MQSeries for Windows**

GC33-1965-00

User’s Guide

Version 2.1

IBM

MQSeries for Windows**

GC33-1965-00

User’s Guide

Version 2.1

Note

Before using this information and the product it supports, be sure to read the general information in Appendix F, “Notices” on page 225.

First Edition (September 1997)

This edition applies to IBM MQSeries for Windows, Version 2.1, program number 5639-B69, 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

Contents

Contents

Figures and tables

Figures

Tables

Figures and tables

About this book

IBM MQSeries for Windows** Version 2.1 provides messaging and queuing services on Microsoft Windows 95 and Windows NT Version 4.0. It contains many improvements over Version 2.0: the main changes are listed in “Summary of Changes” on page xix.

This book tells you how to install, set up, administer, and operate MQSeries for Windows. However, this book is just to get you started with MQSeries, so for more advanced topics (such as detailed guidance on how to write an MQSeries application) it refers you to other MQSeries publications.

This book introduces you to the MQSeries Properties dialog box, which is the user interface for MQSeries for Windows. Using this, you can control the operation of MQSeries for Windows. For detailed information on this dialog box, see the online help. This book does not contain descriptions of the commands you can use when you are running or administering MQSeries for Windows; for descriptions of those commands, see the online *MQSeries for Windows Command Reference* that is installed as part of the product.

Who this book is for

This book is for:

- Users of MQSeries applications who want to install and configure MQSeries for Windows.
- MQSeries administrators and operators who want to add an MQSeries for Windows queue manager to an existing MQSeries network.
- Application programmers who want to modify an existing MQSeries application or write a new application to run with MQSeries for Windows. This book explains those features of the IBM Message Queue Interface (MQI) that MQSeries for Windows does not support.

What you need to know to understand this book

The knowledge you need in order to understand this book depends on what you want to use MQSeries for Windows for:

- If you want to use MQSeries for Windows on your own workstation to run MQSeries applications, you need basic skills in the Microsoft Windows operating system to understand the descriptions and procedures in this book. Also, it is helpful to have a basic understanding of TCP/IP and the local area network you will use. You do not need experience of other MQSeries products.
- If you want to support other users of MQSeries for Windows, you need some experience of system administration, in addition to the skills required by application users. This book gives you an introduction to using standard MQSeries administration features on MQSeries for Windows, and it describes how the administration features of MQSeries for Windows differ from those of other

members of the MQSeries family. For more detailed information on the standard MQSeries administration features, you must read the *MQSeries Programmable System Management* manual.

- If you want to write applications to run under MQSeries for Windows, you need experience of designing and writing MQSeries applications. This book gives you an introduction to writing MQSeries applications, and it describes how the programming features of MQSeries for Windows differ from those of other members of the MQSeries family. For more detailed information on writing MQSeries applications, you must read the *MQSeries Application Programming Guide* and the *MQSeries Application Programming Reference*.

How to use this book

This book is divided into parts, one for each of the types of user of MQSeries for Windows. Within each part, the chapters are organized in a task-oriented way, so you should use the chapter that is appropriate to the task you are performing. The parts of the book are:

Part 1, For all users
This part tells you what you need to know and what you need to do before you start using MQSeries for Windows to run your applications (this includes installing the product and verifying that it has installed correctly).

Use this part of the book to learn what you can do with MQSeries for Windows if you install the Compact version of the product.

Part 2, For MQ administrators
This part is for those who administer MQ applications and their users. It tells you how to prepare files for your users to install remotely, how to create, delete, and change MQSeries for Windows components, and how to make changes and diagnose problems for your users.

Part 3, For application programmers
This part describes the application programming support that MQSeries for Windows provides, and lists the MQI features that MQSeries for Windows does not support. This book does not tell you how to write an MQSeries application; for that information, it refers you to the *MQSeries Application Programming Guide* and to the *MQSeries Application Programming Reference*.

This part also describes the sample programs that are supplied with MQSeries for Windows.

Part 4, Appendixes
This part provides reference information. It describes the differences between MQSeries for Windows and other MQSeries products, and it describes the reason codes and error messages returned by MQSeries for Windows.

Part 5, Glossary and index
The glossary contains descriptions of the new terms introduced in this book, and those terms used with other than their everyday meanings.

Appearance of text in this book

This book uses the following type styles:

Example	***Used for***
channel	In text, the first occurrence of a term that is defined in the “Glossary of terms and abbreviations”
Open	The name of a command, option, or push button
`Name=Sample_QM`	An example of text you see on the screen or in a program listing
you must not	Emphasizing a word or phrase

Terms used in this book

All new terms used in this book are defined in the “Glossary of terms and abbreviations.” These terms are shown *like this* when they first occur in this book.

In the body of this book, ***Windows*** refers to Microsoft Windows 95 and Windows NT Version 4.0 or later. MQSeries for Windows Version 2.1 ***does not*** run on earlier versions of Windows, nor on WIN-OS/2.

The name of the product, MQSeries for Windows, is sometimes shortened to MQSeries or MQ. These short names are also used for any member of the MQSeries family of products when the text does not refer specifically to MQSeries for Windows.

MQSeries for Windows publications

The following information is available for MQSeries for Windows:

- *MQSeries for Windows Version 2.1 User’s Guide*, GC33-1965 (available as a printed book and as an online book)
- *MQSeries for Windows Command Reference* (available as an online book only)
- Online help for the MQSeries Properties dialog box

To use an online book, open it from the menu of any of the help windows, or click on its icon in the MQSeries for Windows directory.

For information that became available after the books and the help were completed:

- Read the READ.ME file (use My Computer to find the file in the \Program Files\MQSeries for Windows directory)
- See the IBM MQSeries site on the Internet (see “MQSeries information available on the Internet” on page xvii)

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 AT&T GIS V5.0
- 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 OS/400 V3R2
- MQSeries for OS/400 V3R7
- MQSeries for SINIX and DC/OSx V2.2
- MQSeries for SunOS V2.2
- MQSeries for Sun Solaris V5.0
- 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 xvi. For a functional comparison of the Level 1 and Level 2 MQSeries products, see the *MQSeries Planning Guide*.)

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 publications

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 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 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 for OS/400

MQSeries for OS/400 Version 3 Release 2 Licensed Program Specifications, GC33-1360 (softcopy only)

MQSeries for OS/400 Version 3 Release 2 Administration Guide, GC33-1361

MQSeries for OS/400 Version 3 Release 2 Application Programming Reference (RPG), SC33-1362

Note: The MQSeries for OS/400 Version 3 Release 2 publications apply also to MQSeries for OS/400 Version 3 Release 7.

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 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 publications

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 Digital VMS VAX Version 1.5 User's Guide, SC33-1144

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

MQSeries for Tandem NonStop Kernel Version 1.5.1 User's Guide, SC33-1755

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

Portable Document Format (PDF)

PDF files can be viewed and printed using the Adobe** Acrobat Reader**.

If you need to obtain the Adobe Acrobat Reader, or would like up-to-date information about the platforms on which the Acrobat Reader is supported, visit the Adobe Systems Inc. web site at URL:

```
http://www.adobe.com/acrobat/acrodist.html
```

PDF files for MQSeries are shipped 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

They are also available from the MQSeries software-server home page at URL:

```
http://www.software.ibm.com/is/sw-servers/mqseries/
```

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 software-server home page at URL:

```
http://www.software.ibm.com/is/sw-servers/mqseries/
```

The following browsers are recommended:

- Lotus Notes 4.5 or later
- Netscape Navigator 3.0 or later
- Microsoft Internet Explorer
- HotJava

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 URLs

The URL of the MQSeries product family home page is:

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

You can find MQSeries SupportPacs at the following URL:

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

MQSeries on the Internet

Summary of Changes

Version 2.1 of MQSeries for Windows differs from Version 2.0 in the following areas:

- Version 2.1 is a 32-bit product, so it supports multithreaded applications. It runs on Windows 95 and Windows NT Version 4.0 only.
- Version 2.1 provides a choice of two versions at installation time:
  - The Compact version is for the users of MQ applications; these users do not need to configure MQ themselves so much of MQ is hidden from them.
  - The Complete version is for the administrators of the users of MQ applications, and for application developers; it contains extra features to help to administer users and develop applications.
- Version 2.1 integrates the utilities of Version 2.0 into a single properties dialog box.
- Version 2.1 uses a new component known as an MQ connection to hide queue managers and their components from users. An MQ connection contains all the MQ objects an application user needs to run an MQ application.
- Version 2.1 uses definition (MQD) files to configure MQ components automatically. These are similar to the initialization (INI) files that Version 2.0 uses, but Version 2.1 adds more keywords and more choices on when the file is run.
- Version 2.1 provides support for dial-up networking connections.
- Version 2.1 provides a command server, like that of other MQSeries queue manager products. This means that an MQSeries for Windows queue manager can be administered remotely using the PCF commands that are available on other Level 2 MQ products.
- Version 2.1 provides support for MQSeries events.
- Version 2.1 provides support for some extra MQSC commands.
- Version 2.1 provides support for signaling when an application uses the MQGET call.
- Version 2.1 provides fast-message support for nonpersistent messages.
- Version 2.1 provides some Visual Basic sample programs.

Summary of changes

Part 1. For all users

Chapter 1. Introduction to MQSeries for Windows

MQSeries for Windows Version 2.1 is a lightweight 32-bit messaging product that provides MQSeries functions on workstations that run Microsoft Windows** 95 or Windows NT Version 4.0. 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.

There are two groups of users of MQSeries for Windows, so it offers two versions:

Compact version
MQSeries for Windows is particularly well suited to users of messaging applications who want to use a standard configuration. It can create and start automatically the messaging components you need when you start your workstation. These features mean that users of MQ applications do not need to get involved with MQSeries so they can concentrate on the applications they want to use. Users of MQ applications should install the Compact version of MQSeries for Windows.

Complete version
The Complete version is for those who administer the users of MQ applications and for the developers of MQ applications who want to test and deploy their applications. This version contains additional features (including administration tools and sample programs) to help you with these tasks.

This chapter contains the following sections:

- "What MQ connections are for"
- "Where to use MQSeries for Windows" on page 5
- "Comparing queue managers, clients, and servers" on page 7
- "The features of MQSeries for Windows" on page 8
- "Where to find the information you need" on page 9

What MQ connections are for

To hide the many MQSeries objects from application users, MQSeries for Windows introduces the *connection* component. Typically, when MQSeries for Windows starts, it displays a dialog in which you choose a connection appropriate to the application you want to use. The application might exchange data with another computer using a LAN or dial-up telephone link, but all the information MQ needs to manage this is contained in the definition of the connection you select.

For example, if you always use a single MQ application through a LAN connection in your office, you need only a single connection to the office server, and you can configure MQ so that it starts this connection automatically for you. But if you are a mobile user, such as a travelling salesman, you might require three connections for the same application:

- One for access by telephone while working from home
- Another for working without a connection in a customer's office
- Another for access through the LAN while working in your own office

You can select which connection to use when you start your workstation.

If you use more than one application, you will need more connections. For example:

- Connection A for using application X over a LAN
- Connection B for using application X over a telephone link
- Connection C for using application Y over a LAN
- Connection D for using application Y over a telephone link
- Connection E for using application Z over a telephone link

You can have only one active connection, so if you want to use a different connection, you must first stop the active one.

If you want to learn more about MQ connections, see "Why you need MQ connections" on page 46.

If you want to learn about the MQ components that are included in your connections, see Chapter 6, "Understanding the components of MQ" on page 37.

What the Compact version provides

If you use the Compact version of MQSeries for Windows, MQ creates the connections you will need to run your MQ applications. This is because they are defined in a definition (MQD) file that MQ runs when it starts. If you need to change these connections or add new ones, your MQ administrator will give you a new MQD file and MQ can run that for you automatically as well.

