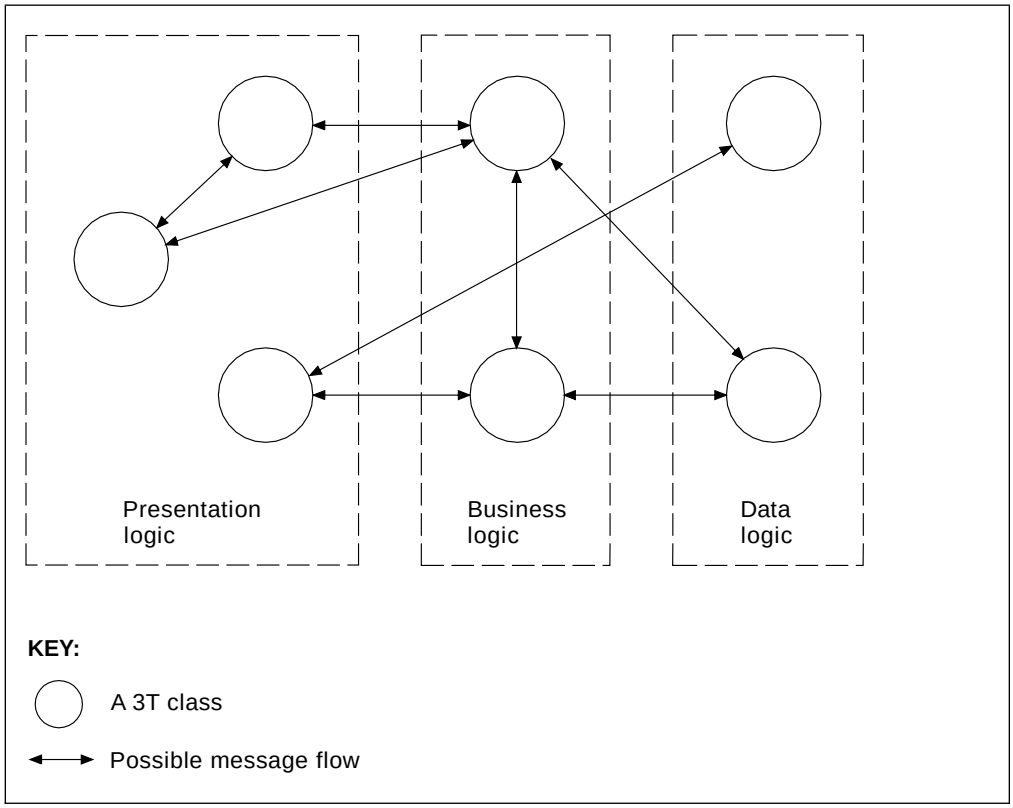

Figure 15. 3T classes and the three-tier, client/server model

3T uses MQSeries messages for communication between classes. This gives the following advantages:

- Applications can be asynchronous in their operation, that is, one class can communicate with a second class without the second one being ready to receive that communication
- Programs can communicate across many platforms without you having to write any communication code
- The MQSeries products ensure that messages are delivered

Each class consists of one or more *methods* that perform the work of the class; for example, one method could get the name of a customer, another could get the customer's account number.

The following sections describe each of the three tiers. Note that although it is usually convenient to run each tier on a separate computer, 3T does not require this.

Interacting with end users (presentation logic)

The presentation logic (PL) provides the client function of the application. It is the part of the application that runs on a user's workstation; this is the part with which the end user interacts. Its classes use programs that prompt the user to enter data (such as a name and address) and that present the results of the application (such as the balance of a bank account) to the user.

The PL programs are self-contained and are suitable for reuse by other applications. The users of the applications can be more productive because you can give them similar interfaces for all the applications they use.

The methods that constitute a PL class are packaged together into a single executable program. A PL program is a collection of PL methods, all supporting the same class. You can write the methods in C, COBOL, or PL/I, or build them using a GUI-building tool such as VisualAge. They can run under Windows and OS/2 on workstations.

The PL classes interact with the rest of the application by sending and receiving messages. They can send messages directly to a DL class; they do not have to go through a BL class, although most applications will use BL classes.

Presentation logic manager

A 3T Presentation Logic Manager (PLM) runs on every user's workstation. One copy of the PLM supports many 3T classes. It transmits messages between the PL classes it supports and other BL and DL classes.

Manipulating data (business logic)

The business logic (BL) provides the server function of an application. It comprises the classes that manipulate the data entered by the end user and any extra data retrieved from a database. These are the classes that perform the bulk of the work of your application, and they are specific to your business. They perform tasks such as compiling a quotation for a set of goods, linking customer information from two databases (for example, linking a name and an address with a policy number), or calculating share dividends.

You can benefit by creating small, reusable programs (the methods) for this part of your application. 3T provides features that allow your application to select the methods it runs depending on the type of data it receives. You can also define a *work area* in which the BL class can store data while it is working.

You can write BL methods in C, COBOL, or PL/I. They can run on OS/2 and AIX servers. You can package them separately or as a single code library.

A BL class interacts with the remainder of the application by sending and receiving messages.

Business logic manager

A 3T application needs one copy of the 3T Business Logic Manager (BLM) for each BL class. The BLM runs on a server. It transmits messages between the BL class it supports and other classes.

Retrieving and updating data (data logic)

Your application will probably need to query one or more databases to retrieve the information it needs to satisfy the requests of the end user. It will probably also need to update the information held in these databases. These databases can be on a mainframe computer or on a local server. The part of a 3T application that performs these tasks is the data logic (DL).

In some applications, you might want to combine the data logic with the business logic, especially if the database you need to query is on a local server.

You need at least one DL class for each database system (for example, CICS or IMS) you need to access. However, it is advisable to create a DL class for each logical group of data to avoid having to change the application if the data is moved to another system.

A DL class consists of transactions that read information from, and write information to, a database. Usually the transactions are responsible for maintaining the integrity of the information held in that database.

In many cases, you will already have programs to access your databases, and there is no need to rewrite them to take advantage of the features of 3T. Just use the application design features of 3T to identify these programs as DL classes; at run time, 3T sends messages to them. This means you can preserve the investment your organization has already made in creating its databases.

If you need to write new DL classes to retrieve data from a database, you need to do so only once. For each new application that needs to query a particular database, you can reuse the classes you have already written.

A DL class does not run under the control of 3T, but it uses the MQSeries Message Queue Interface (MQI) to receive (and to respond to) requests for queries and changes to a database. A DL class can run on any server or mainframe computer that can be accessed by an MQSeries product. You can write DL classes in any of the languages that are available in the environment of the database.

3T application development tools

3T provides the following tools to help you to design and develop applications:

An Application Simulator
To simulate message flows and help you to assess the performance of your application before you write any code

A Class Compiler
To compile the class definitions you write in your class source files

A 3T Part for use with VisualAge
To help you to create PL programs visually

Sample applications
To help you to understand the structure of 3T applications and the features provided by 3T

3T is an enhancement of the MQI

Programmers of applications that use the MQI must write programs that deal with the arrival of messages at any time. The program that processes messages must decide what to do with each message. It must answer questions such as:

- Did I expect this message?
- Is this one of a series of messages I am expecting?
- If the message arrives later than I expected, can I still use it?

In a client/server environment, these questions are more important because of the complex nature of the applications. In a typical client/server application:

- A PL class sends requests and waits for replies.
- A BL class receives requests from PL classes. It sends one or more requests to other BL and DL classes, and waits for replies.

So there are many messages involved.

3T helps the programmer by:

- Routing messages to the appropriate piece of code (the method) depending on the circumstances under which the message arrived
- Routing many reply messages to the same method
- Routing to a method all the replies that arrive before a timer expires
- Routing late replies to a separate method
- Detecting duplicate and out-of-sequence messages
- Providing an integrated design, development, and test environment in which to create client/server applications

In addition to processing messages, a client/server application might have to save “work-in-progress data”; that is, save data while it is waiting for other messages to arrive or for more user input. 3T provides a work area for this purpose for each instance of a BL class. This work area is retained at the end of a method, so a method can use the work area to pass data to another method. You can choose to make 3T preserve the work area on a queue so that the instance can recover in the event of the server having to be restarted.

By managing this complexity for a client/server application, 3T helps programmers to be productive and allows them to concentrate on writing code that is directly related to the business of their organization.

Further information

Further information on writing 3T applications is given in:

- *MQSeries Three Tier Application Design*
- *MQSeries Three Tier Application Reference*

Information about the management of 3T is given in:

- *MQSeries Three Tier Administration Guide*

A reference summary is also provided:

- *MQSeries Three Tier Reference Summary*

Chapter 43. MQSeries Three Tier planning

3T installations are based on a three-tier data model. Figure 16 shows schematically how the three tiers are related to the MQSeries base product. In a typical production installation, there would be many, perhaps hundreds, of Presentation Logic Managers (one per workstation); many servers, each supporting multiple Business Logic Managers; and as many DL servers as required.

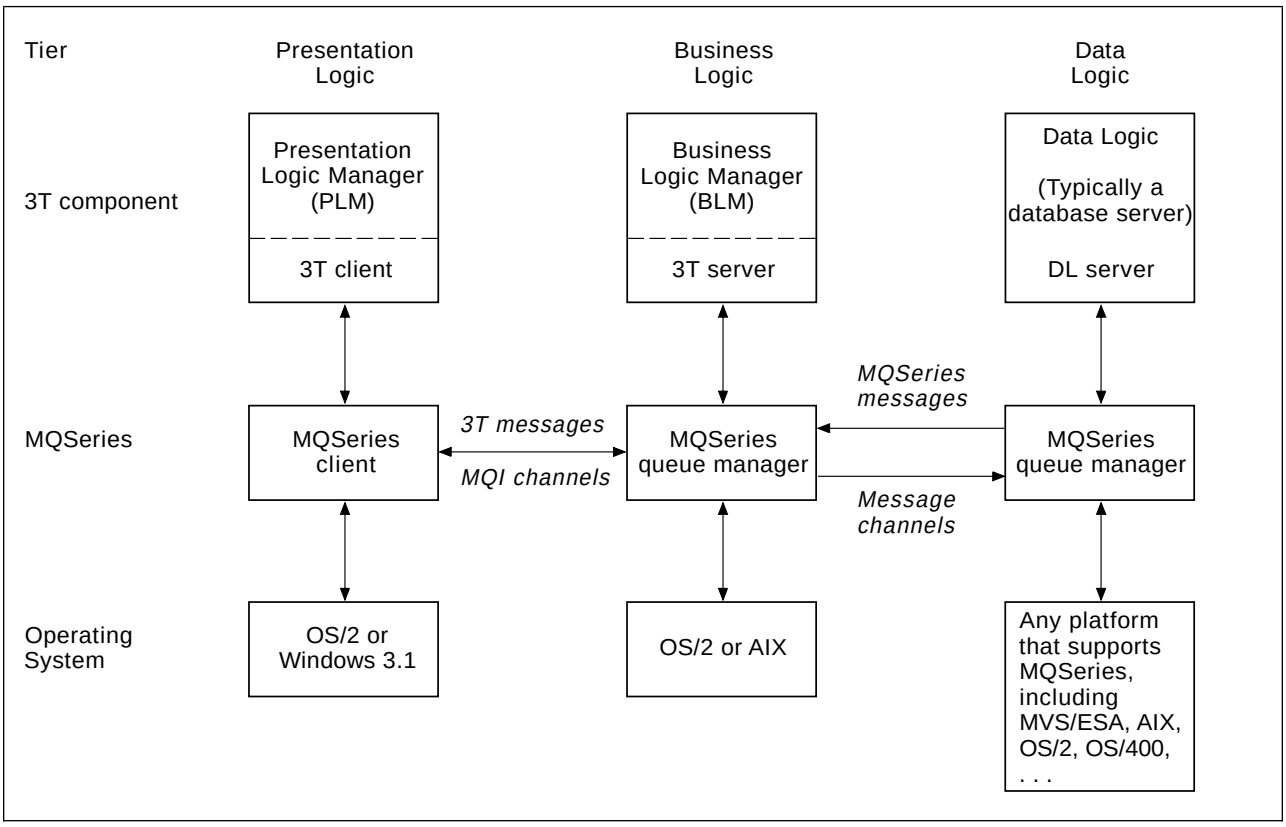

Figure 16. Relationship between 3T and MQSeries in the 3-tier data model. The three tiers are presentation logic, business logic, and data logic. The DL server does not contain any 3T code.