If you need to, you can monitor the status of your connections using the MQSeries Properties dialog box.

What the Complete version provides

If you use the Complete version of MQSeries for Windows, you have all the facilities provided by the Compact version, and in addition you can create, change, and delete objects using the MQSeries Properties dialog box.

Also, there is information in this book to help you create MQD files for your application users and use MQ commands (MQSC and PCF) to administer their applications.

You can also write applications using the supplied header files and libraries.

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 1).

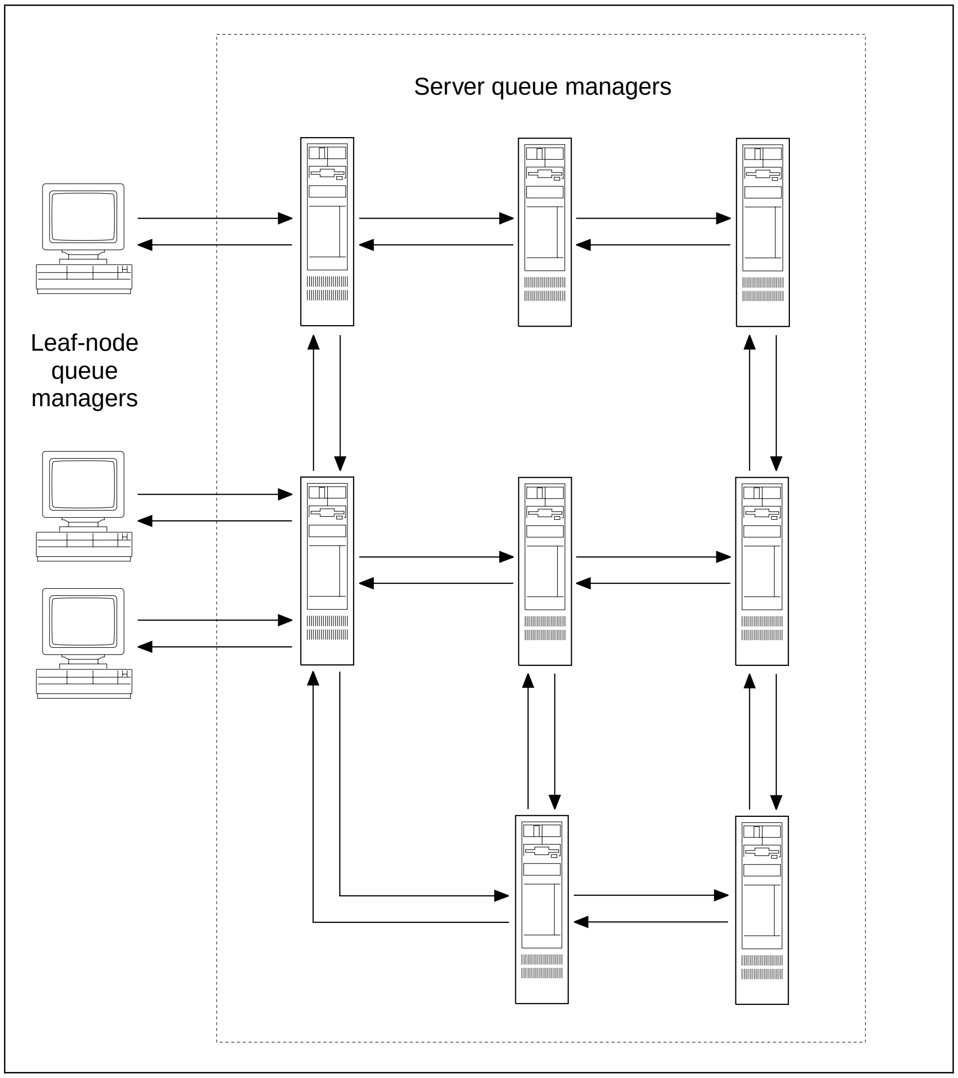


Figure 1. A network of server queue managers and three leaf-node queue managers. The leaf node queue managers run on Windows; they each connect to only one server. The server queue managers run on any other MQSeries platform; they can each be connected to other servers.

Introduction

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. MQSeries for Windows does not support MQSeries clients.

- 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 may 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 Appendix A, “Differences from the other members of the MQSeries family” on page 195.

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.

Comparing queue managers, clients, and servers

MQSeries for Windows is a single-user queue manager; it is not an MQSeries client or a server, and it does not support MQSeries clients. For a summary of the differences between an MQSeries for Windows queue manager, an MQSeries client, and an MQSeries server (such as MQSeries for Windows NT or MQSeries for OS/2), see Table 1.

Table 1. Comparison of supported features

Feature	**MQSeries for Windows**	**MQSeries client on Windows**	**MQSeries server**
Independent operation	Yes	No	Yes
Queue manager	Yes	No	Yes
Queues	Yes	No	Yes
Message channels	Yes	No	Yes
Run MQSC commands	Interactively or from a command file	No	On a command line or from a 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 startup	Yes	No	No
Supports MQSeries clients	No	Not applicable	Yes

Introduction

The features of MQSeries for Windows

MQSeries for Windows provides the following features of the MQSeries family of products, but on the Windows operating system:

- The MQSeries Message Queue Interface (MQI) for application development on Windows
- Communication between queue managers using TCP/IP
- Standard MQSeries message types and formats
- Support for multithreaded applications
- *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)
- Remote administration using PCF commands and MQ events
- Automatic installation using InstallShield and silent install
- Report generation, including confirmation of arrival (COA), confirmation of delivery (COD), and message expiry

MQSeries for Windows is a small footprint, 32-bit queue manager that runs on Windows 95 and Windows NT Version 4.0. In addition to the general MQSeries features, MQSeries for Windows provides extra features:

- To help users of applications to get started quickly and easily the first time they use the product, MQSeries for Windows can automatically create and start the messaging components the users need.
- To help users of applications to get started every time they start their workstations, MQSeries for Windows can automatically start its components.
- To help you to set up and work with your MQ components, MQSeries for Windows provides an MQSeries Properties dialog box. You can open this from the Windows taskbar and Control Panel. The pages of this dialog box have extensive online help.
- To make it easier to work with MQ components, MQSeries for Windows provides MQ *connections*. An MQ connection contains all the objects an application user needs to run an MQ application.
- 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 are started and stopped together.
- To make it easier to work with dial-up devices (such as modems) when you connect two queue managers, MQSeries for Windows allows you to create dial-up connections that use the dial-up networking features provided by Windows.

- To prevent application users having to organize MQ windows on their desktop, you can reduce MQSeries for Windows to an icon on the Windows taskbar. You can even hide this icon if you want to.
- To support application development, MQSeries for Windows provides support for the C and Visual Basic programming languages.

Where to find the information you need

The information you need when you use MQSeries for Windows depends on what you want to use the product for and how much MQSeries experience you have:

If you are new to MQSeries
For an introduction to MQSeries for Windows, see:

- Part 1, “For all users” on page 1

If you want to use the online information
To learn how to use the online information supplied with MQSeries for Windows, see:

- “The online information” on page 10

If you want information on installing MQSeries for Windows
To learn more about how to install MQSeries for Windows, see:

- Chapter 3, “Installing MQSeries for Windows” on page 15

If you want to run an MQSeries application
To learn how to use MQSeries for Windows to run an MQSeries application, see:

- Chapter 4, “Using the Compact version” on page 23

If you have used MQSeries on other platforms
To understand how MQSeries for Windows differs, see:

- Chapter 2, “Planning for MQSeries for Windows” on page 11
- Part 3, “For application programmers” on page 155
- Appendix A, “Differences from the other members of the MQSeries family” on page 195

If you want to support other users of MQSeries for Windows
To learn about the facilities MQSeries for Windows provides to help you, see:

- Part 2, “For MQ administrators” on page 33

If you are new to writing MQSeries applications
To learn how to write MQSeries applications, see:

- The *MQSeries Application Programming Guide*
- The *MQSeries Application Programming Reference*
- Part 3, “For application programmers” on page 155
- Appendix A, “Differences from the other members of the MQSeries family” on page 195

Introduction

If you have written MQSeries applications for other platforms
To learn how to migrate an existing MQSeries application to Windows, see:

- The programming restrictions described in Part 3, “For application programmers” on page 155
- Appendix A, “Differences from the other members of the MQSeries family” on page 195

The online information

When you are looking at the MQSeries Properties dialog box, you can get help about the current page by clicking on the **Help** push button.

To get help about other parts of the product and to see the online documentation:

1. Open the MQSeries Properties dialog box.
2. On any page of this dialog, click on the **Help** push button.
3. On the help page, select the **Help Topics** menu item.
4. Select the online book you want to use, then click on the **Open** push button (or double-click on the name of the book).

Chapter 2. Planning for MQSeries for Windows

This chapter describes the hardware and software you should plan to acquire before you use MQSeries for Windows, and it lists things you need to consider if you are migrating to MQSeries for Windows from other products. This chapter contains the following sections:

Suggested hardware

MQSeries for Windows is a 32-bit product, so it runs on computers that run Windows 95 or Windows NT Version 4.0. Table 2 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 may be running on the computer at the same time as MQSeries for Windows.

Table 2. Suggested hardware configurations for MQSeries for Windows

Configuration	Processor	RAM	Hard disk (Note 1)
For running applications	386DX or better	At least 4 MB	At least 3.5 MB available
For developing applications (Note 2)	486 66 MHz or better	At least 8 MB	At least 5 MB available

Note:

1. This is the space required to install MQSeries for Windows. In addition to this, you need disk space to store the MQ components you create.
2. The specification for developing applications does not include hardware requirements for other development tools (for example, compilers).

Required software

This section describes the software you require before you can use MQSeries for Windows. 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, you need the following software (or later versions):

- Microsoft Windows 95
  or Windows NT Version 4.0
- A TCP/IP product for the operating system you are using. (You can use the TCP/IP that is supplied as part of the operating system.)

MQSeries for Windows uses the TCP/IP port 1414. If you want to use a different port, add a different TCP/IP services entry in the TCP/IP configuration file named Services. You must use the name shown here (and you must type this name exactly as it is shown here, in mixed case):

```
MQSeries
```

Note: MQSeries for Windows ***does not*** run on any of the following operating systems:

- Microsoft Windows 3.1
- Versions of Windows NT earlier than 4.0
- Win32s (the Windows 32-bit subsystem that is available for 16-bit Windows)
- WIN-OS/2 on OS/2 Warp

Before you can use dial-up MQ connections, you must define dial-up networking connections using the dial-up networking support provided by Windows.

For developing MQSeries applications

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

Migrating from MQSeries for Windows Version 2.0

If you have used Version 2.0 of MQSeries for Windows, note the following when you move to Version 2.1:

- You are not recommended to install Version 2.1 of MQSeries for Windows on a workstation that has Version 2.0 already installed. You should uninstall Version 2.0 before you install Version 2.1.
- The appearance of Version 2.1 on the desktop is like that of other 32-bit Windows products. However, if you prefer to access features using icons, you can use the shortcuts that Version 2.1 puts in the MQSeries for Windows folder.

  To see how to perform the tasks you are used to performing using the Version 2.0 utilities, see Table 3 on page 14.
- Version 2.1 does not have a Create and Go utility; instead, it tests the file named CREATEMQ.MQD each time it starts. (But you can use a different file name if you use an environment variable to define it.) MQ runs this file if its date or time has changed since it was last run. Also, the user can choose to run the MQD file at any time. For more information, see Chapter 8, “Creating an MQD file” on page 57.
- If you want to reuse the INI files you used with the Create and Go utility in Version 2.0, you must compare your files with the syntax described in Chapter 8, “Creating an MQD file” on page 57.
- Version 2.1 runs only 32-bit applications, so you must change your 16-bit applications.
- Version 2.1 provides support for some additional MQSC commands. For more information, see “MQSC commands supported by MQSeries for Windows” on page 116.