In this model, the *server* part of 3T contains the Business Logic Manager and its associated classes and methods. The server is a central resource that can be used by many clients. Typically, servers require more powerful machines than clients in terms of the processor type, speed (clock frequency), RAM, and disk space. You can run a server on an AIX (RS/6000) machine or on an OS/2 machine.

The *client* contains the Presentation Logic Manager with its classes and methods. A *client* component is dedicated to an individual user performing specific presentation functions; for example, functions related to a sales representative or an order clerk. A 3T client must reside on each user workstation, and can run on OS/2 or Microsoft Windows 3.1.

How the MQSeries base product fits in

3T uses MQSeries base product resources including queue managers, queues, and channels. You must configure the MQSeries base product for each tier in this model. Figure 16 on page 201 summarizes the relationship between MQSeries and each tier. On each workstation, you create an MQSeries base installation that supports the required 3T installation. Typically, 3T clients are installed with MQSeries clients on the same workstation; 3T servers are installed on the same machine as MQSeries servers. You must configure MQSeries channels between:

- A 3T server and each of its 3T clients
- A 3T server and any other 3T servers, as required
- A 3T server and a DL server

MQSeries queuing requirements

Each 3T server must have an MQSeries queue manager installed. Each user workstation requires a 3T client and an MQSeries client to be installed. Optionally, 3T clients can run under an MQSeries queue manager, but this is recommended only where data recovery from user workstations is a business requirement.

The following MQSeries queues are required for the exclusive use of 3T:

- One MQSeries queue for each client. Typically, this queue is located on the server machine.
- Two queues for each server machine, plus one queue for each Business Logic Manager supported on the machine.

The MQSeries Three Tier products

MQSeries Three Tier is supplied as two separate products, based on the target server:

- MQSeries Three Tier for OS/2
- MQSeries Three Tier for AIX

These products have different product numbers and can be ordered separately.

Choose the product for the type of server you will use. When developing an application, you can use either product.

Note: You can use the Application Simulator and the Part for VisualAge with MQSeries Three Tier for OS/2 only.

Table 16 on page 203 summarizes the combinations of servers and clients that are available for each product. Table 17 on page 203 lists the components for each product.

Table 16. 3T clients and servers in different target environments		
Product	**Target environment**	
	Client	**Server**
MQSeries Three Tier for AIX	Windows 3.1	AIX
	OS/2	AIX
MQSeries Three Tier for OS/2	Windows 3.1	OS/2
	OS/2	OS/2

Table 17. Components supplied with MQSeries Three Tier		
Component	**Supplied with MQSeries**	
	Three Tier for AIX	**Three Tier for OS/2**
Development tools for OS/2 clients (runs on OS/2)	Yes	Yes
Development tools for Windows 3.1 clients (runs on Windows 3.1)	Yes	Yes
Run-time client for OS/2	Yes	Yes
Run-time client for Windows 3.1	Yes	Yes
Run-time server for AIX	Yes	No
Run-time server for OS/2	No	Yes
Application Simulator	No	Yes
Class Compiler	Yes	Yes
Part for VisualAge	No	Yes
Sample applications for AIX servers (in C)	Yes	No
Sample applications for OS/2 servers (in C, COBOL, and PL/I)	No	Yes
Development tools for AIX servers	Yes	No
Development tools for OS/2 servers	No	Yes

Networking considerations

Depending on the configuration, you can use 3T with the network protocols as defined in Table 18.

Table 18. Communications protocols and 3T			
	AIX	**OS/2**	**Windows 3.1**
LU 6.2	Yes	Yes	-
NetBIOS	-	Yes	Yes
TCP/IP	Yes	Yes	Yes

Disk space requirements for 3T

To install 3T successfully, you must allocate enough disk storage for the product. This depends on whether the machine is for run-time or development and whether it is a client or a server. The disk storage required includes space for the product code, for the applications you develop, working storage, online documentation, and so on. Table 19 shows the disk space requirements on different operating systems.

Table 19. 3T disk space requirements. (1 MB is 1 048 576 bytes)

Configuration and operating System	**Disk space in MB**
3T server (BL) development for AIX	10-12
3T server (BL) development for OS/2	12-14
3T run-time server for AIX	6.0
3T run-time server for OS/2	6.0
3T client (PL) development for OS/2	6.0
3T client (PL) development for Windows 3.1	6.0
3T run-time client for OS/2	5.0
3T run-time client for Windows 3.1	3.0

Managing 3T clients and servers

3T provides its own tools to help you run a network of servers and clients and to enable data recovery in case of a failure.

3T provides user commands to start and stop BLMs (on 3T servers) and PLMs (on 3T clients). The operation of a PLM (or BLM) is determined by a profile, which is invoked when the PLM (or BLM) is started. For a BLM, the profile includes specifications for:

- Whether error logging is switched on, and if so, what aspects of the server's operation are to be logged
- How operating system threads are to be used
- The names of the MQSeries queues that are to be used
- Timeouts for rules

By changing the profile, you can change the way in which the BLM works.

3T also provides a set of *server commands* that you can build into an administration application to monitor or change the operation of a server dynamically. To do this, you specify the attributes you want to change or read in an MQSeries Programmable Command Format (PCF) message, which you send to the queue manager on the target server.

3T recovery and restart

For BL servers, 3T allows a server to be restarted after a failure, without loss of data.

You do this by specifying a server as *hard*. In 3T, a hard server can recover its state and any uncommitted messages after a system failure. All the processing for a method is carried out under queue manager syncpoint control. 3T stores its current state in a persistent MQSeries message, which it puts on an MQSeries message queue. When the state changes, the new state is stored on the queue; the old state is overwritten. Only 3T servers (with BL classes) can be hardened; 3T clients cannot.

Part 9. The MQSeries family

Chapter 44. MQSeries product summaries

This chapter gives summary information for all the products in the MQSeries family. It has the following sections:

Lists of MQSeries products

At the time of publication of this manual, MQSeries products were announced or available for the hardware and software platforms in the lists below.

The products are arranged in two groups, Level 1 and Level 2, for convenience when referring to products within this manual.

Some of the products were renamed in November 1994; for those products, the name by which the product used to be known is given, as well as the current product name.

Level 1 products

Table 20. MQSeries products, Level 1

Product name	Previous name
MQSeries for SCO UNIX V1.4	ezBRIDGE Transact on SCO UNIX for MQSeries Release 3.0
MQSeries for UnixWare V1.4.1	ezBRIDGE Transact on UnixWare for MQSeries Release 3.0
MQSeries for VSE/ESA V1.4	ezBRIDGE Transact on VSE/ESA for MQSeries Release 3.0

Level 2 products

Table 21 (Page 1 of 2). MQSeries products, Level 2

Product name	Previous name
MQSeries for AIX V5.0 (clients and servers)	None (new version)
MQSeries for AT&T GIS UNIX V2.2 (clients and servers)	None (new release)
MQSeries for Digital OpenVMS V2.2 (clients and servers)	None (new release)
MQSeries for HP-UX V5.0 (clients and servers)	None (new version)
MQSeries for MVS/ESA V1.2	None (new level)
MQSeries for OS/2 Warp V5.0 (clients and servers)	None (new version)
MQSeries for AS/400 V4R2	None (new release)
MQSeries for SINIX and DC/OSx V2.2 (clients and servers)	None (new product)
MQSeries for SunOS V2.2 (clients and servers)	None (new product)
MQSeries for Sun Solaris V5.0 (clients and servers)	None (new version)

Table 21 (Page 2 of 2). MQSeries products, Level 2	
Product name	**Previous name**
MQSeries for Tandem NonStop Kernel V2.2	None (new release)
MQSeries Three Tier for AIX	None (new product)
MQSeries Three Tier for OS/2	None (new product)
MQSeries for Windows V2.0	None (new product)
MQSeries for Windows V2.1	None (new release)
MQSeries for Windows NT V5.0 (clients and servers)	None (new version)

MQSeries interoperability summary

The tables in this section show the transmission protocols that are supported by the channels in each of the MQSeries products. Tables are provided for the two types of channel that are supported by the MQSeries products:

- **Message channels** - used to link MQSeries queue managers
- **MQI channels** - used to link MQSeries clients and servers

The tables can be used to determine the alternative protocols that can be used to link any two products together, as follows:

1. Find the row in the table that contains the first MQSeries product of interest.
2. Find the row that contains the second MQSeries product of interest.
3. Look at each column position in each of these rows. Where there is a column that has a check symbol √ in both rows, then the transmission protocol associated with that column can be used to link the two products.

The “at a glance” sections in Chapter 45, “MQSeries at a glance” on page 215 give details of the prerequisite hardware and software necessary at the two ends of a link for a particular transmission protocol.

Message channels - transmission protocols supported

Table 22 (Page 1 of 2). Message channels, transmission protocols supported					
MQSeries product	**SNA LU 6.2**	**TCP/IP**	**NetBIOS**	**DECnet**	**SPX**
MQSeries for AIX	√	√	—	—	—
MQSeries for AT&T GIS UNIX	√	√	—	—	—
MQSeries for Digital OpenVMS	√	√	—	√	—
MQSeries for HP-UX	√	√	—	—	—
MQSeries for MVS/ESA	√	√	—	—	—
MQSeries for OS/2 Warp	√	√	√	—	√
MQSeries for AS/400	√	√	—	—	—
MQSeries for SCO UNIX	√	√	—	—	—
MQSeries for SINIX and DC/OSx	√	√	—	—	—
MQSeries for SunOS	√	√	—	—	—
MQSeries for Sun Solaris	√	√	—	—	—
MQSeries for Tandem NSK	√	√	—	—	—
MQSeries for UnixWare	√	√	—	—	—

Table 22 (Page 2 of 2). Message channels, transmission protocols supported

MQSeries product	SNA LU 6.2	TCP/IP	NetBIOS	DECnet	SPX
MQSeries for VSE/ESA	√	—	—	—	—
MQSeries for Windows	—	√	—	—	—
MQSeries for Windows NT	√	√	√	—	√

MQI channels - transmission protocols supported

Table 23. MQI channels, transmission protocols supported by servers

MQSeries servers	SNA LU 6.2	TCP/IP	NetBIOS	DECnet	SPX
Digital OpenVMS	—	√	—	√	—
MVS/ESA, Tandem NSK	√	√	—	—	—
OS/2 Warp, Windows NT	√	√	√	—	√
OS/400	√	√	—	—	—
UNIX systems	√	√	—	—	—

Table 24. MQI channels, transmission protocols supported by clients

MQSeries clients	SNA LU 6.2	TCP/IP	NetBIOS	DECnet	SPX
OS/2 Warp, Windows NT	√	√	√	—	√
UNIX systems, VM/ESA	√	√	—	—	—
DOS, Windows 3.1, Windows 95	—	√	√	—	√
Digital OpenVMS	√(1)	√	—	√	—

Note: (1) On Digital OpenVMS, SNA LU 6.2 supports only PU 2.0. Therefore communication can be to PU 5.0 only on an MVS/ESA server. If you want to communicate with a server on a platform other than MVS/ESA, you must use another protocol. See the *MQSeries Clients* manual for more information.

MQSeries product functional comparison

Table 25 is a summary of the MQI functions that are provided by the MQSeries products. Those functions that are marked with the symbol √ are supported by all products identified by the column heading, except where indicated by the notes following the table.

Table 25 (Page 1 of 2). MQSeries product functional comparison

Function	Digital OpenVMS	MVS/ESA	OS/400	Tandem NSK	UNIX systems, OS/2 Warp, Windows NT (1)	Windows V2.0 & V2.1	Level 1 products (1)
MQCONN/MQDISC	√	√	√	√	√	√	√
Queue manager groups	√ (2)	—	—	√ (2)	√ (2)	—	—
MQOPEN/MQCLOSE	√	√	√	√	√	√	√
Queue-manager aliases	√	√	√	√	√	√	√
Reply-to queue aliases	√	√	√	√	√	√	√
Default input open option	√	√	√	√	√	√	—
Model/dynamic queues	√	√	√	√	√	√	—
Namelists	—	√	—	—	—	—	—
Default transmission queue	√	√	√	√	√	√	—
Distribution lists	—	—	√	—	√ (3)	—	—
MQPUT	√	√	√	√	√	√	√
Max message length	4 MB	4 MB	4 MB	4 MB	100 MB(4)	4 MB	(5)
Dead-letter queue	√	√	√	√	√	—	√
Nonpersistent messages	√	√	√	√	√	√	—
Triggering (first and every)	√	√	√	√	√	—	— (6)
Full triggering (depth, priority)	√	√	√	√	√	—	—
Message priority	√	√	√	√	√	√	—
Application-specified syncpoint	√	√	√	√	√	√	— (7)
Context	√	√	√	√	√	√ (8)	— (9)
Exception reports	√	√	√	√	√	√	—
Exception reports with data	√	√	√	√	√	√	—
COA, COD reports	√	√	√	√	√	√	—
Message expiry	√	√	√	√	√	√	—
Report options for up-level	√	√	—	√	√	√	—
MQGET	√	√	√	√	√	√	√
Browse	√	√	√	√	√	√	√ (10)
Browse with lock	√	—	√	√	√	—	√ (11)
Browse under cursor	√	—	√	√	√	√	—
Shared input	√	√	√	√	√	√	√
Get with signal	—	√	—	√	—	√ (12)	—
Get by MsgId/CorrelId	√	√	√	√	√	√	—
Message backout count	√	√	√	√	√	√	—
Mark skip backout	—	√	—	—	—	—	—
Message data conversion	√	√ (13)	√	√	√	—	—
MQINQ	√	√	√	√	√	√	√ (14)
Queue retention interval	√	√	√	√	√	—	—
MQSET	√	√	√	√	√	√	—
MQCMIT/MQBACK	√	√	—	√	√	√	—
MQBEGIN	—	—	—	—	√ (3)	—	—

Table 25 (Page 2 of 2). MQSeries product functional comparison

Function	Digital OpenVMS	MVS/ESA	OS/400	Tandem NSK	UNIX systems, OS/2 Warp, Windows NT (1)	Windows V2.0 & V2.1	Level 1 products (1)
API crossing exit	—	√ (15)	—	—	—	—	—
Message channel exits	√	√	√	√	√	√	—
Events	√	√	√	√	√	—	—
Segmented messages	—	—	√	—	√ (3)	—	—
Reference messages	—	—	√	—	√ (3)	—	—
Fast channels	—	√	√	—	√ (3)	√ (12)	—
Message retry exit	√	—	√	√	√	—	—
Channel heartbeats	—	√ (16)	√	—	√ (3)	—	—
Channel auto-definition	—	—	√	—	√ (3)	—	—

Notes:

1. Functions similar on clients and server.
2. From client applications only.
3. AIX, HP-UX, OS/2 Warp, Sun Solaris, and Windows NT only.
4. In the DOS environment, limitations might be imposed by memory availability. On MQSeries for AT&T GIS UNIX, SINIX and DC/OSx, and SunOS, the maximum message length is 4 MB.
5. 64 000 on SCO UNIX and UnixWare, 30 000 on VSE/ESA; all sizes exclude descriptors.
6. Triggering is supported on VSE only, but without a real initiation queue.
7. Always coordinated on VSE; never coordinated on SCO UNIX, or UnixWare.
8. Context passing is not supported.
9. Context fields are conveyed without any checks.
10. Browse next on VSE only.
11. Must do *Browse First and Lock* (except on VSE).
12. Version 2.1 only.
13. Not supported by the CICS/MVS adapter. It is only supported by the CICS/ESA adapter.
14. For queues only.
15. CICS only.
16. Not for MQI channels.

Chapter 45. MQSeries at a glance

This chapter gives information about the requirements for each MQSeries platform.

Table 26. MQSeries products at a glance

For information about...	See page...
AIX	216
AS/400	218
AT&T GIS UNIX (1)	219
Digital OpenVMS	220
DOS client	221
HP-UX	222
MVS/ESA	224
OS/2 Warp	228
SCO UNIX	231
SINIX and DC/OSx	233
SunOS	235
Sun Solaris	237
Tandem NonStop Kernel	239
Three Tier for AIX	241
Three Tier for OS/2	242
UnixWare	244
VM/ESA client	246
VSE/ESA	247
Windows (16-bit)	248
Windows (32-bit)	250
Windows NT	252
Windows 3.1 client	254
Windows 95 client	255
MQSeries link for R/3	256
Note: (1) This platform has become NCR UNIX SVR4 MP-RAS, R3.0	

MQSeries for AIX

MQSeries for AIX, Version 5

Machine requirements

MQSeries for AIX runs on any IBM RS/6000 that is capable of running the required level of AIX, and that has sufficient storage to meet the combined requirements of the programming prerequisites, MQSeries for AIX, the access methods, and the application programs:

- IBM RS/6000 POWERserver
- IBM RS/6000 POWERstation
- IBM Scalable POWERparallel systems
- Any other trademarked AIX systems, whether from IBM or other vendors, for example:
  - Bull** DPX/20 (RISC)
  - Bull** Escala (SMP)
  - Motorola**
  - Zenith**

An MQSeries client can run on any personal computer that is capable of running the client code and that has sufficient storage to meet the combined requirements of the programming prerequisites, the client code, access methods, and the application programs.

Software requirements

Software requirements are identical for server and client AIX environments unless otherwise stated.

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

- AIX Version 4.1.4 or AIX Version 4.2 or later Version 4.x.

  Note: For Version 4.1.4, PTF U449790 is required if user data conversion of Greek, Cyrillic, Eastern European, Turkish, Japanese, or Korean language text longer than 2000 bytes is required. For Version 4.2, use level 4.2.1 for these languages.

Clients

Client code for AIX, DOS, HP-UX, OS/2 Warp, Sun Solaris, Windows NT, Windows 3.1, and Windows 95 workstations is distributed with the server code. The Windows 3.1 client can operate under Windows 3.1, Windows 95, or within the WIN-OS/2 environment under OS/2 Warp.

Client software provides a remote interface to an MQSeries server. Client support does not result in distributed coordination of units of work.

The MQSeries client for Java is also distributed with the server code; see the *MQSeries Clients* manual for more information.

Connectivity

- IBM Communications Server for AIX, Version 4.0
- TCP/IP (as part of the base operating system)

Options

- Transaction processing managers (server only)
  - Transaction Server for AIX, Version 4.0
  - BEA Tuxedo, Version 5.1 or 6.1
- Databases (server only)
  - Oracle7, Release 7.3.2.1
  - DB2 for AIX, Version 2.1.1
- IBM Software Servers
  - Communications Server for AIX, Version 4.0
  - Database Server for AIX, Version 4.0
  - Directory Security Server for AIX, Version 4
  - Internet Connection Server for AIX, Version 4.1.1
  - Internet Connection Secure Server for AIX, Version 4.1
  - Transaction Server for AIX, Version 4.0
- DCE
  - IBM Directory and Security Server for AIX, Version 4 and later compatible versions. This must be the U.S. Domestic version with DES encryption if running the MQSeries-supplied DCE send, receive, or message exits.
  - MQSeries DCE names and security modules are provided as part of MQSeries for AIX.

Languages and compilers

- C using IBM C for AIX, Version 3.1.4
- C and C++ using IBM C Set++ for AIX, Version 3.1
- COBOL using Micro Focus** COBOL for UNIX, Version 4.0
- COBOL using IBM COBOL Set for AIX, Version 1
- IBM PL/I set for AIX V1.1

Delivery

MQSeries for AIX is supplied on CD-ROM. Two CD-ROMs are supplied; one containing the MQSeries for AIX server and client, and the other containing the other MQSeries clients shipped with MQSeries for AIX.

Installation

MQSeries for AIX is installed using either **Installation assistant**, **smit** or **installp**.

The installation can be performed in approximately 5 minutes. Customization of the product is then required, the duration of this process being dependent on the individual requirements of the enterprise.

The *MQSeries for AIX V5.0 Quick Beginnings* booklet contains specific instructions for installing MQSeries for AIX.

MQSeries for AS/400 V4R2

IBM MQSeries for AS/400 Version 4 Release 2

Machine requirements

MQSeries for AS/400 V4R2 runs on any AS/400 processor capable of running the required level of OS/400 and which has enough processor storage to meet the combined requirements of the programming prerequisites, the access methods, and the application programs.

Software requirements

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

- OS/400 Version 4 Release 2

MQSeries for AS/400 clients

MQSeries for AS/400 supports but does not ship or configure MQSeries clients.

Connectivity

Connectivity can be through SNA LU 6.2 or TCP/IP.

Options

- External Transaction Processing Monitors
  - CICS for AS/400 V4R2

    To use the MQI in application programs that operate under CICS you also require the program libraries for the programming language that you are using

Languages and compilers

- ILE C for AS/400, Version 4.2 (5769-CX2)
- ILE COBOL for AS/400, Version 4.2 (5769-CB1)
- ILE RPG for AS/400, Version 4.2 (5769-RG1)
- IBM VisualAge for C++ for AS/400, Version 4.2 (5769-CX4)

ILE run-time is part of OS/400.

Delivery

MQSeries for AS/400 V4R2 is supplied on all tape and diskette media.

Installation

MQSeries for AS/400 is installed using the AS/400 GO LICPGM command. For installation instructions, refer to the *MQSeries for AS/400 Administration Guide*.

MQSeries for AT&T GIS UNIX

IBM MQSeries for AT&T GIS UNIX Version 2 Release 2

Machine requirements

MQSeries for AT&T GIS UNIX runs on any AT&T GIS 34xx, 35xx or 36xx system. A minimum of 20 MB of disk space is required.

Software requirements

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

- AT&T GIS UNIX SVR4 MP-RAS[3] Version 2.0.3.01 or later Version 2

MQSeries for AT&T GIS UNIX clients

Client code for AT&T GIS UNIX[3], OS/2 Warp, DOS, and Windows 3.1 workstations is distributed with the server code.

Connectivity

The network protocols supported are SNA LU 6.2 and TCP/IP.

- AT&T GIS SNA Services Version 2.06 or later Version 2 to match hardware system
- TCP/IP as part of base operating system

Options

- Transaction Processing monitors (coordination via X/Open XA interface)
  - Novell Tuxedo Version 4.2.2

Languages and compilers

- AT&T GIS High Performance C Version 1.0b
- AT&T C++ language system for AT&T GIS UNIX

Delivery

MQSeries for AT&T GIS UNIX V2.2 is supplied on CD-ROM or QIC tape.

Installation

MQSeries for AT&T GIS UNIX is installed using the **pkgadd** command.

The installation can be performed in approximately 5 minutes. Customization of the product is then required, the duration of this process being dependent on the individual requirements of the enterprise.

The *MQSeries for AT&T GIS UNIX System Management Guide* contains specific instructions for installing this product.

[3] This platform has become NCR UNIX SVR4 MP-RAS, R3.0

MQSeries for Digital OpenVMS

- MQSeries for Digital OpenVMS AXP Version 2 Release 2
- MQSeries for Digital OpenVMS VAX Version 2 Release 2

Machine requirements

Any Digital VAX machine with minimum system disk space of 16 MB for VAX or 18 MB for AXP and minimum memory of 16 MB for VAX or 32 MB for AXP.

Software requirements

Software requirements are identical for server and client Digital OpenVMS environments unless otherwise stated. Minimum supported levels are shown. Later levels, if any, are supported unless otherwise stated.

- Digital OpenVMS VAX Version 6.2 or later

Clients

Client code for Digital OpenVMS, OS/2 Warp, DOS and Windows 3.1 workstations is distributed with the server code. The Windows 3.1 client can operate under Windows 3.1, Windows 95 or within the WIN-OS/2 environment under OS/2 Warp.

Connectivity

Network protocols supported are DECnet, SNA LU 6.2, and TCP/IP.

- DECnet Phase IV
- DECnet OSI, Version 6.3
- Digital SNA APPC LU6.2 Programming Interface, Version 2.2
- Digital DECnet SNA Gateway, Version 1.2A
- Cisco Multinet for OpenVMS, Version 3.5
- DEC TCP/IP Services for OpenVMS, Version 4.0
- Attachmate PathWay for OpenVMS, Version 2.5.1
- Process Software's TCPware for OpenVMS, Version 5.2-3

Options

Distributed Computing Environment for OpenVMS, Version 1.3B

Languages and compilers

- DEC C, Version 5.0
- DEC C++, Version 5.0 (VAX) or 5.2 (AXP)
- DEC COBOL, Version 5.0 (VAX) or 2.2 (AXP)

Delivery

MQSeries for Digital OpenVMS is supplied on CD-ROM for AXP and on CD-ROM and TK50 tape cartridge for VAX.

Installation

MQSeries for Digital OpenVMS is installed with the OpenVMS VMSINSTAL utility, and takes approximately 10 minutes to install from CD-ROM. Customization of the product is then required, the duration of this process being dependent on the individual requirements of the enterprise.

MQSeries client for DOS

This section summarizes the machine and software requirements for the MQSeries DOS client.

Machine requirements

An MQSeries client can run under DOS on any personal computer that is capable of running the client code and which has sufficient storage to meet the combined requirements of the programming prerequisites, the client code, access methods, and the application programs.