Migrating

Table 3. How to perform tasks using the two versions of MQSeries for Windows

Task	**MQSeries for Windows V2.0**	**MQSeries for Windows V2.1**
To create new components	Use the Create Components utility	Use the Components page of the MQSeries Properties dialog box in the Complete version, or add definitions of each component to the MQD file.
To delete components	Use the Delete Components utility	Use the Components page of the MQSeries Properties dialog box in the Complete version, or use the MQD file.
To start and stop components, and monitor their status	Use the Standard and Advanced Controls utilities	Use the MQSeries Properties dialog box or the MQ taskbar icon.
To change components	Use the Advanced Controls utility	Use the MQSeries Properties dialog box or the MQD file.
To run a definition (MQD) or initialization (INI) file	Use the Create and Go utility to run an INI file	The MQD file is run automatically when MQSeries for Windows starts. Also, you can run it at any time from the menu of the MQ status icon on the Windows taskbar.

Migrating from MQSeries for Windows NT

If your workstation already has MQSeries for Windows NT installed (this is a server product), note the following:

- If your workstation has a version of MQSeries for Windows NT earlier than Version 5.0 already installed, you are not recommended to install Version 2.1 of MQSeries for Windows. You should uninstall MQSeries for Windows NT before you install MQSeries for Windows.
- If your workstation has Version 5.0 of MQSeries for Windows NT already installed, you can also install Version 2.1 of MQSeries for Windows, but you are not recommended to do this except in a test environment.

Chapter 3. Installing MQSeries for Windows

This chapter tells you how to install MQSeries for Windows on a single workstation using the supplied diskettes or CD-ROM. It contains the following sections:

- “Installing the product”
- “Changing your installation” on page 19
- “Verifying your installation” on page 20

MQSeries for Windows is enabled for remote (or silent) installation so you can put the installation files on a LAN server for easier access. For more information on this, see Chapter 7, “Installing MQSeries for Windows automatically” on page 51.

Installing the product

If you already have the following products installed on your workstation, read the migration notes before you install Version 2.1 of MQSeries for Windows:

- MQSeries for Windows Version 2.0; see “Migrating from MQSeries for Windows Version 2.0” on page 13
- MQSeries for Windows NT; see “Migrating from MQSeries for Windows NT” on page 14

Starting the installation

To install the product:

1. On the Windows desktop, click on **Start**.
2. Select **Settings** and open the Control Panel.
3. Double-click on **Add/Remove Programs**.
4. Click on **Install**.
5. Insert the first MQSeries for Windows diskette in your diskette drive, or insert the CD-ROM in your drive.
6. Follow the instructions shown in the windows to start the InstallShield wizard (which is named Setup).

   On the CD-ROM, the setup program is in the DISK1 directory.
7. Proceed to “Completing the installation” on page 16.

Installing the product

Completing the installation

When Setup prompts you to select a type of installation, choose one of the following versions of MQSeries for Windows:

Compact If you are installing MQSeries for Windows only so that you can run MQSeries applications, select the Compact version. It includes:

- The MQSeries for Windows runtime code
- The online help
- The definitions of default MQSeries objects
- The online *MQSeries for Windows User's Guide*
- The online *MQSeries for Windows Command Reference*
- The READ.ME file, which contains information that was not available when this book was published

Complete If you are installing MQSeries for Windows so that you can create your own MQSeries applications, or so that you can administer other users of MQSeries applications, select the Complete version. It includes:

- All the features provided by the Compact version
- Administration features to help you, for example, create and delete components (these features are available in the Compact version, but they can be hidden from the user)
- The header files and libraries to enable you to compile an MQSeries for Windows application
- Sample MQSC command files
- Sample applications (in both source and executable forms) written in the C language and in Visual Basic
- A Service Trace utility

You cannot install both the Compact and the Complete versions of the product on your workstation. But if MQSeries for Windows is already installed on your workstation, Setup selects the same version automatically and reinstalls it. So, for example, if you have the Compact version of MQSeries for Windows already installed, Setup automatically reinstalls the Compact version in the same directory. If you want to change to the other version, you must first uninstall the existing version.

If you do not want to install the product on the default drive or directory, or in the default folder, Setup gives you a chance to change them.

An indicator shows the progress of the installation process; the process takes only a few minutes.

When the installation process finishes

When the installation of the ***Compact*** version finishes, MQSeries for Windows starts automatically.

When the installation of the ***Complete*** version finishes, you are given the choice of starting MQSeries for Windows immediately or of delaying the start of MQSeries for Windows until you restart your computer.

However in ***both*** cases, if MQ detects that a shared file is being used by another application, you are instructed to restart your computer.

When the installation process finishes, MQ runs the MQD file named CREATEMQ.MQD to configure MQ on your workstation. For information on how to create a new MQD file, see Chapter 8, “Creating an MQD file” on page 57.

Registering your installation

When the installation has finished, you are prompted to register your installation with IBM so we can keep you informed of changes to your product and give you information about new products. If you choose not to register now, you are reminded every 8 days.

If you want to make changes to your registration at any time, use the Registration shortcut in the MQSeries for Windows directory.

Starting to use MQSeries for Windows

MQ runs whenever Windows is running; you will see an MQ icon on your Windows taskbar. To see the MQSeries Properties dialog box (which you use to work with MQ), double-click on this icon. If you want to hide this icon, set the option in the MQSeries Properties dialog box. For more information on how to use MQ, see Chapter 4, “Using the Compact version” on page 23.

Setup also adds an MQ icon to your Control Panel (after installing the product, you cannot see the icon until you refresh the Control Panel). So if you choose to hide the MQ icon from the taskbar, you can open the MQSeries Properties dialog box from the Control Panel.

Setup also adds some shortcuts to the MQSeries for Windows folder. You can use these to open things like the online *Command Reference*.

Installing the product

Directories after installation

Figure 2 shows the directory structure after you have installed the Complete version of MQSeries for Windows. If you install the Compact version, the directories installed are a subset of those shown.

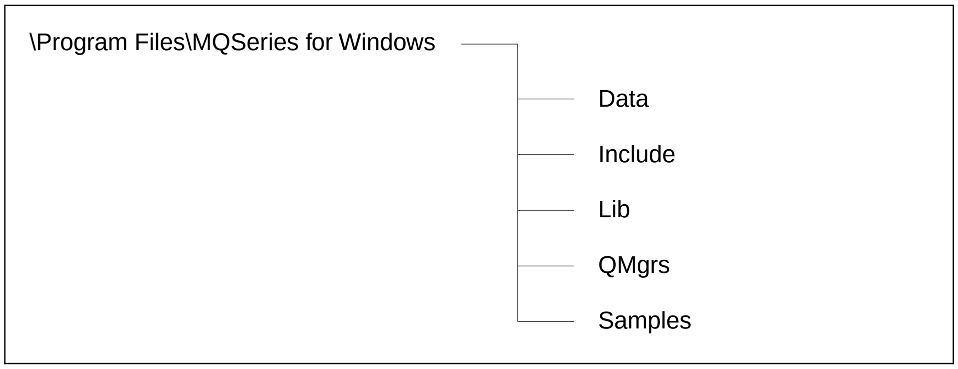


Figure 2. The default directory structure

In Figure 2, the directories are:

Data	Data files for the product, including the product log files.
Include	Include files for the Complete version.
Lib	Product libraries for the Complete version.
QMgrs	A subdirectory for each queue manager you create (this includes the channel log files). Initially, this directory is empty.
Samples	The source code and executable files for the sample programs.

Testing your installation

After you have installed MQSeries for Windows, use the procedures described in “Verifying your installation” on page 20 to verify that the installation completed successfully.

Changing your installation

After installing the product, you can use the Control Panel to:

- Apply maintenance updates
- Remove the product

Applying maintenance updates

When you receive a maintenance update (or “fix”) for MQSeries for Windows, read the READ.ME file supplied with it to check that it applies to your system, and for instructions on how to apply it.

If you want to apply the maintenance update automatically to one or more workstations, you will need response files that you can copy to a LAN server. For more information on this method, see “Changing an automatic installation” on page 55.

Removing the product

Note

When you remove MQSeries for Windows from your workstation, all MQ objects (such as connections, queue managers, and queues) are deleted as well.

To remove the product:

1. On the Windows desktop, click on **Start**.
2. Select **Settings** and open the Control Panel.
3. Double-click on **Add/Remove Programs**.
4. Select MQSeries for Windows from the list of installed software.
5. Click on **Add/Remove**.
6. Follow the on-screen instructions.

If you want to reinstall the product, you must first restart your computer.

Moving the product to another directory

You are recommended not to move the product files to different directories after installation. Setup writes configuration information to the system registry, and this information includes the location of the MQSeries product.

If you want to move the product to another directory, you must first remove it from your workstation, then reinstall it at the new location. This ensures the information in the system registry is correct.

Verifying your installation

When you have finished installing MQSeries for Windows, you should verify that the installation was successful before you try to use the product. You can use Verify at any time to test that your copy of MQSeries for Windows is working correctly. To do this, use the Verify function on the Service page of the MQSeries Properties dialog box.

You can choose to run the Verify function on your own workstation using a standalone connection, or you can run it using a connection to a server on a LAN:

- If you run Verify in standalone mode, it uses a standalone MQ connection on your own workstation to confirm MQ is installed and configured correctly.
- If you run Verify in LAN mode, it uses a LAN MQ connection to a server on your LAN so you can be sure you can exchange messages with that server.

However, you can use LAN mode only if:

- You have no active connection
- Your workstation has been configured for LAN verification
- The remote server has been configured for LAN verification

For information on how to configure the workstation and the server for LAN verification, see “Configuring for verification using a LAN” on page 102.

If you install the Complete version, you can also use the sample programs to test your installation:

- Chapter 12, “Running the sample programs on one workstation” on page 127 explains how to create a test connection and to run the sample programs on that connection.
- Chapter 13, “Running the sample programs on two workstations” on page 133 explains how to set up two connections (each on a separate workstation) and to use the sample programs to send messages between them.

Running the Verify function

To start the Verify function:

1. Open the Service page of the MQSeries Properties dialog box.
2. Open the Verify page.
3. Select either a Standalone or a LAN type of verification.
4. If you choose LAN verification, ensure the wait interval allows time for the response from the server to arrive (30 seconds should be sufficient).
5. Click on the **Start** push button.

The verification process runs automatically and takes only a few minutes. If the verification is successful, Verify displays a confirmation message.

If any of the verification tasks fail, Verify stops and displays message AMQ3601. The message shows the verification task that failed. If you can rectify the problem, do so, then restart the verification by clicking on the **Start** push button on the Verify page. If you cannot rectify the problem, try restarting your computer; otherwise you must remove the product and try again. If the error persists, contact your MQ administrator. For more information, see "If standalone verification fails."

If standalone verification fails

The most likely reasons for failure of a standalone verification are:

- There is not enough space available on the disk on which MQSeries for Windows is installed to run Verify. Or there is insufficient memory available.

  The Service page of the MQSeries Properties dialog box provides information on the amount of resources available.

  Free some space on the disk on which you have installed MQSeries for Windows, or close some of the programs you are running, then retry Verify.
- The MQSC command file that defines the default queues is missing.

  The file AMQSCOMW.TST must be in the \Program Files\MQSeries for Windows\QMgrs directory. If you have moved this file, move it back to this directory; otherwise reinstall MQSeries for Windows.
- A queue manager named SYSTEM.LCLVFY.USER.QUEUE.MANAGER already exists.

  If you have created a queue manager of this name, you must delete it before you run Verify.
- A connection named SYSTEM.LCLVFY.USER.CONNECTION already exists.

  If you have created a connection of this name, you must delete it before you run Verify.

For information on the most likely reasons for failure of a LAN verification, see "If LAN-mode verification fails" on page 104.

How standalone verification works

In standalone mode, Verify performs the following tasks if there is no active connection:

1. Create a connection
2. Start the connection
3. Open the default queue
4. Put a test message on the queue
5. Get the message from the queue
6. Close the queue
7. Stop the connection
8. Delete the connection

Verifying your installation

If there is a connection already active, Verify uses that connection to perform the following tasks:

1. Open the default queue
2. Put a test message on the queue
3. Get the message from the queue
4. Close the queue

Chapter 4. Using the Compact version

The Compact version of MQSeries for Windows is for the users of MQSeries applications. However, these users need very little contact with the MQSeries product; they need it running only so that their applications can run. If they want to make use of all the features of MQ, or make changes to their installation, this chapter describes the tasks they can perform with MQSeries for Windows. Those tasks are:

- "Starting and stopping MQSeries for Windows"
- "Choosing an MQ connection" on page 24
- "Monitoring the status of MQ components" on page 26
- "Choosing options" on page 29
- "Installing a new MQD file" on page 30
- "Getting information about your installation" on page 31
- "Using the administration features" on page 31

Starting and stopping MQSeries for Windows

MQSeries for Windows starts automatically every time Windows starts. You can choose to be prompted to select a particular connection when MQ starts; you make this choice on the Options page of the MQSeries Properties dialog box:

- If you choose to be prompted, MQ displays the Connections page of the MQSeries Properties dialog box; on this page, double-click on the name of the connection you want to start.
- If you choose not to be prompted, MQ starts the connection that is defined to start automatically. If there is no such connection, MQ does not start one.

When a connection is active, use the MQSeries icon on your Windows taskbar to monitor the status of that connection (see "The taskbar icon" on page 26).

If you choose (from the Options page of the MQSeries Properties dialog box) not to have the MQSeries icon visible on your Windows taskbar, you can open the MQSeries Properties dialog box either by:

- Opening the Control Panel and double-clicking on the MQSeries icon
- Selecting the MQSeries properties shortcut from the MQSeries for Windows folder

If you want to stop the active connection, do one of the following:

- Select **Stop connection** from the menu of the taskbar icon.
- Click the **Stop** push button on the Connections page of the MQSeries Properties dialog box.
- Double-click on the entry for the connection on the Connections page of the MQSeries Properties dialog box.

User tasks

You cannot stop MQSeries for Windows without stopping Windows. However, MQSeries for Windows uses very few resources when it has no connection active and the MQSeries Properties dialog box is not displayed, so you can leave MQ running all the time.

The remaining sections of this chapter describe the tasks you can perform using the pages of the MQSeries Properties dialog box.

Choosing an MQ connection

If there is more than one MQ connection defined on your workstation, MQ can be set up so that it opens the Connections page of the MQSeries Properties dialog box when you start your workstation. On this page, you can choose the connection you want to work with. To help you understand how you might need to choose your connection, read the following example.

A travelling salesman, named John, collects sales orders during his working day, filling in a form on his laptop computer for each order. He uses a standalone form-filling MQ application for this, which creates an MQ message for each form he fills. In the evening, John has to send the day's orders to his office. He uses another MQ application for this, and his home telephone. On some days, John has to go to his office to work, where he uses other MQ applications.

To make it easier for John to organize his work, his MQ administrator has installed MQSeries for Windows on John's laptop computer and defined three MQ connections for him to use. The connections are named:

- Collecting an order
- Sending orders
- Working at the office

Collecting an order

On the days that John is out visiting customers' offices collecting orders, he works like this:

1. When John starts to work with a customer, he powers on his laptop computer. This starts Windows 95 and MQSeries for Windows.
2. On the Connections page of MQSeries for Windows, John selects the 'Collecting an order' connection. This is a standalone MQ connection, so the only queue manager it uses is the one on his laptop.
3. John starts his form-filling application and fills in the details of his customer's order. The application saves this data as a number of persistent MQ messages on a queue on his laptop.
4. John ends his application and powers off his laptop.

Sending orders

At the end of those days that he has been out collecting orders, John has to send the day's orders to his office. On those evenings, John does this from his home like this:

1. John powers on his laptop computer. This starts Windows 95 and MQSeries for Windows.
2. John connects his laptop computer to his modem so the Windows 95 dial-up networking software can connect to the office server.
3. On the Connections page of MQSeries for Windows, John selects the 'Sending orders' connection. This is a dial-up MQ connection that can transmit messages from the queue manager running on his laptop to one running on the office server.
4. When the MQ icon on his Windows taskbar shows that the dial-up connection is ready, John starts the MQ application that processes his orders and sends them to the server.
5. When the application has sent all his messages, John ends it and powers off his laptop.

Working at the office

On the days that John has to work at the office, he works like this when he arrives there:

1. John connects his laptop computer to the office LAN, then powers on the laptop. This starts Windows 95 and MQSeries for Windows.
2. On the Connections page of MQSeries for Windows, John selects the 'Working at the office' connection. This is a LAN MQ connection that can exchange messages with the queue manager running on the office server.
3. When the MQ icon on his Windows taskbar shows that the LAN connection is ready, John starts the MQ application he uses to work with his data. He sends data to, and gets data from, the office server using this application.
4. When he has finished his work, John ends his application and powers off his laptop.

Monitoring the status of MQ components

The MQSeries icon on the Windows taskbar gives you a quick indication of the status of your MQ connection. This is described in “The taskbar icon.”

If you want information about each of the MQ components that make up your connection, look at the Status page of the MQSeries Properties dialog box; this page is described in “The Status page” on page 27.

The taskbar icon

If you have set the option on the Options page of the MQSeries Properties dialog box, you can see an MQSeries icon on the Windows taskbar. The presence of this icon shows that MQSeries is running. In addition, the icon can have a marker to show the status of the current connection, so that you can see at a glance whether you can use it (if there is no marker, there is no active connection):

✓	The connection is running correctly. You can use it to transmit messages.
!	The connection is starting or stopping. If you are starting the connection, you cannot use it until this marker changes to show the connection is running correctly.
✖	The connection has started, but it has found a problem. For more information, see the Status page of the MQSeries Properties dialog box.

You can also use the icon to:

- Display the MQSeries Properties dialog box (by double-clicking on the icon)
- Stop the active connection or start a new one (select these items from the menu that appears when you click on the icon using mouse button 2)
- Run an MQD file, provided the MQSeries Properties dialog box is not displayed and there is no active connection (for more information, see “Installing a new MQD file” on page 30)

The Status page

The Status page of the MQSeries Properties dialog box of the Compact version of MQSeries for Windows has a tree view showing the components that comprise your connections. You can expand each component to see the components it owns. For example, if you expand the Sample Connection that is supplied with MQSeries for Windows, you see that it has a queue manager named Sample_QM. If you expand this queue manager while the connection is active, you see that the queue manager owns some sample and default queues. If the queue manager owned any channel groups, channels, or a phonebook entry, they would appear in this tree view.

Figure 3 shows the Status page.

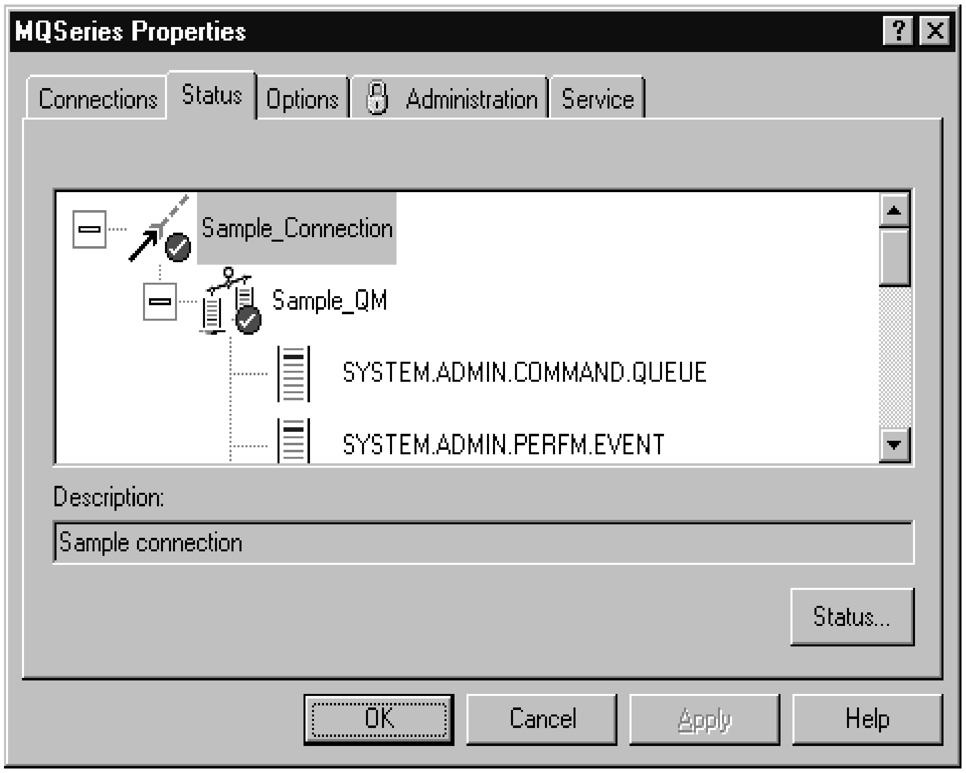


Figure 3. The Status page of the MQSeries Properties dialog box showing the default MQ components. In the Complete version, this information is shown in the Components page.

User tasks

Alongside each name in the tree view there is an icon that shows the type and the status of that component. The types of component are:

A ***dial-up connection***; use this to transmit data to a computer that is connected to yours through a telephone link.

A ***LAN connection***; use this to transmit data to a computer that is connected to yours through a local area network.

A ***standalone connection***; use this when you want to work with MQ on your own workstation and you do not need a connection to another computer.

The ***queue manager*** that the connection uses. This queue manager owns the queues and channels your connection uses to transmit your messages.

The ***channel group*** that the connection uses. This channel group contains the channels your connection uses to transmit your messages.

A ***channel*** that belongs to the channel group. This is one of the channels your connection uses to transmit your messages.

The ***channel listener*** program. This monitors connection requests from other computers. The listener belongs to a channel group.

A ***queue*** that belongs to the queue manager. This is one of the queues your connection uses to transmit your messages.

The ***phonebook entry*** that the dial-up connection uses when it uses the Windows dial-up networking support to call another computer.

If the component is idle (that is, it has not started), the icon appears as it is shown above. Otherwise, the icon for each component can have one of three markers to show its status:

The component is active. You can use it to transmit data.

The component is starting or stopping. For a channel, this state could mean that the channel is retrying to transmit data. If you are starting the connection, you cannot use it until this marker changes to show that the component is running correctly.

The component has not started correctly. You will find problems if you try to use the component when it is in this state. For more information, select the component, then click on the **Status** push button.

This tree view shows you at a glance which components you are currently using, and which, if any, are not working.

If you need more information about a component, select it in the tree view, then click on the **Status** push button. In the resulting window, you can select a status attribute to see its value. These attributes describe transient status information such as the number of bytes sent or received on a channel. For more information on these attributes, see the online help.

Choosing options

On the Options page of the MQSeries Properties dialog box you can choose options that change the following parts of MQ:

The size of the MQ icons
You can choose between displaying small and large icons on the pages of the MQSeries Properties dialog box.

The Windows taskbar
You can choose whether you want the MQ status icon to appear in the Windows taskbar. If you choose to show the icon, you can see at a glance the status of your MQ connection (see “The taskbar icon” on page 26). When you double-click on this icon, MQSeries for Windows displays the MQSeries Properties dialog box. If you choose not to have the MQ icon visible on your taskbar, you can see the MQSeries Properties dialog box by opening the Control Panel and double-clicking on the MQ icon, or by using the shortcut in the MQSeries for Windows folder.