Software requirements

The following are prerequisites for MQSeries applications running on a DOS client.

Minimum supported software levels are shown. Later levels, if any, are supported unless otherwise stated.

Workstation clients

Client code for DOS workstations is distributed with the server code for all servers except OS/400 and MVS/ESA.

- DOS 5.0

Options

- TCP/IP for OS/2 V2.0. The base kit is necessary. The DOS access kit allows clients access to TCP/IP via programs that run in a DOS box.
- TCP/IP V2.1.1 for DOS.
- IBM NetBIOS V2.1.1 for DOS/Windows.
- Novell IPX, using Novell NetBIOS emulation.

Languages and compilers

The following compilers are supported:

- C using Microsoft C/C++ Version 7.0
- C using Microsoft Visual C++ for Windows Version 4.0
- COBOL using Micro Focus COBOL Version 3.3

MQSeries for HP-UX

IBM MQSeries for HP-UX, Version 5

Machine requirements

The MQSeries for HP-UX servers can run on any HP 9000 Family 700 or Family 800 or Stratus Continuum/400 machine. A minimum of 20 MB of disk space is required.

Software requirements

Minimum supported levels are shown.

- HP-UX Version, 10.10 or later Version 10.x. Version 10.20 or later is required if you want to use the MQSeries client for Java.

Clients

Client code for AIX, DOS, HP-UX, OS/2 Warp, Sun Solaris, Windows NT, Windows 3.1, and Windows 95 workstations is distributed with the server code. The Windows 3.1 client can operate under Windows 3.1, Windows 95, or within the WIN-OS/2 environment under OS/2 Warp.

Client software provides a remote interface to an MQSeries server. Client support does not result in distributed coordination of units of work.

The MQSeries client for Java is also distributed with the server code; see the *MQSeries Clients* manual for more information.

Connectivity

The network protocols supported are SNA LU 6.2 and TCP/IP.

- HP SNAplus2
- TCP/IP as part of base operating system

Options

- Transaction processing monitors (server only)
  - CICS for HP9000, Version 2.1.1
  - BEA Tuxedo, Version 5.1 or 6.1
  - HP Encina/9000, Version 1.2
  - Transarc Encina, Version 2.5
- Databases (server only)
  - Oracle, Release 7.3.2.3 (with patches 437448 and 441647)
  - DB2 for HP-UX, Version 2.1.1
- IBM Software Servers
  - Internet Connection Server for HP-UX, Version 4.2.1
  - Internet Connection Secure Server for HP-UX, Version 4.2.1

- DCE
  - The HP DCE/9000 version appropriate for the level of the HP-UX operating system in use, providing that this is compatible with DCE Version 1.4.1. This must be the U.S. Domestic version with DES encryption if running the MQSeries supplied DCE send, receive, or message exits.

    Note: You must apply the required HP-UX service in order to use DCE. Contact your local HP support center to obtain a current list of the required patches.
  - MQSeries DCE names and security modules are provided as part of MQSeries for HP-UX.

Languages and compilers

- C using the bundled compiler or HP-UX ANSI C compiler
- C, using C Softbench, Version 5.0
- C and C++, using HP C++, Version 3.1
- COBOL, using the Micro Focus** COBOL compiler for UNIX, Version 4
- COBOL, using COBOL Softbench, Version 4.0
- HP DCE/9000 application development tools (with applicable patches)

Delivery

MQSeries for HP-UX is supplied on CD-ROM. Two CD-ROMs are supplied; one containing the MQSeries for HP-UX server and client, and the other containing the other MQSeries clients shipped with MQSeries for HP-UX.

Installation

MQSeries for HP-UX V5.0 is installed using the **update** command.

The installation can be performed in approximately 5 minutes. Customization of the product is then required, the duration of this process being dependent on the individual requirements of the enterprise.

The *MQSeries for HP-UX V5.0 Quick Beginnings* book contains specific instructions for installing this product.

MQSeries for MVS/ESA

IBM MQSeries for MVS/ESA Version 1 Release 2

Machine requirements

MQSeries for MVS/ESA runs on any IBM System/370 or System/390 processor that is capable of running the required level of MVS/ESA, and which has enough storage to meet the combined requirements of the programming prerequisites, MQSeries for MVS/ESA, the access methods, and the application programs.

Software requirements

The following tables list the program products that you need before you can install and use MQSeries for MVS/ESA. The tables show which versions of these products you can use, and list any APARs that you need to apply to your system before you install MQSeries. See the *MQSeries for MVS/ESA Program Directory* for the latest version of this information.

To install MQSeries for MVS/ESA, the prerequisite products listed in Table 27 are required.

Table 27. Installation requirements

Function	MVS	Runtime libraries	Other
Without optional communications features	4.3		SMP/E 1.8.0 with APAR IR29060
With optional IBM TCP/IP feature	4.3	LE/370 1.5 with the following APARs: • PN78163 • PN80015 • PN80739 • PN82174 • PN82557 • PN86618 • PQ03507	• SMP/E 1.8.0 with APAR IR29060 • TCP/IP 3.1 or 3.2 only
With optional Interlink SNS/TCPaccess feature	4.3	LE/370 1.5 with the following APARs: • PN78163 • PN80015 • PN80739 • PN82174 • PN82557 • PN86618 • PQ03507	• SMP/E 1.8.0 with APAR IR29060 • Interlink SNS/TCPaccess** 3.1 or later.

Notes:

1. Unless otherwise noted, any later level is acceptable.
2. The APARs listed might not apply to all versions of the prerequisite products.
3. If you are using the DFSMS/MVS binder utility, ensure that APAR OW05347 has been applied.

To use MQSeries for MVS/ESA, the prerequisite products listed in Table 28 on page 225 are required.

Table 28. Programming requirements

Function	MVS	Other
General	4.3 with the following APARs: • OW07179 • OY64365 • OY67606	• JES2 or JES3 3.1.0 • DFP 3.1 • ESA machine
Security	4.3	RACF 2.1
Measured usage license charges	4.3	MVS APAR OW02855
Batch applications	4.3	C/370, COBOL, LE/370, or PL/I runtime libraries
IMS applications	4.3	• IMS 3.1.0 • C/370, COBOL, LE/370, or PL/I runtime libraries
CICS V2 applications	4.3	• CICS 2.1.2 only • C/370, COBOL, LE/370, or PL/I runtime libraries
CICS V3 applications	4.3	• CICS 3.3.0 • C/370, COBOL, LE/370, or PL/I runtime libraries
Operations and controls panels	4.3	• ISPF/PDF 3.2.0 • TSO/E 2.0.0
MQSeries-IMS bridge	5.1	IMS 5.1 with the following APARs: • PN87811
Distributed queuing	4.3	See Table 29 on page 225
Distributed queuing (using CICS)	4.3	• CICS 3.2.1 (for local administration) • CICS 3.3.0 (for remote administration) • VTAM as needed by MVS

Notes:

1. Unless otherwise noted, any later level is acceptable.
2. The APARs listed might not apply to all versions of the prerequisite products.

Table 29. Additional requirements for distributed queuing

Runtime libraries	Other
LE/370 1.5 with the following APARs: • PN78163 • PN80015 • PN80739 • PN82174 • PN82557 • PN86618 • PQ03507	**LU 6.2 communications** • VTAM as needed by MVS **TCP/IP communications** Either: • TCP/IP 3.1 or 3.2 only with APAR PN85260 • Interlink SNS/TCPaccess 3.1 or later

Program numbers

- MQSeries for MVS/ESA (5695-137)
- MVS/ESA 4.3 or later (5695-047 (JES2), 5695-048 (JES3))
- MVS/ESA 5.1 or later (5655-068 (JES2), 5655-069 (JES3))
- SMP/E 1.8 (5668-949)
- DFSMS/MVS binder utility (5695-DF1)
- DFP 3.1 or later (5665-XA3)
- RACF 2.1 (5695-039)
- ISPF/PDF or later (5685-054)
- TSO/E 2.0 or later (5685-025)
- CICS/MVS 2.1.2 (5665-403)
- CICS/ESA 3.2.1 or later (5685-083)
- CICS/ESA 3.3 or later (5685-083)
- IMS/ESA 3.1 or later (5665-409)
- IMS/ESA 5.1 or later (5695-176)
- SAA AD/Cycle LE/370 (5688-198)
- ACF/VTAM 3.4.1 (5685-085)
- IBM TCP/IP 3.1 and 3.2 (5655-HAL)

Table 30. Compilers supported

Language	Compilers	Program number
Assembler	• Assembler H • IBM High level assembler MVS	• 5668-962 • 5696-234
COBOL	• VS COBOL II • IBM SAA AD/Cycle COBOL/370	• 5668-958 • 5688-197
C	• C/370 Release 2.1.0 (with APAR UN37741) • IBM SAA AD/Cycle C/370	• 5688-187 • 5688-216
PL/I	• OS PL/I Optimizing Compiler • SAA AD/Cycle PL/I Compiler	• 5668-910 • 5688-235

Clients

For MQSeries for MVS/ESA to support clients you need to install distributed queuing without CICS, using either SNA LU 6.2, SNS/TCPaccess, or TCP/IP, as described above. You also need the client/server support code that is provided by the Client Attachment Feature of MQSeries for MVS/ESA. However, you can administer clients without this feature.

Delivery

MQSeries for MVS/ESA is supplied on either 6250 tape or 3480 cartridge. (It is also available on 4-mm DAT tape for PC/390.) One tape or cartridge contains the product code together with four language features – US English (mixed case), US English (upper case), Japanese, and Simplified Chinese – and the distributed queuing facility. The second tape or cartridge contains the optional Client Attachment Feature.

Installation

MQSeries for MVS/ESA is installed with SMP/E using the receive-apply-accept approach. JCL is provided on the tape to assist with this process.

The installation can be performed in approximately two hours. Customization of the product is then required, the duration of this process being dependent on the individual requirements of the enterprise.

The *MQSeries for MVS/ESA Program Directory* contains specific instructions for installing MQSeries for MVS/ESA.

MQSeries for OS/2 Warp

MQSeries for OS/2 Warp, Version 5

Machine requirements

MQSeries for OS/2 Warp runs on any personal computer that is capable of running the required level of OS/2 Warp, and which has sufficient RAM and disk storage to meet the combined requirements of the programming prerequisites, MQSeries for OS/2 Warp, the access methods, and the application programs. The system unit must have a CD-ROM device.

An MQSeries client can run on any personal computer that is capable of running the client code and which has sufficient storage to meet the combined requirements of the programming prerequisites, the client code, access methods, and the application programs.

Software requirements

Software requirements are identical for server and client OS/2 Warp environments unless otherwise stated.

Minimum supported software levels are shown. Later levels, if any, are supported unless otherwise stated.

- OS/2 WARP, Version 4 or later Version 4.x
- OS/2 Warp Server, Version 4.0
- OS/2 Warp Server Advanced SMP feature, V4.0

Clients

Client code for AIX, DOS, HP-UX, OS/2 Warp, Sun Solaris, Windows NT, Windows 3.1, and Windows 95 workstations is distributed with the server code. The Windows 3.1 client can operate under Windows 3.1, Windows 95, or within the WIN-OS/2 environment under OS/2 Warp.

Client software provides a remote interface to an MQSeries server. Client support does not result in distributed coordination of units of work.

The MQSeries client for Java is also distributed with the server code; see the *MQSeries Clients* manual for more information.

Connectivity

- Communications Manager/2 for OS/2, Version 1.11 (this includes LU 6.2 and NetBIOS)
- IBM Communications Server, Version 4.0
- Novell NetWare Client for OS/2, Version 1.20 (for direct IPX/SPX support)
- TCP/IP for OS/2 Version 2.0 base kit plus NetBIOS kit

Options

- Transaction processing monitors (server only)
  - Transaction Server for OS/2, Version 4
- Databases (server only)
  - DB2 for OS/2, Version 2.1.1
- IBM Software Servers
  - Communications Server for OS/2 Warp, Version 4.0
  - Database Server for OS/2 Warp, Version 4.0
  - Directory Security Server for OS/2 Warp (DES version), Version 4.0
  - Directory Security Server for OS/2 Warp (CDMF version), Version 4.0
  - Internet Connection Server for OS/2 Warp, Version 4.2.1
  - Internet Connection Secure Server for OS/2 Warp, Version 4.2.1
  - Transaction Server for OS/2, Version 4.0
- DCE
  - IBM Directory and Security Server for OS/2 Warp, Version 4 or later compatible versions. This must be the U.S. Domestic version with DES encryption if running the MQSeries-supplied DCE send, receive, or message exits.

    If used as a DCE server, the following minimum system is recommended:

    - A Pentium processor running 90 MHz or faster
    - 64 MB or more of memory
    - OS/2 Warp Server V4.0 or later
  - MQSeries DCE names and security modules are provided as part of MQSeries for OS/2 Warp

Languages and compilers

- C using IBM C Set++ for OS/2, Version 2.1 (C bindings)
- C using Borland C++ Compiler, Version 2.0 (C bindings)
- C and C++ using IBM VisualAge for C++ for OS/2, Version 3.0
- COBOL using Micro Focus** COBOL, Version 4
- COBOL using IBM VisualAge for COBOL for OS/2, Version 1.1
- PL/I using IBM PL/I for OS/2 Version 1.2
- PL/I using IBM VisualAge for PL/I for OS/2

Delivery

MQSeries for OS/2 Warp is supplied on CD-ROM. Two CD-ROMs are supplied; one containing the MQSeries for OS/2 Warp server and client, and the other containing the other MQSeries clients shipped with MQSeries for OS/2 Warp.

Installation

MQSeries for OS/2 Warp is installed using Software Installer/2. The installation can be performed in approximately 15 minutes. Customization of the product is then required, the duration of this process being dependent on the individual requirements of the enterprise.

The *MQSeries for OS/2 Warp V5.0 Quick Beginnings* booklet contains specific instructions for installing MQSeries for OS/2 Warp. It also gives information about using Configuration, Installation, and Distribution (CID), and NetView Distribution Manager/2 (DM/2) to install MQSeries for OS/2 Warp.

MQSeries for SCO UNIX

IBM MQSeries for SCO UNIX Version 1 Release 4

Machine requirements

For the following configurations: combined file/communications server, communications server only, file server only, or client only:

- Any 386 DX PC or better
  - Minimum system memory: 16 MB
  - Minimum disk space:
    - Combined file/comms server: 2 MB + size of queues
    - File server: size of queues
    - Communications server: 2 MB
    - Client: normal disk space supplied with machine
- Any LAN adapter

For SNA connectivity (communications server):

- Any 486 PC or better (including an ISA bus)
  - Minimum system memory: 16 MB
  - Minimum disk space: 50 MB + size of queues (where appropriate)
- Apertus Technologies Inc ELC Adapter (ISA) with:
  - Express 2.1.1 (for SCO OpenServer Desktop 5.0.0b or SCO OpenServer Enterprise 5.0.0b or later 5.x)
  - Express 2.04b (for SCO Open Desktop 3.0 or later 3.x or SCO Open Server 3.0 or later 3.x)

or

- Emulex Adapter (ISA) with:
  - Express 2.1.1 (for SCO OpenServer Desktop 5.0.0b or SCO OpenServer Enterprise 5.0.0b or later 5.x)
  - Express 2.04b (for SCO Open Desktop 3.0 or later 3.x or SCO Open Server 3.0 or later 3.x)

or

- Madge Token Ring Adapter (ISA) with:
  - Madge Smart 16/4 AT PLUS card with Express 2.1.1 (for SCO OpenServer Desktop 5.0.0b or SCO OpenServer Enterprise 5.0.0b or later 5.x)
  - One of the following for SCO Open Desktop 3.0 or later 3.x or SCO Open Server 3.0 or later 3.x:

Madge card	Express Version
Madge Smart 16/4 AT card	2.04b or 2.1.1
Madge Smart 16/4 AT PLUS card	2.1.1

Software requirements

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

For the following configurations: combined file/communications server, communications server only, file server only, or client only:

- SCO Open Desktop Version 3.0 or later Version 3.x, including:
  - SCO Support Level Supplement UOD385A
  - TCP/IP

  or
- SCO Open Server Version 3.0 or later Version 3.x, including:
  - SCO Support Level Supplement UOD385A
  - TCP/IP
- SCO OpenServer Desktop 5.0.0b or later 5.x or SCO OpenServer Enterprise 5.0.0b or later 5.x, including:
  - TCP/IP
- For the file server and client configurations, appropriate LAN software, for example, NFS to match TCP/IP. (If you plan to use NFS, please contact your services representative to obtain any patches that you might need.)

Languages and compilers

Supported language for application development:

- A C compiler conformant with:

  ANSI programming language C, X3.159-1989
  ISO/IEC 9899:1990
  ISO/IEC 0045-1:1990

  such as SCO C.

Delivery

MQSeries for SCO UNIX is supplied on 3.5-inch diskettes.

Installation

MQSeries for SCO UNIX is installed using the custom utility.

The installation can be performed in approximately 15 minutes.

The *MQSeries for SCO UNIX User's Guide* contains specific instructions for installing this product.

MQSeries for SINIX and DC/OSx

IBM MQSeries for SINIX and DC/OSx V2.2

Machine requirements

SINIX: RM200, RM300, RM400, RM600 systems with minimum disk space of 30 MB. If DynaText books are installed, a minimum of 50 MB of system disk space is needed.

DC/OSx: MIServer, Nile systems with minimum disk space of 30 MB.

Software requirements

Software requirements are identical for server and client environments unless otherwise stated.

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

- SINIX operating system, for RM200, RM300, or RM400 – SINIX-N Version 5.42C10 or later
- SINIX operating system, for RM600 – SINIX-Y Version 5.42A40 or later
- DC/OSx operating system Version 1.1-cd079

MQSeries for SINIX and DC/OSx clients

Client code for SINIX and DC/OSx, OS/2 Warp, DOS, and Windows 3.1 workstations is distributed with the server code.

The Windows 3.1 client can operate under Windows 3.1, Windows 95 or within the WIN-OS/2 environment under OS/2 Warp.

Client software provides a remote interface to a LAN server. It can reside at the server or at a file server and be copied dynamically to the client for use, or it can reside on the client's disk space.

Client support does not result in distributed coordination of units of work.

Connectivity

The network protocols supported are SNA LU 6.2 and TCP/IP.

- SINIX: SNA

  TRANSIT-SERVER 3.4 (SNA Communication Server Version)
  TRANSIT-CLIENT 3.4 (SNA Comm. Client / Local Functions)
  TRANSIT-CPIC 3.4 (SNA LU 6.2 Communication and CPI-C)

- SINIX: OpenNet TCP/IP (shipped with base SINIX operating system)
- DC/OSx: TCP/IP Version 1.0

- DC/OSx: SNA requires LU 6.2 SW Version 1.3
  - To support the ISC-2 (Intelligent Synchronous Controller) serial line
    - Comm Services V 1.2 and ISC with SNA engine V 3.1
  - To support the ILC-T (Intelligent LAN Controller, Token ring) interface
    - Comm Services V 1.2 and Token Ring Mac interface V 1.3
  - To support the SNA on the ESCON IBM Channel link

    XVI/ESCON Driver 1.0

Options

- Transaction Processing monitors (coordination via X/Open XA interface)
  - SINIX: Encina (version 1.1A00)
  - SINIX: CICS for Siemens Nixdorf SINIX (version 2.1)
  - SINIX: Novell Tuxedo (Version 5.0)
  - DC/OSx: Novell Tuxedo (Version 5.0)
- DCE
  - SINIX: Version DCE-MI V1.03B00

  Note: DCE cannot be invoked from installable services or from user exits on the SINIX platform. However, stand-alone DCE programs can invoke the MQI.

Languages and compilers

- SINIX: C compiler (C-DS, MIPS) version 1.1
- DC/OSx: C4.0 compiler version 4.0.1
- SINIX: Micro Focus COBOL version 3.2
- DC/OSx: Micro Focus COBOL version 3.2

Delivery

- For SINIX, MQSeries is supplied on CD-ROM.
- For DC/OSx, MQSeries is supplied on QIC-320 cartridge tape.

Documentation browsing

On SINIX, to view the online documentation, DynaText is required. Either use SINIX/Windows version 2 or later, which provides a DynaText viewer, or use the SINIX online documentation package (separate product), which also provides a DynaText viewer.

Installation

MQSeries for SINIX is installed using the **sysadm** command.

MQSeries for DC/OSx is installed using the **pkgadd** command.

The base directory for installation is `/opt/mqm`.

MQSeries for SunOS

IBM MQSeries for SunOS Version 2.2

Machine requirements

Only the Sun SPARC hardware is supported. A minimum of 25 MB of system disk space is required.

Software requirements

Software requirements are identical for server and client on SunOS environments unless otherwise stated.

Minimum supported levels are shown. Later levels are supported unless otherwise stated.

- Sun SunOS UNIX Version 4.1.3

MQSeries for SunOS clients

Client code for SunOS, OS/2 Warp, DOS, and Windows 3.1 workstations is distributed with the server code.

The Windows 3.1 client can operate under Windows 3.1, Windows 95 or within the WIN-OS/2 environment under OS/2 Warp.

Client software provides a remote interface to a LAN server. It can reside at the server or at a file server and be copied dynamically to the client for use, or it can reside on the client's disk space.

Client support does not result in distributed coordination of units of work.

Connectivity

The network protocols supported are SNA LU 6.2 and TCP/IP.

- SunLink SNA Peer-to-Peer Version 7.0 or later
- TCP/IP as part of the base operating system

Options

- Transaction Processing monitors (coordination via X/Open XA interface)
  - For example, Novell Tuxedo V5.0

Languages and compilers

- SPARCompiler C 3.0.1
- Micro Focus COBOL Version 3.0

Delivery

MQSeries for SunOS is supplied on CD-ROM.

Installation

MQSeries for SunOS V2.2 is installed using the shell script `amqinst.sh`, which is provided in the root directory of the CD-ROM. The base directory for installation is /usr/mqm.

The *MQSeries for SunOS System Management Guide* contains specific instructions for installing this product. Before installing the product refer to the README to see the latest information about prerequisites.

MQSeries for Sun Solaris

IBM MQSeries for Sun Solaris, Version 5

Machine requirements

Only the Sun SPARC and Sun UltraSPARC hardware is supported. A minimum of 25 MB of system disk space is required.

Software requirements

Software requirements are identical for server and client on Solaris environments unless otherwise stated.

Minimum supported levels are shown. Later levels are supported unless otherwise stated.

- Sun Solaris, Version 2.5.1 or later 2.x. You need to apply the following patches to Version 2.5.1 if you want to use the MQSeries client for Java:
  - 103566-08
  - 103600-13
  - 103640-08

Clients

Client code for AIX, DOS, HP-UX, OS/2 Warp, Sun Solaris, Windows NT, Windows 3.1, and Windows 95 workstations is distributed with the server code. The Windows 3.1 client can operate under Windows 3.1, Windows 95, or within the WIN-OS/2 environment under OS/2 Warp.

Client software provides a remote interface to an MQSeries server. Client support does not result in distributed coordination of units of work.

The MQSeries client for Java is also distributed with the server code; see the *MQSeries Clients* manual for more information.

Connectivity

The network protocols supported are SNA LU 6.2 and TCP/IP.

- SunLink SNA Peer-to-Peer, Version 9.0 or later 9.x (with patches 102713, 102690, 102312, 102874).
- If a token ring is to be used, SunLink Token Ring Interface /SBus, Version 3.0.2 (with patch 102463).
- TCP/IP as part of the base operating system.

Options

- Transaction Processing monitors (server only)
  - IBM CICS for Solaris, Version 2.1.1
  - Transarc Encina, Version 2.5
  - BEA Tuxedo, Version 5.1 or Version 6.1
- Databases (server only)
  - Oracle 7, Release 7.3.2.3
  - DB2 for Solaris, Version 2.1.1

- IBM Software Servers
  - Internet Connection Server for Sun Solaris, Version 4.1
  - Internet Connection Secure Server for Sun Solaris, Version 4.1
- DCE
  - Transarc DCE, Version 1.1 and later compatible releases. This must be the U.S. Domestic version with DES encryption if running the MQSeries supplied DCE send, receive, or message exits.
  - MQSeries DCE names and security modules are provided as part of MQSeries for Sun Solaris.

Languages and compilers

- C using Sun SPARCompiler C, Version 4.0 and 4.2
- C++ using Sun SPARCompiler C++, Version 4.1 and 4.2
- COBOL using Micro Focus COBOL for UNIX, Version 4.0

Delivery

MQSeries for Sun Solaris is supplied on CD-ROM. Two CD-ROMs are supplied; one containing the MQSeries for Sun Solaris server and client, and the other containing the other MQSeries clients shipped with MQSeries for Sun Solaris.