The connection that MQ starts
You can choose to select a connection next time MQ starts:

- If you choose this option, MQ displays the Connections page of the MQSeries Properties dialog box when it starts. On this page, double-click on the name of the connection you want to start.
- If you do not choose this option, MQ starts the connection that is defined to start automatically. If there is no such connection, MQ does not start one.

The Service Trace program
You can make Service Trace start next time MQ starts. Service Trace collects trace information to help you isolate any problem with your application or with MQ. It is intended primarily for use under the guidance of IBM Service personnel. If you need to use it, see “Service Trace” on page 150.

The refresh of status information
You can choose how often MQ refreshes the status information shown on its icons and on the pages of the MQSeries Properties dialog box. If you do not choose this option, you can still refresh the status information manually by pressing the F5 key. Note that if you choose a frequent refresh, MQ will perform more slowly.

Installing a new MQD file

Your MQSeries administrator may give you a new MQ definition (MQD) file that changes, adds, or removes the MQ components you use with your applications. Copy the file to your workstation:

- If the file is named CREATEMQ.MQD, copy it to replace the file that MQ supplies. In a default installation, this is in the \Program Files\MQSeries for Windows directory.
- If the file has another name, copy it to the directory your administrator specifies and set the value of the MQW_MQDPATH environment variable to the name and location of the file.

If your administrator tells you to use the new MQD file immediately:

1. Stop the active MQ connection (see "Starting and stopping MQSeries for Windows" on page 23).
2. If you cannot see the MQ icon on the Windows taskbar, make it visible:
   a. Open the Control Panel.
   b. Double-click on the MQSeries icon.
   c. In the MQSeries Properties dialog box, open the Options page.
   d. Check the box to display the status icon on the taskbar.
   e. Click on the **OK** push button.
3. Close the MQSeries Properties dialog box by clicking on the **OK** or **Cancel** push button.
4. Click with mouse button 2 on the MQ icon on the Windows taskbar.
5. Click on the **Run MQD file now** item on the menu.

   MQ shows its progress while it processes the MQD file. If MQ cannot process the file (for example, because the file contains errors), it undoes any changes the file has made to your installation and continues to use your previous MQD file.

If you do not run the MQD file straight away, MQ automatically runs the file (and performs the definitions it contains) next time you start Windows.

However, whichever method you use to run the new file, MQ does not process it if the file contains instructions to MQ to process it only after a specified date or time. In this case, MQ processes the file the first time it starts (or the first time you use the **Run MQD file now** action, whichever is the sooner), after that date or time.

If you need to know how to create an MQD file, see Chapter 8, "Creating an MQD file" on page 57.

Getting information about your installation

If you have a problem with MQ, you or your MQ administrator can use the Service page of the MQSeries Properties dialog box to see information that may help you solve the problem. This page provides information about the version of MQSeries for Windows that is installed, together with information about the resources on your system. The page is described in “Service information” on page 149.

The Service page also contains the Verify function. Use this at any time to test that your installation is working correctly. For more information, see “Verifying your installation” on page 20.

Using the administration features

Note

Use the administration features of MQSeries for Windows only under the guidance of your MQSeries administrator.

If your MQSeries administrator asks you to make changes to your installation, you need to use the Administration page of the MQSeries Properties dialog box. To prevent you accidentally changing your installation, your administrator can define a password to lock this page. If the page is locked, you must type the password your administrator gives you before you can see the page.

Note: The administration password is case sensitive, so you must type it exactly as your administrator has defined it.

If you had to unlock the Administration page, MQ locks it again when you close the MQSeries Properties dialog box.

On the Administration page you can:

- View and change the attributes of MQ components
- Use MQSC commands
- Control the command server

Viewing and changing attributes

If you want to look at or change the attributes of MQ components, see “Viewing and changing object attributes” on page 98.

Using MQSC commands

If you want to use MQSC commands to change your MQ objects (either by typing the commands or by running a command file), see Chapter 10, “Using MQSC commands” on page 107.

User tasks

Controlling the command server

If your MQ administrator wants to make changes to your MQ installation while working from their own server, they will ask you to start the command server on your workstation.

If there is no active connection, you cannot use the command server. To start and stop the command server, first open the Administration page of the MQSeries Properties dialog box, then use the push buttons on the Command Server page. When the command server is running, MQ adds a check mark to the tab of the Command Server page of the MQSeries Properties dialog box.

If you stop the active connection, the command server also stops.

The command server processes the *programmable command format* (PCF) commands that administration applications put on a special queue. The command server runs on the queue manager owned by the active connection.

Part 2. For MQ administrators

Chapter 5. What an MQ administrator can do

If you administer the users of MQSeries for Windows applications, you will need to create and change objects for them and diagnose their problems. So that you have the tools you need to do this, you should install the Complete version of MQSeries for Windows on your own workstation.

The following list gives an overview of the tasks you will want to perform; it also shows where to find information on how to perform the tasks.

Understand MQ components
To administer the users of MQSeries for Windows applications, you need to understand the MQ components you will be working with. For more information, see Chapter 6, "Understanding the components of MQ" on page 37.

Install MQ automatically on your users' workstations
If you want to install MQSeries for Windows on a LAN so that your users can install it automatically from there, see Chapter 7, "Installing MQSeries for Windows automatically" on page 51.

Customize MQ for your users
You can provide a customized MQ definition (MQD) file to your application users so that MQ can create working connections for them automatically. For more information, see Chapter 8, "Creating an MQD file" on page 57.

Create, change, and delete MQ components manually
You may need to change quickly the configuration of an MQ connection on a workstation when you are developing, testing, or supporting MQ applications. In the Complete version, the MQSeries Properties dialog box provides a Components page in which you can create, change, and delete components. For more information, see Chapter 9, "Working with the components of MQSeries for Windows" on page 87.

Make changes to queue managers using MQSC commands
Users of the Compact version can issue MQSC commands and run command files to make changes to queue managers on their workstations, but they must first make the administration features available. You can choose to hide these features from the user by specifying a password in the user's MQD file.

With the Complete version, you can issue MQSC commands and run command files directly from the MQSeries Properties dialog box. For more information, see Chapter 10, "Using MQSC commands" on page 107.

Remotely administer a user's queue manager
You can make changes to a user's queue manager using a PCF application on your own server. You may want to do this if you are at a different location from the user. For more information, see Chapter 11, "Making changes for a user" on page 119.

Administration tasks

Run the sample programs
The Complete version provides some sample programs. For information on how to run them, see:

- Chapter 12, "Running the sample programs on one workstation" on page 127
- Chapter 13, "Running the sample programs on two workstations" on page 133

Diagnose problems
If your users have problems on their queue managers, you can use the Service Trace utility and MQ events to get more information to help you solve those problems. For information on these features, and for advice on diagnosing problems, see Chapter 14, "Diagnosing problems" on page 143.

If you need to migrate your application users from other MQ products, see:

- "Migrating from MQSeries for Windows Version 2.0" on page 13
- "Migrating from MQSeries for Windows NT" on page 14

Chapter 6. Understanding the components of MQ

This chapter helps you to understand the MQ components you will work with (connections, queue managers, channel groups, channels, and queues). It contains the following sections:

- "Introduction to messaging and queuing"
- "Why you need channels" on page 38
- "Why you need channel groups" on page 44
- "Why you need MQ connections" on page 46

Introduction to messaging and queuing

The IBM MQSeries range of products provides application programming services that enable application programs to communicate with each other using *messages* and *queues*. This form of communication is referred to as *commercial messaging*. It provides assured, once-only delivery of messages. Using MQSeries means that you can separate application programs, so that the program sending a message can continue processing without having to wait for a reply from the receiver. If the receiver, or the communication channel to it, is temporarily unavailable, the message can be forwarded at a later time. MQSeries also provides mechanisms for providing acknowledgements of messages received.

The programs that comprise an MQSeries application can be running on different workstations, on different operating systems, and at different locations. The applications are written using a common programming interface known as the *Message Queue Interface (MQI)*, so that applications developed on one platform can be transferred to another.

When two applications communicate using messages and queues, one application puts a message on a queue, and the other application gets that message from the queue. In MQSeries, queues are managed by a component called a *queue manager*. The queue manager provides messaging services for the applications and processes the MQI calls they issue. The queue manager ensures that messages are put on the correct queue or that they are routed to another queue manager.

Before applications can send any messages, you must create a queue manager and some queues. MQSeries for Windows provides features to help you do this.

How applications identify themselves to queue managers

Any MQSeries application must make a successful MQI connect call to a queue manager before it can make any other MQI calls. If the call is successful, the queue manager returns a *connection handle*. This is an identifier that the application must specify each time it issues an MQI call. An application can connect to only one queue manager at a time (known as its *local queue manager*), but many application threads can connect to the same queue manager. When the application has connected to a queue manager, all the MQI calls it issues are processed by that queue manager until it issues another MQI call to disconnect from that queue manager.

Channels

Opening a queue

Before your application can use a queue for messaging, it must open the queue. If you are putting a message on a queue, your application must open the queue for putting. Similarly, if you are getting a message from a queue, your application must open the queue for getting. You can specify that a queue is opened for both getting and putting, if required. The queue manager returns an *object handle* if the open request is successful. The application specifies this handle, together with the connection handle, when it issues a put or a get call. This ensures that the request is carried out on the correct queue.

Putting and getting messages

When the open request is confirmed, your application can put a message on the queue. To do this, it uses another MQI call in which you have to specify a number of parameters and data structures. These define all the information about the message you are putting, including the message type, its destination, which options are set, and so on. The message data (that is, the application-specific contents of the message your application is sending) is defined in a buffer, which you specify in the MQI call. When the queue manager processes the call, it adds a *message descriptor*, which contains information that is needed to ensure the message can be delivered properly. The message descriptor is in a format defined by MQSeries; the message data is defined by your application (this is what you put into the message data buffer in your application code).

The program that gets the messages from the queue must first open the queue for getting messages. It must then issue another MQI call to get the message from the queue. On this call, you have to specify which message you want to get.

Figure 4 on page 39 shows how messaging works in the simple case where the program putting the message and the program getting the message are both on the same workstation and connected to the same queue manager.

Messaging using more than one queue manager

The arrangement shown in Figure 4 on page 39 is not typical for a commercial messaging application because both programs are running on the same workstation, and connected to the same queue manager. In a commercial application, the putting and getting programs would probably be on different workstations, and so connected to different queue managers. In this situation, you also need to create *message channels* to carry MQSeries messages between the queue managers. This situation is described in “Why you need channels.”

Why you need channels

The queue manager to which an application is connected is known as its *local queue manager*. Any other queue manager is known as a *remote queue manager*, whether it is running on the same workstation or on any other, no matter where that workstation is situated. So any queues owned by the local queue manager are known as *local queues* and those owned by a remote queue manager are known as *remote queues*.