Installation

MQSeries for Sun Solaris is installed using the **pkgadd** command or the **sysadm** command. The *MQSeries for Sun Solaris V5.0 Quick Beginnings* contains specific instructions for installing this product.

MQSeries for Tandem NSK

IBM MQSeries for Tandem NSK Version 2 Release 2

Machine requirements

Minimum hardware requirements are:

- Any of the Tandem NSK range of machines supported by Tandem NSK D3x, D4x, or G02
- 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:

- Tandem NSK D3x, D4x, or G02 operating systems, including TM/MP (TMF), ENSCRIBE, and EMS
- TS/MP (PATHWAY) to match the operating system
- SCF for configuration, command, and control of TCP and SNA network transports

Transaction logging is maintained with the Tandem TM/MP (TMF) product.

Connectivity

For SNA connectivity, either:

- SNAX/APC and SNAX/XF or SNAX/APN to match the operating system
- Insession ICE to match the operating system

For TCP/IP connectivity:

- TCP/IP to match the operating system

Clients

MQSeries for Tandem NSK supports but does not ship or configure MQSeries clients.

Languages and compilers

Supported languages for application development:

- TAL
- C
- COBOL-85

Delivery

MQSeries for Tandem NSK is supplied on a 3480 cartridge.

Installation

The *MQSeries for Tandem NonStop Kernel System Management Guide* contains specific instructions for installing this product.

MQSeries Three Tier for AIX

IBM MQSeries Three Tier for AIX Version 1.0

MQSeries Three Tier for AIX allows you to:

- Run application servers on AIX
- Build applications on AIX

You must run your application clients on Windows 3.1 (or later) or OS/2.

Machine requirements

MQSeries Three Tier for AIX runs on any machine that can run the required software. The exact machine specifications depend on the level of performance you can accept.

Software requirements

The software you require depends on the tasks you want to perform using MQSeries Three Tier for AIX:

- Server
  - IBM AIX Version 3.2.5
  - IBM MQSeries for AIX Version 2.0 with PTF U438737, or Version 2.2
  - IBM SNA Services for AIX
- Application development
  - IBM AIX Version 3.2.5
  - IBM MQSeries for AIX Version 2.0 or later
  - A compiler

Languages and compilers

IBM C Set++ for AIX/6000, Version 2.1 (program number 5765-186)

Delivery

MQSeries Three Tier for AIX is supplied on CD-ROM or 8mm tape.

Installation

MQSeries Three Tier for AIX provides an installation utility. Installation takes approximately 15 minutes.

For descriptions of the components you can choose to install, and for instructions on installing, see the *MQSeries Three Tier Administration Guide*, SC33-1451.

MQSeries Three Tier for OS/2

IBM MQSeries Three Tier for OS/2 Version 1.0

MQSeries Three Tier for OS/2 allows you to:

- Run application clients on Windows 3.1 (or later) and OS/2
- Run application servers on OS/2
- Build applications on Windows 3.1 (or later) and OS/2

Machine requirements

MQSeries Three Tier for OS/2 runs on any machine that can run the required software. The exact machine specifications depend on the level of performance you can accept. Table 31 indicates some typical machine configurations for MQSeries Three Tier for OS/2 server, client, and developer's workstations. These are not intended to be minimum configurations.

Table 31. Typical MQSeries Three Tier for OS/2

Configuration	Processor	RAM	Video	Hard disk
Three Tier client (OS/2 or Windows 3.1)	486 25MHz CPU or better	16 MB	14x monitor	120 MB
Three Tier server (OS/2)	486DX4 100MHz CPU or better	32 MB	14x monitor	540 MB
Programming	Pentium 66MHz CPU or better	32-64 MB	17x monitor, 2 MB video RAM	760-1520 MB

Note: The recommended minimum configuration for developing applications with VisualAge is 24 MB RAM with 30 MB available for the swapper file.

Software requirements

The software you require depends on the tasks you want to perform using MQSeries Three Tier for OS/2:

- OS/2 client
  - IBM OS/2 Version 2.1 or later
  - IBM MQSeries for OS/2 Version 2.0 with PTF U200051, or later
  - MQSeries communication support:
    - IBM Communication Manager/2 Version 1.1 for OS/2 (this includes LU 6.2 and NetBIOS)
    - TCP/IP for OS/2 Version 2.0
- Windows 3.1 client
  - DOS Version 5 or later
  - Microsoft Windows 3.1
  - IBM MQSeries for OS/2 Version 2.0 with PTF U200051 (Windows client), or later

  - For communication, one of the following:
    - TCP/IP for OS/2 Version 2.0
    - LU 6.2
    - NetBIOS

- OS/2 server
  - IBM OS/2 Version 2.1 or later
  - IBM MQSeries for OS/2 Version 2.0 with PTF U200051, or later
  - MQSeries communication support:
    - IBM Communication Manager/2 Version 1.1 for OS/2 (this includes LU 6.2 and NetBIOS)
    - TCP/IP for OS/2 Version 2.0

- Application development on OS/2
  - IBM OS/2 Version 2.1 or later
  - IBM MQSeries for OS/2 Version 2.0 with PTF U200051, or later
  - IBM VisualAge Version 2.0 for OS/2 Team Version (optional)
  - A compiler

- Application development on Windows 3.1
  - DOS Version 5.0 or later
  - Microsoft Windows 3.1
  - IBM OS/2 Version 2.1 or later
  - IBM MQSeries for OS/2 Version 2.0 with PTF U200051, or later

Languages and compilers

C language	For OS/2, IBM C Set++ for OS/2, Version 2.0
	For Windows 3.1, the Microsoft Visual C++ Development System, Version 1.5
COBOL	IBM COBOL VisualSet for OS/2
	Micro Focus** 32-bit COBOL Software Development Kit for OS/2
PL/I	The IBM PL/I Package/2 V1 R2

Delivery

MQSeries Three Tier for OS/2 is supplied on CD-ROM and 3.5-inch diskettes.

Installation

MQSeries Three Tier for OS/2 provides an installation utility. Installation takes approximately 15 minutes.

You can install MQSeries Three Tier for OS/2 from a LAN.

For descriptions of the components you can choose to install, and for instructions on installing, see the *MQSeries Three Tier Administration Guide*, SC33-1451.

MQSeries for UnixWare

IBM MQSeries for UnixWare Version 1 Release 4.1

Machine requirements

For the following configurations: combined file/communications server, communications server only, file server only, or client only:

- Any 386 DX PC or better
  - Minimum system memory: 16 MB
  - Minimum disk space:
    - Combined file/comms server: 2 MB + size of queues
    - File server: size of queues
    - Communications server: 2 MB
    - Client: normal disk space supplied with machine
- Any LAN adapter

For SNA connectivity (communications server):

- Any 486 PC or better (including an ISA bus)
  - Minimum system memory: 16 MB
  - Minimum disk space: 28.5 MB + size of queues (where appropriate)
- Apertus Technologies Inc ELC Adapter (ISA) with:
  - Express 2.04b or 2.1.1 (for UnixWare Application Server SDK 1.1 or later 1.x)
  - Express 2.1.2 (for UnixWare Personal Edition 2.01 or later 2.x or Application Server 2.01 or 2.1)

or

- Emulex Adapter (ISA) with:
  - Express 2.04b or 2.1.1 (for UnixWare Application Server SDK 1.1 or later 1.x)
  - Express 2.1.2 (for UnixWare Personal Edition 2.01 or later 2.x or Application Server 2.01 or 2.1)

or

- Madge Token Ring Adapter (ISA) with:
  - Madge Smart 16/4 AT PLUS card with Express 2.1.2 (for UnixWare Personal Edition 2.01 or later 2.x or Application Server 2.01 or 2.1)
  - One of the following for UnixWare Application Server SDK 1.1 or later 1.x:

Madge card	Express Version
Madge Smart 16/4 AT card	2.04b or 2.1.1
Madge Smart 16/4 AT PLUS card	2.1.2

Software requirements

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

For the following configurations: combined file/communications server, communications server only, file server only, or client only:

- UnixWare Application Server SDK Version 1.1 or later Version 1, including TCP/IP. (For bridge support across TCP/IP, contact your Novell service representative.)
- UnixWare Personal Edition 2.01 or later 2.x or Application Server 2.01 or 2.1 including TCP/IP.
- Appropriate LAN software (file server or client): for example, NFS to match TCP/IP. Customers who plan to use NFS should contact their service representative to obtain all available patches.

Languages and compilers

Supported language for application development:

- C

Delivery

MQSeries for UnixWare is supplied on 3.5-inch diskettes.

Installation

MQSeries for UnixWare is installed using the **pkgadd** command.

The installation can be performed in approximately 10 minutes.

The *MQSeries for UnixWare User's Guide* contains specific instructions for installing this product.

MQSeries client for VM/ESA

This section summarizes the machine and software requirements for the MQSeries client for VM/ESA.

Machine requirements

An MQSeries client can run under VM/ESA on any of the following machines:

- S/390 Parallel Enterprise Server - Generation 3
- S/390 Multiprise 2000
- ES/9000 Processors

Software requirements

The following are prerequisites for MQSeries applications running on a VM/ESA client.

Minimum supported software levels are shown. Later levels, if any, are supported unless otherwise stated.

- VM/ESA Version 2.3
- LE/370 Release 1.6
- TCP/IP Release 2.0 or VTAM LU 6.2

Languages and compilers

The following compilers are supported:

- IBM C for VM Release 3.1
- IBM VS COBOL II
- IBM OS/PL/I Release 2.3
- IBM VM/ESA REXX/VM
- IBM Assembler

Delivery

The MQSeries client for VM/ESA is shipped with the VM/ESA product. It is preloaded with the installation of CMS, and can be accessed on the `MAINT 193` disk.

MQSeries for VSE/ESA

IBM MQSeries for VSE/ESA Version 1 Release 4

Machine requirements

- Any ESA/390 system capable supporting the required level of VSE/ESA
  - Minimum system memory = normal memory as supplied with machine
  - Minimum system DASD = 2 cylinders (3390) + size of queues
- Any communications hardware supporting SNA LU 6.2.

Software requirements

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

Either:

- VSE/ESA V1.4 in ESA mode, with ACF/VTAM for VSE/ESA V3.4

  or

- VSE/ESA V2.1 with ACF/VTAM for VSE/ESA V4.2

With

- CICS/VSE V 2.3
- IBM Language environment (LE) for VSE runtime library with IBM COBOL for VSE

Languages and compilers

Supported language for application development:

- IBM COBOL for VSE (IBM COBOL for VSE is fully upward compatible with VS COBOL II).

Delivery

MQSeries for VSE/ESA is supplied on 4mm tape and 3480 tape cartridge.

Installation

MQSeries for VSE/ESA is installed using the VSE Maintenance System History Program (MSHP).

The *MQSeries for VSE/ESA User's Guide* contains specific instructions for installing this product.

MQSeries for Windows Version 2.0

IBM MQSeries for Windows Version 2.0

Machine requirements

MQSeries for Windows version 2.0 is a 16-bit product, so it runs on computers that run Windows 3.1, and it runs in 16-bit compatibility mode on Windows 95. Table 32 suggests two configurations: one for running applications and the other for developing applications.

Note: These recommendations are for guidance only. They do not take into account the effects of any other software that might be running on the system at the same time.

Table 32. Suggested hardware configurations for MQSeries for Windows V2.0

Configuration	Processor	RAM	Hard disk
For running applications	386 16 MHz	4–8 MB	3.9 MB available
For developing applications	486 66 MHz or better	8–16 MB	5 MB available
Note: The specification for developing applications does not include hardware requirements for other development tools (for example, compilers).			

Software requirements

This section describes the software you require before you can use MQSeries for Windows version 2.0. This depends on whether you want to run MQSeries applications on MQSeries for Windows, or develop your own applications for it.

For running MQSeries applications

For running applications on MQSeries for Windows V2.0, you need the following software (or later versions):

- MS-DOS or PC DOS Version 3.3
- Microsoft Windows 3.1,
  or Windows 95,
  or Windows for Workgroups 3.11,
  or Win-OS/2 on OS/2 Version 3.0 (Warp)

  MQSeries for Windows V2.0 runs in 16-bit compatibility mode on Windows 95.

Connectivity

- TCP/IP for the operating system you are using:
  - For Microsoft Windows 3.1, you need IBM TCP/IP for DOS V2.1.1 with CSD 2.1.1.4.
  - For Microsoft Windows 95, use the version of TCP/IP supplied with Windows 95.
  - For Microsoft Windows for Workgroups 3.11, you need IBM TCP/IP for DOS V2.1.1 with CSD 2.1.1.4.
  - For Win-OS/2, you need IBM TCP/IP for OS/2 Version 2.