Channels

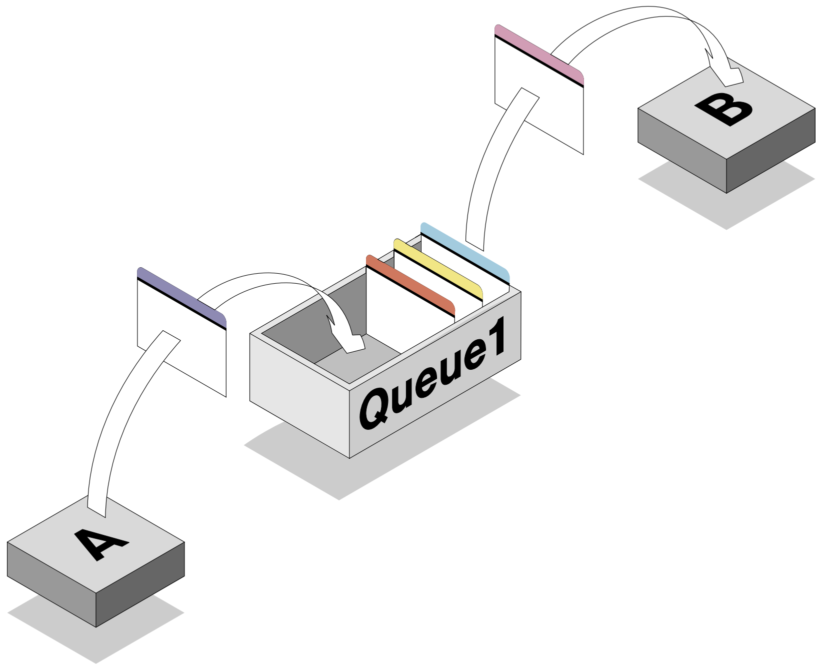


Figure 4. Programs connected to the same queue manager. Program A puts messages on the queue; program B gets messages from the queue. In this case, the programs and the queue manager are running on the same workstation.

When an application issues a call to put a message on a remote queue, the local queue manager first puts the message on one of its *transmission queues*. A transmission queue is a special type of local queue. The local queue manager stores a copy of the message on the transmission queue until the message is successfully transmitted to the remote queue manager.

The local queue manager needs to know where the remote queue is. For this reason, when you set up your queue manager you must provide a *local definition* of each remote queue the queue manager will use. This definition includes the name of the queue, the name of its owning queue manager, and the name of the transmission queue you want to use on the local queue manager to store messages destined for the remote queue. If you do not specify the name of a transmission queue, the local queue manager looks for a transmission queue with the same name as the remote queue manager. You can also define a default transmission queue for the local queue manager to use.

For example, the sample file MARS.TST that is used in Chapter 13, "Running the sample programs on two workstations" on page 133 uses the queue definitions shown in Figure 5 on page 40.

Channels

```
* Define a local transmission queue.
DEFINE QLOCAL('SAMPLE.MARS.XMIT') REPLACE +
       DESCR('Local transmission queue')  +
       USAGE(XMITQ)

* Define the remote queue.
DEFINE QREMOTE('SAMPLE.MARS.REMOTE') REPLACE  +
       DESCR('Remote queue defined on MARS') +
       DEFPSIST(YES) +
*           This is the name of the local queue on the remote machine.
       RNAME('SAMPLE.VENUS.LOCAL') +
*           This is the name of the queue manager on the remote machine.
       RQMNAME('VENUS') +
*           This is the name of the local transmission queue to be used.
       XMITQ('SAMPLE.MARS.XMIT')
  ⋮
* Define the local queue where the remote machine will put its messages.
DEFINE QLOCAL('SAMPLE.MARS.LOCAL') REPLACE +
       DESCR('Local queue') +
       DEFPSIST(YES)  +
       SHARE
```

Figure 5. Queue definitions from the supplied file MARS.TST

MQ transfers the message from the transmission queue to the remote queue manager through a message channel. A channel is a ***one-way*** communication link between two queue managers. This means that MQSeries messages flow in only one direction (although channel control messages flow in both directions). For two-way message flow, you must have two channels running between the two queue managers. Each end of a channel is controlled by an MQSeries-supplied program called a *message channel agent (MCA).*

An example of how to use two queue managers

To help you to understand all these features, study the example shown in Figure 6 on page 41. This shows the relationship between applications, queue managers, queues, and channels. In this example, application A (which is connected to queue manager MARS) wants to send a message, using Queue1, to application B (which is connected to queue manager VENUS).

But to application A, Queue1 is a remote queue because it is not owned by MARS. So for this communication to be successful, MARS must have:

- A local definition of remote queue Queue1
- A transmission queue to transfer messages to VENUS
- A message channel to VENUS

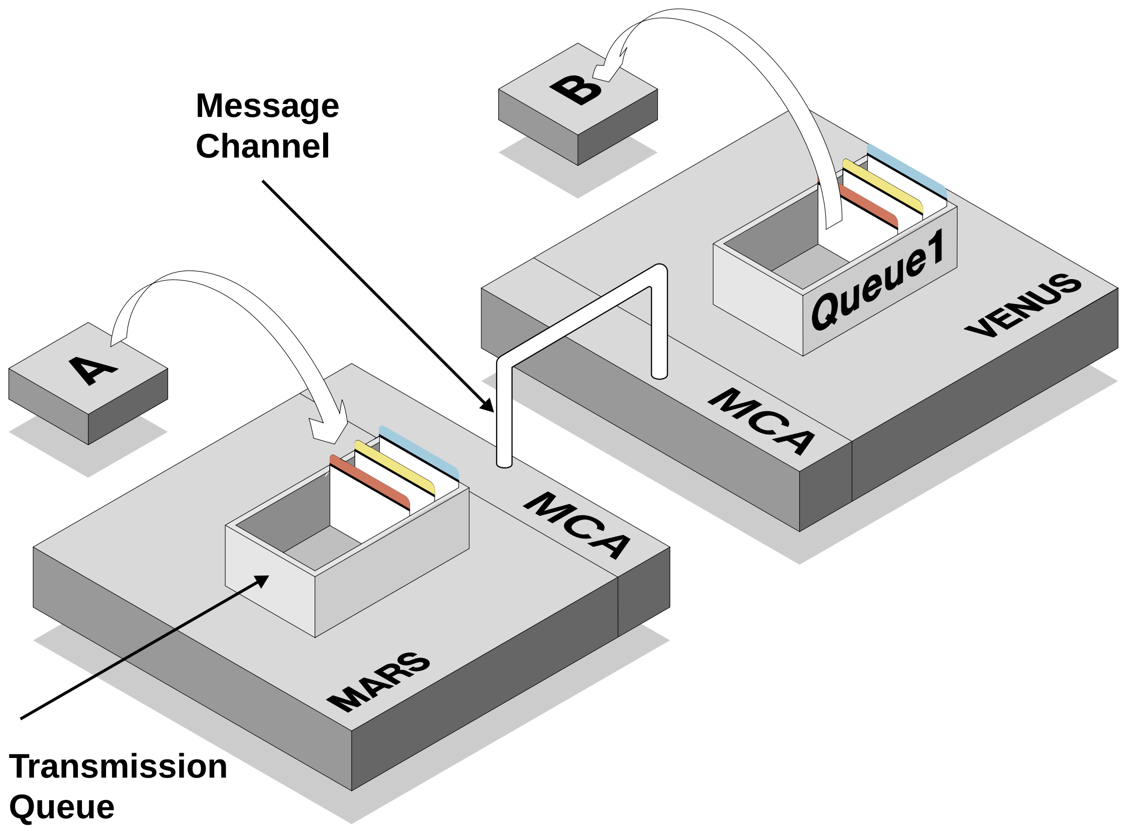


Figure 6. Communication between two queue managers. Application A puts a message on a local definition of Queue1. Application B gets the message from Queue1.

When application A puts a message (specifying the local definition of Queue1), MARS takes that message and moves it to the transmission queue. The MCA running on MARS then transfers the message to the MCA running on VENUS. The receiving MCA transfers the message from the channel to Queue1.

Note that this is a simple example, and application B cannot send replies or new messages to application A. For application B to be able do this, there must also be:

- A transmission queue on VENUS to temporarily hold messages destined for MARS
- A second channel to carry messages from VENUS to MARS (remember that message channels carry messages in one direction only)
- An MCA running on VENUS, and one on MARS, to run the new channel
- An application queue on MARS so that application A can get messages

In reality, you will probably need to use more than two queue managers in your work, but the same principles apply. If there is no direct connection between the sending queue manager and the target queue manager, the message may have to pass from one queue manager to another until it reaches the target. This is sometimes known as multi-hopping. To ensure the message can reach its destination, you must create a channel from each queue manager to the next one in the network, and there must be a transmission queue on each queue manager to store messages until they can be forwarded to the next queue manager in the network.

How a channel starts

An MCA can act either as a *caller* or as a *responder*. A caller MCA is the MCA that starts a channel by sending a connection request to a responder MCA at the other end of the channel. MQ supplies a program known as the *channel listener*, which listens for connection requests from caller MCAs. A connection request contains the channel name of the MCA for which the request is intended; the listener uses this information to start the correct responder MCA. When a responder MCA receives a connection request, it responds to the caller; the channel is now ready to carry MQSeries messages.

MQSeries for Windows uses components known as *channel groups* (see “Why you need channel groups” on page 44). You can work with channels only as part of a channel group, and that channel group must belong to an MQ connection. To start a channel, you must start the MQ connection that owns the channel group to which the channel belongs.

Channel definitions

A channel is defined by a pair of compatible definitions—one at each end of the channel. The two definitions must have the same name to identify them as a pair. You should consider using meaningful names for the definitions so it is easy to see their purpose; for example, `MARS.TO.VENUS`.

The definitions of the channel determine which end sends messages and which end receives them. There are four types of channel definition:

- Sender (sends messages)
- Receiver (receives messages)
- Server (sends messages)
- Requester (receives messages)

Sender, server, and requester channels can be callers or responders. A receiver channel can be a responder only, so it cannot start a channel.

You must use compatible channel definitions at each end of the channel. One end must be a sender or server (to move messages from a transmission queue and put them on the channel), and the other end must be a receiver or requester (to move messages from the channel to the destination queue).

You can use any of the following combinations when you define the two ends of a channel (they are summarized in Table 4 on page 44):

Sender-receiver
The sender is the caller, and the receiver is the responder. The sender calls the receiver to start the channel, then sends messages from its transmission queue to the receiver. The receiver puts the messages on the destination queues.

An example of an application that would use this type of channel is an e-mail application that allows the user to send messages.

Requester-server
The requester is the caller, and the server is the responder. The requester calls the server to start the channel. The server then sends messages from its transmission queue to the requester.

An example of an application that would use this type of channel is a mailing application that allows users to collect their mail by making a call.

You can also use a server-requester channel, which is similar, but the server initiates it. An example of this is a mailing application that delivers mail by making a call.

Requester-sender
Initially, the requester is the caller, and the sender is the responder. But the sender terminates the connection, then it becomes the caller. The sender calls back the requester, which becomes the responder. The sender then sends messages from its transmission queue to the requester. This arrangement is known as *call back*.

An example of an application that would use this type of channel is a mailing application that allows users to request their mail by making an initial call. The application then calls them back to deliver the mail, so the users do not have to pay for the deliveries.

The remaining combinations perform in the same way as those already described:

Sender-requester
This performs like a sender-receiver channel.

Server-receiver
This performs like a sender-receiver channel.

Server-requester
This performs like a sender-receiver channel.

Table 4. Allowed combinations of MCA types

	Receiver (on responder)	Requester (on responder)	Sender (on responder)	Server (on responder)
Receiver (on caller)	No	No	No	No
Requester (on caller)	No	No	Yes	Yes
Sender (on caller)	Yes	Yes	No	No
Server (on caller)	Yes	Yes	No	No

The two ends of a channel are defined on different queue managers, so they can have different attributes. Some attributes are compatible, but others are not. To resolve any differences, there is a negotiation between the two MCAs when the channel starts. If they cannot resolve the differences, the channel ends without transferring any messages.

Why you need channel groups

Channels are unidirectional, so you always need at least two of them. For this reason, MQSeries for Windows allows you to group channels together into *channel groups*. A channel group is an MQSeries for Windows component; it is simply a named collection of channels and it can include the channel listener. You define an MQ connection to include the channel group; this means that when you start or stop the connection, the channels it uses also start or stop.

A channel group is owned by the queue manager the connection uses. Each queue manager can own many channel groups, but a connection can own only one channel group.

A group can contain a maximum of 8 channels (including the listener), and all the channels in the group must belong to the same queue manager. A channel can belong to more than one channel group.

You can create a channel group at any time using the Components page of the MQSeries Properties dialog box in the Complete version of MQSeries for Windows. Use the same page to add a channel to an existing channel group or to change the attributes of the group. For more information, see “Creating a channel group” on page 90.

You can also create, change, or delete channels individually using, for example, MQSC commands. But you must not start them individually; if you do, the MQ connection to which they belong will not know their status.

Note: You cannot create, change, or delete channel groups using PCF or MQSC commands.

Designing a channel group

When you add a channel to a channel group, you are defining a caller MCA that is started when you start the group. You cannot add a responder to a group. This means you cannot add a receiver to a group because a receiver is always a responder (it cannot be a caller).

Instead of adding responder MCAs to a group, you add the MQSeries for Windows channel listener. The listener starts any number of responder MCAs.

Here are some examples to help you.

For a sender-receiver channel

For a sender-receiver channel, you must create two channel groups, one at each end of the channel:

Group at the calling end	**Group at the responding end**
Group contains sender channel	Group contains the listener

The listener starts the receiver MCA.

For two channels

If there are two channels (allowing two-way communication), you still need two channel groups. For example, if the two channels are a sender-receiver and a requester-server:

Group at the calling end	**Group at the responding end**
Group contains the sender channel and the requester channel	Group contains the listener

The listener starts the receiver MCA for the sender-receiver channel, and it starts the server MCA for the requester-server channel.

For two channels using call back

If there are two channels using call back, you need two channel groups. The two channels are a sender-receiver and a requester-sender.

Group at the calling end	**Group at the responding end**
Group contains the sender channel and the requester channel and the listener	Group contains the listener

The listener at the responding end starts the receiver MCA for the sender-receiver channel, and it starts the sender MCA for the requester-sender channel. When the sender MCA for the requester-sender channel ends the connection and calls back, the listener on the original caller restarts the requester MCA.

MQSeries connections

Why you need MQ connections

To hide the complexities of queue managers, channel groups, and phonebook entries from application users, MQSeries for Windows introduces the *connection* component. An MQSeries for Windows connection comprises a queue manager and all the objects the queue manager needs in order to communicate with another queue manager. So all the application user has to do is start the appropriate connection. A connection has a name, just like other MQ components.

So that your application users do not have to do any MQ administration tasks (such as creating queues and changing definitions), you should create connections for them to use. Typically you will define connections in an MQD file so that all they have to do is run that file, then choose a connection that sets up the MQ environment they need to run their application.

There are three types of MQ connection:

A ***standalone connection*** is for using MQ without connecting to other computers. It comprises a queue manager only.

Users need a standalone connection to run an application that does not need to communicate with another queue manager during the current session. For example, they might be working at a location where they do not have access to a LAN or telephone, and they just want to enter some data into their MQ applications.

A ***LAN connection*** is for using MQ together with queue managers on servers to which your workstation is connected by a local area network. It comprises a queue manager and a channel group.

Users need a LAN connection to run an application that has to communicate with another queue manager running on a workstation connected to the same LAN. For example, they might be working in their own offices where they have access to a LAN.

A ***dial-up connection*** is for using MQ together with queue managers on servers to which your workstation can connect using a dial-up telephone link. It comprises a queue manager, a channel group, and the phonebook entry that Windows uses to dial the server.

Users need a dial-up connection to run an application that has to communicate with another queue manager running on a computer to which they can dial in. For example, they might be working in their own homes or in a hotel room. For more information, see “Dial-up connections” on page 47.

You can configure MQ either to prompt the user to choose a connection each time MQ starts, or to start one automatically.

You can create connections using the MQSeries Properties dialog box (see “Creating a connection” on page 92), but this allows you to create connections on one workstation only. Alternatively, you can create definitions in a file you can give to many users; for more information on this, see Chapter 8, “Creating an MQD file” on page 57.

Other features of an MQ connection are:

- There must be at least one connection defined on an MQSeries for Windows workstation.
- Only one connection can be active at one time.
- A queue manager can belong to more than one connection.
- You can create a queue manager without defining a connection, but you cannot use the facilities of MQSeries for Windows to manage that queue manager until you define a connection that includes the queue manager.
- You cannot create, delete, or change connections using MQSC or PCF commands.

Dial-up connections

A dial-up connection uses the built in dial-up networking support provided by the Windows operating system. If this is not installed, you must install it before you can create a dial-up connection. Then you must use Windows to create what Windows calls *dial-up networking connections*; MQ calls these *phonebook entries*. These entries are like a paper phone book; they relate a person’s name to a telephone number. When you have defined the phonebook entry, you can use the name to access the telephone number.

When you create an MQ dial-up connection, you must select a phonebook entry from the list of those you have defined on your workstation. You must also select a channel group that contains the channels to carry messages to and from the queue manager you have dialed.

Examples of connections

Table 5 shows the composition of the three types of MQ connection.

Table 5. The composition of MQ connections

Connection type	First MQ component	Second MQ component	Transport component
Standalone connection	Queue manager		
LAN connection	Queue manager	Channel group	
Dial-up connection	Queue manager	Channel group	Phonebook entry

Note:

- A standalone connection requires a queue manager only.
- A LAN connection uses the transport services provided by the LAN.

MQSeries connections

Figure 7 shows examples of five MQ connections. Table 6 shows the composition of each of the example connections. Notice that a queue manager can belong to more than one connection. Also note that each queue manager owns a set of channel groups, but the phonebook entry (PBE) is independent of the queue manager (it is managed by the operating system).

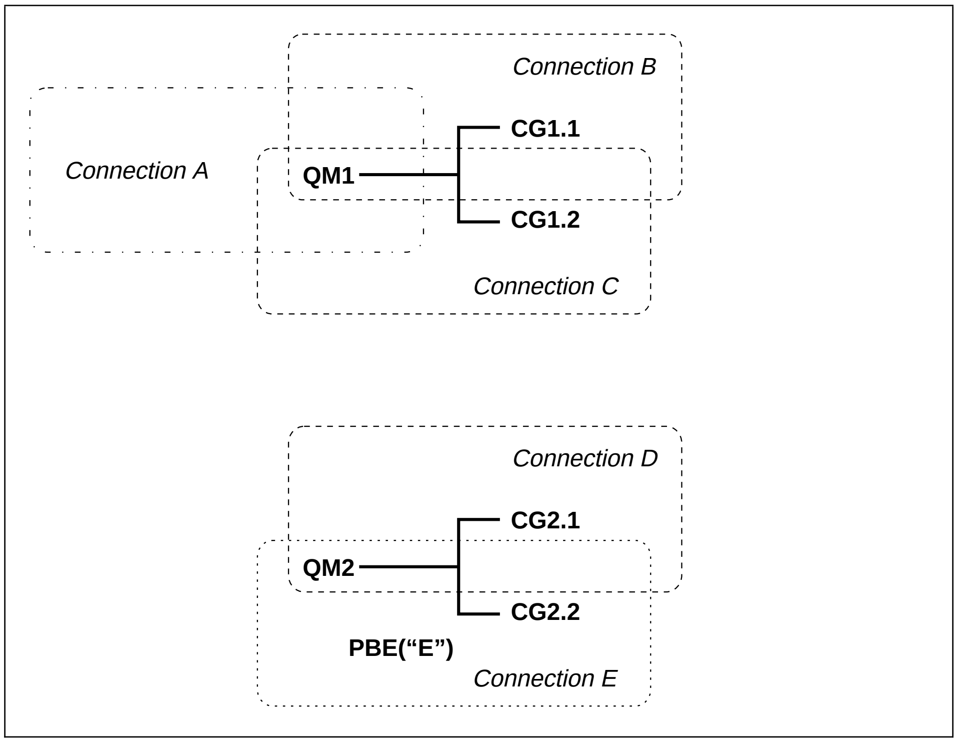


Figure 7. Examples of MQ connections

Table 6. The composition of the example MQ connections

Connection name	Connection type	Queue manager name	Channel group name	Phonebook entry name
A	Standalone	QM1		
B	LAN	QM1	CG1.1	
C	LAN	QM1	CG1.2	
D	LAN	QM2	CG2.1	
E	Dial-up	QM2	CG2.2	E

MQSeries connections

Users who always run a single MQ application through a LAN connection in their offices require only a single connection to the office server. But mobile users, such as utilities engineers, might require three connections for the same application:

- One for access by telephone while working from home
- Another for working without a connection in a customer's office
- Another for access through the LAN while working in their own office

Such users can choose to be prompted to select which connection to use when they start their workstations. If they choose not to be prompted, MQ can automatically start one of the connections.

Users who run more than one application might require more connections. For example:

- Connection 1 for using application X over a LAN
- Connection 2 for using application X over a telephone link
- Connection 3 for using application Y over a LAN
- Connection 4 for using application Y over a telephone link
- Connection 5 for using application Z over a telephone link

MQSeries connections

Chapter 7. Installing MQSeries for Windows automatically

This chapter tells you how to install MQSeries for Windows on one or more workstations connected to a LAN, with either little or no user intervention. You will find this method especially useful if you have to install MQSeries for Windows on a large number of workstations. You can do both the initial installation and the installation of any maintenance updates in this way.

There are two ways of starting an automatic installation:

- If you want to start the automatic installation manually on each workstation (sometimes known as an attended automatic installation), you must install MQSeries for Windows on a server, prepare installation response files, then start the installation process from each workstation. The installation process takes instructions from your response files. This process is described in "Installing from a file server" on page 54.
- If you want to start the automatic installation remotely using a software distribution package such as Microsoft's Systems Management Server (SMS), you must create some installation response files for the package to use. (This is sometimes known as an unattended installation.) This process is described in "Installing using a software distribution package" on page 54.

For both types of installation, you must first prepare your server as described in "Preparing your server."

Preparing your server

Before you perform an automatic installation (either attended or unattended), you must prepare your server as follows:

1. Copy the MQSeries for Windows product files to your server as described in "Preparing your files" on page 52.
2. If you want to change the drive or directory into which MQSeries for Windows is installed on your target workstations, change the supplied response files. For information on how to do this, see "Changing the destination drive and directory" on page 52.
3. If you want to change the program folder into which MQSeries for Windows is installed on your target workstations, change the supplied response files. For information on how to do this, see "Changing the program folder" on page 53.
4. If you want to change the supplied product so that your users do not have to do any configuration themselves, see "Customizing MQ for your users" on page 53.
5. Proceed to one of:
   - "Installing from a file server" on page 54
   - "Installing using a software distribution package" on page 54

Automatic installation

Preparing your files

You must copy on to your server the files that MQSeries for Windows supplies on diskettes and CD-ROM. Create a directory on your server to hold these files and within the directory create the structure shown in Figure 8. In this figure, the name `MyDirectory` is an example only; you must use all the other names.

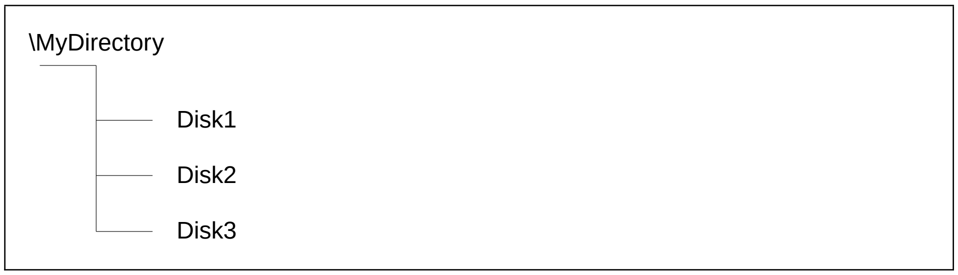


Figure 8. The directory structure for automatic installation from a server. The name `MyDirectory` *is an example only; you must use all the other names.*

Then copy the supplied files to this structure:

sms.bat This is the supplied batch file that runs InstallShield; you need it for both attended and unattended installation. Copy this file into the directory you create (for example, MyDirectory).