Languages and compilers

To develop and test MQSeries applications that run on Windows, ***in addition to the software listed in "For running MQSeries applications" on page 248*** , you need only the compiler for the programming language you will use:

For 16-bit C
Microsoft Visual C++ Version 1.5

For 32-bit C
Microsoft Visual C++ Version 2.0

For 16-bit BASIC
Microsoft Visual Basic Version 3.0
or Microsoft Visual Basic Version 4.0

For 32-bit BASIC
Microsoft Visual Basic Version 4.0

MQSeries for Windows V2.0 runs in 16-bit compatibility mode on Windows 95, but you can write 32-bit MQSeries for Windows applications.

Delivery

MQSeries for Windows Version 2.0 is supplied on diskettes. It is enabled for Configuration, Installation, and Distribution (CID) so you can put the installation files on a LAN server for easier access.

Installation

The *MQSeries for Windows User's Guide* contains specific instructions for installing this product.

MQSeries for Windows Version 2.1

IBM MQSeries for Windows Version 2.1

Machine requirements

MQSeries for Windows version 2.1 is a 32-bit product, so it runs on computers that run Windows 95 or Windows NT Version 4.0. Table 33 suggests two configurations: one for running applications and the other for developing applications.

Note: These recommendations are for guidance only. They do not take into account the effects of any other software that might be running on the system at the same time.

Table 33. Suggested hardware configurations for MQSeries for Windows V2.1

Configuration	Processor	RAM	Hard disk
For running applications	386DX or better	At least 4 MB	At least 3.5 MB available
For developing applications	486 66 MHz or better	At least 8 MB	At least 5 MB available

Note: The specification for developing applications does not include hardware requirements for other development tools (for example, compilers).

Software requirements

This section describes the software you require before you can use MQSeries for Windows version 2.1. This depends on whether you want to run MQSeries applications on MQSeries for Windows, or develop your own applications for it.

For running MQSeries applications

For running applications on MQSeries for Windows V2.1, you need the following software (or later versions):

- Microsoft Windows 95 or Windows NT Version 4.0

Connectivity

- TCP/IP for the operating system you are using.

Languages and compilers

To develop and test MQSeries applications that run on Windows, ***in addition to the software listed in "For running MQSeries applications"*** , you need only the compiler for the programming language you will use:

- Microsoft Visual C++ Version 4.0
- Borland C
- Microsoft Visual Basic Version 4.0

Delivery

MQSeries for Windows Version 2.1 is supplied on CD-ROM or diskettes. It is enabled for remote (or silent) installation so you can put the installation files on a LAN server for easier access.

Installation

The *MQSeries for Windows User's Guide* contains specific instructions for installing this product.

MQSeries for Windows NT

IBM MQSeries for Windows NT, Version 5

Machine requirements

MQSeries for Windows NT can run on any Intel 486 (or above) processor-based IBM PC machine or compatible.

Software requirements

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

- Microsoft Windows NT, Version 3.5.1, with Service Pack 5 installed to include TCP/IP, NetBIOS, and SPX
- Microsoft Windows NT, Version 4.0 or later Version 4.x

Clients

Client code for AIX, DOS, HP-UX, OS/2 Warp, Sun Solaris, Windows NT, Windows 3.1, and Windows 95 workstations is distributed with the server code. The Windows 3.1 client can operate under Windows 3.1, Windows 95, or within the WIN-OS/2 environment under OS/2 Warp.

Client software provides a remote interface to an MQSeries server. Client support does not result in distributed coordination of units of work.

The MQSeries client for Java is also distributed with the server code; see the *MQSeries Clients* manual for more information.

Connectivity

The network protocols supported are SNA LU 6.2, TCP/IP, NetBIOS, and SPX.

- SNA LU 6.2
  - Attachmate EXTRA! Personal Client, Version 6.1 and 6.2
  - IBM Communications Server for Windows NT, Version 5.0
  - IBM Personal Communications for Windows NT, Version 4.0
  - Microsoft SNA Server, Version 2.11 and Version 3
- For TCP/IP use the TCP/IP facilities within Windows NT
- For NetBIOS use the NetBIOS facilities within Windows NT
- For SPX use the SPX facilities within Windows NT
- For clients only:
  - OnNet SDK for Windows
  - FTP Software
  - Inc

Options

- Transaction Processing monitors (server only)
  - Transaction Server for Windows NT, Version 4.0
  - BEA TUXEDO, Version 5.1 or Version 6.1
- Databases
  - DB2 for Windows NT and Windows 95, Version 2.1.1
- IBM Software Servers
  - Communications Server for Windows NT, Version 5.0
  - Database Server for Windows NT, Version 4.0
  - Directory Security Server for Windows NT, Version 5.0
  - Internet Connection Server for Windows NT, Version 4.2.1
  - Internet Connection Secure Server for Windows NT, Version 4.2.1
  - Transaction Server for Windows NT, Version 4.0
- DCE
  - IBM Directory and Security Server for Windows NT, Version 5. This must be the U.S. Domestic version with DES encryption if running the MQSeries-supplied DCE send, receive, or message exits.
  - MQSeries DCE security modules are provided as part of MQSeries for Windows NT.

Languages and compilers

- C and C++ using Microsoft Visual C++ for Windows 95 and NT, Version 4.0
- C and C++ using IBM VisualAge for C++ for Windows, Version 3.5
- COBOL using Micro Focus** COBOL for Windows NT, Version 3.3 or Version 4
- IBM VisualAge COBOL for Windows NT, Version 2.1
- IBM PL/I for Windows, Version 1.2
- IBM VisualAge for PL/I for Windows

Delivery

MQSeries for Windows NT is supplied on CD-ROM. Two CD-ROMs are supplied; one containing the MQSeries for Windows NT server and client, and the other containing the other MQSeries clients shipped with MQSeries for Windows NT.

Installation

MQSeries for Windows NT is installed using the **setup** utility.

The installation can be performed in approximately 15 minutes. The actual time taken depends on several factors, including the following:

- The speed of the machine
- Which functions you are installing
- Where you are installing from

Customization of the product is then required, the duration of this process being dependent on the individual requirements of the enterprise.

The *MQSeries for Windows NT V5.0 Quick Beginnings* manual contains specific instructions for installing this product.

MQSeries client for Windows 3.1

This section summarizes the machine and software requirements for the Windows 3.1 client.

Machine requirements

An MQSeries client can run under Windows 3.1 on any personal computer that is capable of running the client code and which has sufficient storage to meet the combined requirements of the programming prerequisites, the client code, access methods, and the application programs.

Software requirements

The following are prerequisites for MQSeries applications running on a Windows 3.1 client.

Minimum supported software levels are shown. Later levels, if any, are supported unless otherwise stated.

Workstation clients: Client code for Windows 3.1 workstations is distributed with the server code for all servers except OS/400 and MVS/ESA.

- Windows 3.1

Options

- TCP/IP for OS/2 V2.0. The base kit is necessary. The DOS access kit allows clients access to TCP/IP via programs that run from WIN-OS2.
- TCP/IP V2.1.1 for DOS.
- IBM NetBIOS V2.1.1 for DOS/Windows.
- Novell IPX, using Novell NetBIOS emulation.

Languages and compilers

The following compilers are supported:

- C using Microsoft C/C++ Version 7.0
- C using Microsoft Visual C++ for Windows Version 2.0
- C++ using Microsoft Visual C++ Version 1.5
- COBOL using Micro Focus Visual COBOL for Windows Version 3.3

MQSeries client for Windows 95

This section summarizes the machine and software requirements for the Windows 95 client.

Machine requirements

An MQSeries client can run under Windows 95 on any personal computer that is capable of running the client code and which has sufficient storage to meet the combined requirements of the programming prerequisites, the client code, access methods, and the application programs.

Software requirements

The following are prerequisites for MQSeries applications running on a Windows 95 client.

Minimum supported software levels are shown. Later levels, if any, are supported unless otherwise stated.

Workstation clients

Client code for Windows 95 workstations is distributed with the server code for all servers except OS/400 and MVS/ESA.

- Windows 95

Connectivity

TCP/IP, SPX, and NetBIOS are all provided in the operating system.

Languages and compilers: The following compilers are supported:

- C using Microsoft Visual C++ for Windows 95/NT Version 2.0
- C++ using Microsoft Visual C++ Version 2.0
- C++ using IBM VisualAge C++ Version 3.5
- COBOL using Micro Focus COBOL Workbench Version 4.0

MQSeries link for R/3

This section tells you about the hardware and software you need to run MQSeries link for R/3.

Hardware requirements

To install and run this product, you need about 5 MB of available hard disk space. There are no additional hardware requirements except for those listed for MQSeries and R/3 on the platform you are using. Please consult the installation and planning sections in the appropriate books for these products.

Software requirements

To run the R/3 link, you must have installed the following:

- At least one R/3 system, version 3.0E or later. You can use version 3.0D, if you apply upgrade BINK090538. This fix will be supplied by SAP AG; details of how to get the fix are supplied with your R/3 system.

  Note: The fix adds MQSeries Options to the Destination Menu if you specify SAP administration function sm59.

- At least one instance of MQSeries Version 2.2.1 or later for the chosen platform (Version 2.0 for Windows NT).
- The relevant MQSeries link for R/3 product for the chosen platform.

Restrictions

The maximum size of MQSeries messages (4 MB) limits the amount of R/3 IDoc data that can be sent in a single message. Because the R/3 link header fields are part of the space that is assigned to MQSeries messages, the actual maximum amount of space available for R/3 transaction data is less than the full 4 MB. R/3 transaction data can consist of one IDoc or batch IDocs.

SAP AG recommends a maximum of 2 MB for each IDoc.

Supported platforms

MQSeries link for R/3 is available for the following release levels of these platforms:

- AIX 4.1
- HP-UX 10.01
- Sun Solaris 2.5
- Windows NT 3.5.1

MQSeries platforms

Using R/3 link, you can connect your R/3 system to other platforms that run MQSeries.

Part 10. Appendix

Appendix

Appendix. Notices

The following paragraph does not apply to any 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 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.

References in this publication to IBM products, programs, or services do not imply that IBM intends to make these available in all countries in which IBM operates. 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. Subject to IBM’s valid intellectual property or other legally protectable rights, any functionally equivalent product, program, or service may be used instead of the IBM product, program, or service. The evaluation and verification of operation in conjunction with other products, except those expressly designated by IBM, are the responsibility of the user.

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 Laboratory Counsel, MP151, IBM United Kingdom Laboratories, 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.

IBM may have patents or pending patent applications covering subject matter in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to the IBM Director of Licensing, IBM Corporation, 500 Columbus Avenue, Thornwood, New York 10594, U.S.A.

Trademarks

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

ACF/VTAM	AD/Cycle	AIX
AIX/6000	AIXwindows	AS/400
AT	BookManager	C/370
C/400	CICS	CICS/MVS
CICS/VSE	CICS/400	COBOL/370
COBOL/400	CT	DB2
HACMP/6000	IBM	ILE
IMS	IMS/ESA	MQ
MQSeries	MQSeries Three Tier	MVS/ESA
NetView	Open Blueprint	Operating System/400
OS/2 Warp	OS/400	POWERparallel
POWERserver	POWERstation	Presentation Manager
PS/2	RACF	RPG/400
RMF	RS/6000	SAA
S/390	System/370	System/390
VisualAge	VM/ESA	VSE/ESA
VTAM	WIN-OS/2	

PC Direct is a trademark of Ziff Communications Company and is used by IBM Corporation under license.

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

C-bus is a trademark of Corollary, Inc.

Microsoft, Windows, Windows NT, and the Windows logo are registered trademarks of Microsoft Corporation.

Java and HotJava are trademarks of Sun Microsystems, Inc.

Other company, product, and service names, which may be denoted by a double asterisk (**), may be trademarks or service marks of others.

Part 11. Glossary and index

Glossary of terms and abbreviations

This glossary defines MQSeries terms and abbreviations used in this book. If you do not find the term you are looking for, 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 may 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

active log. See *recovery log*.

adapter. An interface between MQSeries for MVS/ESA and TSO, IMS, CICS, or batch address spaces. An adapter is an attachment facility that enables applications to access MQSeries services.

add-in task. A function provided by MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT that coordinates the passing of data between a Lotus Notes application and an MQSeries application.

address space. The area of virtual storage available for a particular job.

administrator commands. MQSeries commands used to manage MQSeries objects, such as queues, processes, and namelists.

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.

application environment. The software facilities that are accessible by an application program. On the MVS platform, CICS and IMS are examples of application environments.

archive log. See *recovery log*.

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 service. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, a service that provides authority checking of commands and MQI calls for the user identifier associated with the command or call.

authorized program analysis report (APAR). A report of a problem caused by a suspected defect in a current, unaltered release of a program.

B

backout. 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. Contrast with *commit*.

bootstrap data set (BSDS). A VSAM data set that contains:

- An inventory of all active and archived log data sets known to MQSeries for MVS/ESA
- A wrap-around inventory of all recent MQSeries for MVS/ESA activity

The BSDS is required if the MQSeries for MVS/ESA subsystem has to be restarted.

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.

BSDS. Bootstrap data set.

buffer pool. An area of main storage used for MQSeries for MVS/ESA queues, messages, and object definitions. See also *page set*.

C

CCSID. Coded character set identifier.

CDF. Channel definition file.

channel. See *message channel*.

channel definition file (CDF). In MQSeries, a file containing communication channel definitions that associate transmission queues with communication links.

channel event. An event indicating that a channel instance has become available or unavailable. Channel events are generated on the queue managers at both ends of the channel.

checkpoint. (1) A time when significant information is written on the log. Contrast with *syncpoint*. (2) 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.

CI. Control interval.

circular logging. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, the process of keeping all restart data in a ring of log files. Logging fills the first file in the ring and then moves on to the next, until all the files are full. At this point, logging goes back to the first file in the ring and starts again, if the space has been freed or is no longer needed. Circular logging is used during restart recovery, using the log to roll back transactions that were in progress when the system stopped. Contrast with *linear logging*.

CL. Control Language.

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 *MQSeries 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.

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 server. The MQSeries component that reads commands from the system-command input queue, verifies them, and passes valid commands to the command processor.

commit. An operation that applies 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. Contrast with *backout*.

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.

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.

context. Information about the origin of a message.

context security. In MQSeries, a method of allowing security to be handled such that messages are obliged to carry details of their origins in the message descriptor.

control command. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, a command that can be entered interactively from the operating system command line. Such a command requires only that the MQSeries product be installed; it does not require a special utility or program to run it.

control interval (CI). A fixed-length area of direct access storage in which VSAM stores records and creates distributed free spaces. The control interval is the unit of information that VSAM transmits to or from direct access storage.

Control Language (CL). In MQSeries for AS/400, a language that can be used to issue commands, either at the command line or by writing a CL program.

CRE. Common Run-Time Environment.

Cross Systems Coupling Facility (XCF). Provides the MVS coupling services that allow authorized programs in a multisystem environment to communicate with programs on the same or different MVS systems.

D

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.

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.

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 some basic services, making the development of distributed applications easier. DCE is defined by the Open Software Foundation (OSF).

distributed queue management (DQM). In message queuing, the setup and control of message channels to queue managers on other systems.

DLQ. Dead-letter queue.

DQM. Distributed queue management.

dual logging. A method of recording MQSeries for MVS/ESA activity, where each change is recorded on two data sets, so that if a restart is necessary and one data set is unreadable, the other can be used. Contrast with *single logging*.

dual mode. See *dual logging*.

dynamic queue. A local queue created when a program opens a model queue object. See also *permanent dynamic queue* and *temporary dynamic queue*.

E

environment. See *application environment*.

ESM. External security manager.

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 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.

external security manager (ESM). A security product that is invoked by the MVS System Authorization Facility. RACF is an example of an ESM.

F

FIFO. First-in-first-out.

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. (A)

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

Generalized Trace Facility (GTF). An MVS service program that records significant system events, such as supervisor calls and start I/O operations, for the purpose of problem determination.

get. In message queuing, to use the MQGET call to remove a message from a queue. See also *browse*.

GTF. Generalized Trace Facility.

I

ICE. Intersystem Communications Environment is a family of Tandem-based software products that enables you to access a variety of applications on Tandem computers.

in-doubt unit of recovery. In MQSeries for MVS/ESA, the status of a unit of recovery for which a syncpoint has been requested but not yet performed.

initialization input data sets. Data sets used by MQSeries for MVS/ESA when it starts up.

initiation queue. A local queue on which the queue manager puts trigger messages.

installable services. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, 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.

Interactive System Productivity Facility (ISPF). An IBM licensed program that serves as a full-screen editor and dialog manager. It is used for writing application programs, and provides a means of generating standard screen panels and interactive dialogues between the application programmer and terminal user.

ISPF. Interactive System Productivity Facility.

L

linear logging. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, 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 for incoming network connections.

local definition. An MQSeries object belonging to a local queue manager.

local definition of a remote queue. An MQSeries object belonging 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, a file recording the work done by queue managers while they receive, transmit, and deliver messages.

log control file. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, the file containing information needed to monitor the use of log files (for example, their size and location, and the name of the next available file).

log file. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, a file in which all significant changes to the data controlled by a queue manager are recorded. If the primary log files become full, MQSeries allocates secondary log files.

logical unit of work (LUW). See *unit of work*.

M

MCA. Message channel agent.

MCI. Message channel interface.

message. (1) In message queuing applications, a communication sent between programs. See also *persistent message* and *nonpersistent message*. (2) 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 and a receiver) 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.

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 allows application programs to access message queuing services.

message queuing. A programming technique in which each program within an application communicates with the other programs by putting messages on queues.

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.

MQI. Message queue interface.

MQI channel. Connects an MQSeries client to a queue manager on a server system, and transfers only MQI calls and responses in a bidirectional manner. Contrast with *message channel*.

MQSC. MQSeries commands.

MQSeries. A family of IBM licensed programs that provides message queuing services.

MQSeries client. Part of an MQSeries product that can be installed on a system without installing the full queue manager. The MQSeries client accepts MQI calls from applications and communicates with a queue manager on a server system.

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 for MVS/ESA object that contains a list of queue names.

name service. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, the facility that determines which queue manager owns a specified queue.

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.

nonpersistent message. A message that does not survive a restart of the queue manager. Contrast with *persistent message*.

NSI. Name service interface.

O

OAM. Object authority manager.

object. In MQSeries, an object is a queue manager, a queue, a process definition, a channel, a namelist (MVS/ESA only), or a storage class (MVS/ESA only).

object authority manager (OAM). In MQSeries on UNIX systems and MQSeries for Windows NT, 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.

Open Transaction Manager Access (OTMA). A transaction-based, connectionless client/server protocol. It functions as an interface for host-based communications servers accessing IMS TM applications through the MVS Cross Systems Coupling Facility (XCF). OTMA is implemented in an MVS sysplex environment. Therefore, the domain of OTMA is restricted to the domain of XCF.

OTMA. Open Transaction Manager Access.

output log-buffer. In MQSeries for MVS/ESA, a buffer that holds recovery log records before they are written to the archive log.

P

page set. A VSAM data set used when MQSeries for MVS/ESA moves data (for example, queues and messages) from buffers in main storage to permanent backing storage (DASD).

PCF. Programmable command format.

PCF command. See *programmable command format*.

performance event. A category of event indicating that a limit condition has occurred.

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 or a TCP/IP connection is functioning.

platform. In MQSeries, the operating system under which a queue manager is running.

point of recovery. In MQSeries for MVS/ESA, the term used to describe a set of backup copies of MQSeries for MVS/ESA page sets and the corresponding log data sets required to recover these page sets. These backup copies provide a potential restart point in the event of page set loss (for example, page set I/O error).

principal. In MQSeries on UNIX systems, MQSeries for OS/2 Warp, and MQSeries for Windows NT, a term used for a user identifier. Used by the object authority manager for checking authorizations to system resources.

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 used by:

- User administration applications, to 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. (1) 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*. (2) An MQSeries object that defines the attributes of a particular queue manager.

queuing. See *message queuing*.

R

RBA. Relative byte address.

reason code. A return code that describes the reason for the failure or partial success of an MQI call.

recovery log. In MQSeries for MVS/ESA, data sets containing information needed to recover messages, queues, and the MQSeries subsystem. MQSeries for MVS/ESA writes each record to a data set called the *active log*. When the active log is full, its contents are off-loaded to a DASD or tape data set called the *archive log*. Synonymous with *log*.

relative byte address (RBA). The displacement in bytes of a stored record or control interval from the beginning of the storage space allocated to the data set to which it belongs.

remote queue. A queue belonging 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 that is not the one 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.

request message. A type of message used to request a reply from another program.

RESLEVEL. In MQSeries for MVS/ESA, an option that controls the number of CICS user IDs checked for API-resource security in MQSeries for MVS/ESA.

resource. Any facility of the computing system or operating system required by a job or task. In MQSeries for MVS/ESA, examples of resources are buffer pools, page sets, log data sets, queues, and messages.

resource manager. An application, program, or transaction that manages and controls access to shared resources such as memory buffers and data sets. MQSeries, CICS, and IMS are resource managers.

return codes. The collective name for completion codes and reason codes.

rollback. Synonym for *back out*.

S

SAF. System Authorization Facility.

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.

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*.

signaling. In MQSeries for MVS/ESA and MQSeries for Windows 2.1, a feature that allows the operating system to notify a program when an expected message arrives on a queue.

single logging. A method of recording MQSeries for MVS/ESA activity where each change is recorded on one data set only. Contrast with *dual logging*.

single-phase backout. A method in which an action 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*.

storage class. In MQSeries for MVS/ESA, a storage class defines the page set that is to hold the messages for a particular queue. The storage class is specified when the queue is defined.

subsystem. In MVS, a group of modules that provides function that is dependent on MVS. For example, MQSeries for MVS/ESA is an MVS subsystem.

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 Authorization Facility (SAF). An MVS facility through which MQSeries for MVS/ESA communicates with an external security manager such as RACF.

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

TACL. Tandem Advanced Command Language.

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.

trace. In MQSeries, a facility for recording MQSeries activity. The destinations for trace entries can include GTF and the system management facility (SMF). See also *global trace* and *performance trace*.

transmission program. See *message channel agent*.

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 allowing a queue manager to start an application automatically when predetermined conditions on a queue are satisfied.

trigger message. A message containing information about the program that a trigger monitor is to start.

trigger monitor. A continuously-running application serving 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

UIS. User identifier service.

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 identifier service (UIS). In MQSeries for OS/2 Warp, the facility that allows MQI applications to associate a user ID, other than the default user ID, with MQSeries messages.

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.

X

XCF. Cross Systems Coupling Facility.

Index

Numerics

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X

Sending your comments to IBM

MQSeries

Planning Guide

GC33-1349-06

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 request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should 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, use the Readers' Comment Form
- By fax:
  - From outside the U.K., after your international access code use 44 1962 870229
  - From within the U.K., use 01962 870229
- Electronically, use the appropriate network ID:
  - IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
  - IBMLink: WINVMD(IDRCF)
  - Internet: idrcf@winvmd.vnet.ibm.com

Whichever you use, ensure that you include:

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

Readers' Comments

MQSeries

Planning Guide

GC33-1349-06

Use this form to tell us what you think about this manual. If you have found errors in it, or if you want to express your opinion about it (such as organization, subject matter, appearance) or make suggestions for improvement, this is the form to use.

To request additional publications, or to ask questions or make comments about the functions of IBM products or systems, you should talk to your IBM representative or to your IBM authorized remarketer. This form is provided for comments about the information in this manual and the way it is presented.

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.

Be sure to print your name and address below if you would like a reply.

Name	Address
Company or Organization	
Telephone	Email

You can send your comments POST FREE on this form from any one of these countries:

Australia	Finland	Iceland	Netherlands	Singapore	United States
Belgium	France	Israel	New Zealand	Spain	of America
Bermuda	Germany	Italy	Norway	Sweden	
Cyprus	Greece	Luxembourg	Portugal	Switzerland	
Denmark	Hong Kong	Monaco	Republic of Ireland	United Arab Emirates	

If your country is not listed here, your local IBM representative will be pleased to forward your comments to us. Or you can pay the postage and send the form direct to IBM (this includes mailing in the U.K.).

Cut along this line

2 Fold along this line

By air mail
Par avion

IBRS/CCRI NUMBER: PHQ - D/1348/SO

NE PAS AFFRANCHIR

NO STAMP REQUIRED

REPONSE PAYEE
GRANDE-BRETAGNE

IBM United Kingdom Laboratories
Information Development Department (MP095)
Hursley Park,
WINCHESTER, Hants
SO21 2ZZ United Kingdom

3 Fold along this line

From: Name _____

Company or Organization _____

Address _____

_____

EMAIL _____

Telephone _____

Cut along this line

4 Fasten here with adhesive tape

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

GC33-1349-06

Spine information:

MQSeries Planning Guide