Disk1 Copy into this directory the files supplied on Diskette 1 or in the DISK1 directory of the CD-ROM.

Disk2 Copy into this directory the files supplied on Diskette 2 or in the DISK2 directory of the CD-ROM.

Disk3 Copy into this directory the files supplied on Diskette 3 or in the DISK3 directory of the CD-ROM.

MQSeries for Windows supplies the following response files to help you with automatic installation. They are supplied on Diskette 1 and in the DISK1 directory of the CD-ROM.

COMPACT.ISS
Response file for silently installing the Compact version of MQSeries for Windows

COMPLETE.ISS
Response file for silently installing the Complete version of MQSeries for Windows

Changing the destination drive and directory

If you want to change the drive or directory in which MQSeries for Windows is installed on your target workstations, you must change the text **shown like this** in the following line of the supplied response (.ISS) files:

```
szDir=C:\Program Files\MQSeries for Windows
```

Changing the program folder

If you want to change the name of the program folder in which MQSeries for Windows is installed on your target workstations, you must change the text **shown like this** in the following line of the supplied response (.ISS) files:

```
szFolder=MQSeries for Windows
```

Customizing MQ for your users

You can use an MQSeries for Windows definition (MQD) file to define automatically the MQ components you want on each workstation. Chapter 8, "Creating an MQD file" on page 57 describes how to create an MQD file.

After you have created your MQD file, you must choose one of the following methods to make it available for your users:

- MQSeries for Windows supplies a file named CREATEMQ.MQD, which creates a sample connection. After you have copied on to your server the files that MQSeries for Windows supplies, you can replace this file with the MQD file you have created, making sure you use the same file name. Your file will be installed on each workstation when users install MQ automatically, and because it has the default name, MQ will run it.

  Note that the procedures described in Chapter 12, "Running the sample programs on one workstation" on page 127 will not work if you replace the supplied file.

- If you want all your users to use the same MQD file that is on a server, load the MQD file on to a server to which your users' workstations have access. Then set the MQW_MQDPATH environment variable on each workstation to specify the name and location of your MQD file. When the installation process runs on each workstation, it runs this MQD file.
- If you do not want all your users to use the same MQD file, but you do want them to run an MQD file from a server, create a set of different files on your server. You must then set the MQW_MQDPATH environment variable on each workstation to locate the relevant file.
- Copy your MQD file on to each workstation after you have completed the installation. You must set the environment variable to locate the file on the workstation, then run the MQD file from the MQ icon on the Windows taskbar.

If you want your users to use the Verify facility of MQSeries for Windows to test that their installation is working correctly, you must configure their workstations as described in "Configuring for verification using a LAN" on page 102.

Installing from a file server

After you have prepared your server as described in “Preparing your server” on page 51, you can install MQSeries for Windows from it on to a workstation as follows. This is sometimes known as an attended installation.

1. From the workstation, log on to the server on which you have prepared the MQ files.
2. Change to the Disk1 subdirectory you created on the server.
3. Run one of the following batch commands, depending on which type of installation you require:

   ```
   sms compact
   ```

   or:

   ```
   sms complete
   ```

   This runs InstallShield in silent mode using your response files.

When the installation process has completed, you should verify that it was successful. If you have set up the server as described in “Customizing MQ for your users” on page 53, you can use the Verify facility in the MQSeries Properties dialog box. For more information, see “Verifying your installation” on page 20.

Installing using a software distribution package

This section describes how to install MQSeries for Windows using Microsoft’s Systems Management Server (SMS), although you can use other software distribution packages instead. This is sometimes known as an unattended installation.

When you use SMS, you need to:

- Install the SMS server on your LAN server
- Install the SMS client on each target workstation

MQSeries for Windows supplies the following files for you to use with SMS:

COMPACT.PDF	SMS package definition file (PDF) for installing the Compact version of MQSeries for Windows
COMPLETE.PDF	SMS package definition file (PDF) for installing the Complete version of MQSeries for Windows

After you have prepared your server as described in “Preparing your server” on page 51, you can perform an unattended installation as follows:

1. If you want to change the name or location of the MQSeries installation log file or the Management Information File (MIF), change the supplied batch file (SMS.BAT) as described in “Changing the name and location of the log file” on page 55.
2. Create an SMS package for each type of installation (Compact or Complete) you require.

3. Import the appropriate supplied PDF file (COMPACT.PDF or COMPLETE.PDF) into your SMS package.
4. Create an SMS job to distribute your package to your target workstations.
5. Run your SMS job. On the target workstation, SMS runs InstallShield in silent mode, using the information in your response files.

When the installation process has completed, you should ask each user to verify that it was successful. If you have set up the server as described in “Customizing MQ for your users” on page 53, they can use the Verify facility in the MQSeries Properties dialog box. For more information, see “Verifying your installation” on page 20.

Changing the name and location of the log file

If you want to change the name of the log file that InstallShield creates, or change its location, you must change one line in the supplied batch file (SMS.BAT). You can also change the name of the Management Information File (MIF).

- If you are installing the Compact version, change the text **shown like this** in the following line of the file SMS.BAT to the names you want to use:

  ```
  setup -s -SMS -mMQSeries.mif -f1.\compact.iss
  -f2C:\Windows\MQSeries.log
  ```

  Note: This is a single line in the file.

- If you are installing the Complete version, change the text **shown like this** in the following line of the file SMS.BAT to the names you want to use:

  ```
  setup -s -SMS -mMQSeries.mif -f1.\complete.iss
  -f2C:\Windows\MQSeries.log
  ```

  Note: This is a single line in the file.

Changing an automatic installation

If you want to remove the product after you have installed it, you must use the Control Panel as described in “Removing the product” on page 19.

You can apply maintenance updates automatically if those updates include response files for this purpose. You can use those response files in either attended or unattended mode, in the same way you did for the original installation.

Chapter 8. Creating an MQD file

This chapter describes how to create an MQSeries definition (MQD) file to define the MQ components your application users need on their workstations. When this file runs on their workstations, MQSeries for Windows creates those MQ components automatically, so the users do not have to create their own components. You can also use an MQD file to change existing components (by replacing them with components of the same name) or to delete them.

When you have created your own MQD file (using any editor) to define the components needed for your users' applications, you can put the file on the MQSeries for Windows diskettes you give your users to install. All the users need to do is run the installation program on the diskettes; the installation process installs MQ, then runs the MQD file to create the MQ objects they need.

MQ stores the date, time, and size of the MQD file. It checks these values each time it starts; if they are different from those of the MQD file that was last run, MQ prompts the user to choose whether to process the new file. If they choose to process it, MQ creates the components you define in the file. This means you can supply a new MQD file to your users at any time, and MQ will run it next time it starts. In addition, the user can force MQ to run the MQD file at any time. To do this, they must first stop the active connection and close the MQSeries Properties dialog box. Then, in the menu of the MQ icon on the taskbar, click on the **Run MQD file now** item.

However, you can specify a date and time within the file so that MQ runs the file only after this date and time.

This chapter describes:

- "The format of the MQD file" on page 58
- "How MQ processes the MQD file" on page 62
- "Creating MQD files" on page 63

The remainder of the chapter describes the sections of the MQD file and the keywords you can use. The sections are in alphabetic order:

- "The ChannelGroup section" on page 69
- "The Connection section" on page 73
- "The Controls section" on page 78
- "The Process section" on page 79
- "The QueueManager section" on page 82

The format of the MQD file

By default, MQSeries for Windows runs the MQD file named CREATEMQ.MQD. In a default installation, MQ looks for this file in the \Program Files\MQSeries for Windows directory. The installation process puts an example of this file in this directory. For a description of this example file, see “Example MQD files” on page 60.

When you create your own MQD file, you must either rename it to CREATEMQ.MQD and copy it to the default directory, or use the MQW_MQDPATH environment variable to specify the name and location of your file. If you set the environment variable, you must specify the full path and file name. You can use any file name and file-name extension. You can put the MQD file in any directory (which can be on another workstation or a LAN server). This allows you to create only one copy of the file and to set up your users to use this copy.

Note: Chapter 12, “Running the sample programs on one workstation” on page 127, describes how to run the sample programs that are supplied with the Complete version. The procedures in that chapter will not work if you replace the supplied file CREATEMQ.MQD.

You can include comments in the MQD file, but each line of comments must have an asterisk (*) or a semicolon (;) in the first column.

Section names

The MQD file comprises one or more sections, each having the format:

```
   [section name]
   keyword=value

 ⋮
   keyword=value
```

The section names in the MQD file must either be of the form `[Component_`*n*`]` (where *n* is an integer), or one of the special section names described in “Special sections” on page 60. You must enclose the section names in square brackets [ ]. The section names are not case sensitive so, for example, MQ allows either `[ChannelGroup]` or `[channelgroup]`.

If you use one or more Component_*n* sections in the MQD file, one of them must be named `Component_1`. MQ processes any remaining `Component_` sections in numeric order, but they do not have to be in numeric order in the file. MQ stops processing the file when the next number in the sequence is missing.

Note: If you define a connection that refers either to a queue manager or channel group definition in the same file, you must number those definitions so that MQ processes them before the connection definition.

For example, consider this MQD file:

```
    [Component_3]
    ComponentType=Connection
  ⋮
    [Component_1]
    ComponentType=QueueManager
  ⋮

    [Component_2]
    ComponentType=ChannelGroup
  ⋮
```

When MQ processes this file, it first processes the QueueManager section, then the ChannelGroup section, then the Connection section.

Now consider this MQD file:

```
    [Component_5]
    ComponentType=Connection
  ⋮

    [Component_1]
    ComponentType=QueueManager
  ⋮

    [Component_2]
    ComponentType=ChannelGroup
  ⋮

    [Component_3]
    ComponentType=ChannelGroup
  ⋮
```

When MQ processes this file, it first processes the QueueManager section, followed by the two ChannelGroup sections. It then stops processing the file (without processing the Connection section) because there is a gap in the sequence of component definitions.

Format of MQD

Component types

You can define sections in the MQD file using the following component types:

Connection
Defines the properties of an MQ connection

QueueManager
Defines the properties of a queue manager

ChannelGroup
Defines the properties of a channel group

Note: To use the features of MQSeries for Windows, you need an MQ connection, so you should always define a Connection component in the MQD file. To run an MQ application, you must have a queue manager running, so unless the users of your applications are going to create their queue managers themselves, you should always define a QueueManager component in the MQD file. However, you can omit the Connection and QueueManager sections if you are creating an MQD file for users to change installations that have connections and queue managers defined already.

Special sections

In addition to the sections that have names of the form `[Component_`*n*`]`, the MQD file can contain the following special sections:

[Process]
This section defines options that control how the MQD file is run and how MQ appears on the user's workstation.

[Controls]
MQSeries for Windows Version 2.0 used this section. Version 2.1 ignores it.

Keywords

The *keyword=value* statements in the MQD file define the attributes of the components. The keywords must be followed immediately by an equals (=) sign.

The keywords you can use within each section of the MQD file are described in this chapter. They are not case sensitive, so you can type them using uppercase, lowercase, or mixed case letters.

Example MQD files

Figure 9 on page 61 shows the file CREATEMQ.MQD that is supplied with the Compact version. This file creates a queue manager and a connection, and in addition it specifies some process options. These options specify that:

- MQ runs the MQD file only after the given date and time
- Users of the Compact version must type the password `Hursley` before they can use the administration features of MQSeries for Windows
- The MQ status icon appears in the Windows taskbar
- The user cannot choose a connection when MQ starts

The sample connection uses the sample queue manager and does not have a channel group. The connection starts automatically when MQ starts.

The file named CREATEMQ.MQD that is supplied with the Complete version contains an extra line; this runs the MQSC command file that defines the MQ objects the sample programs use.

```
[Process]
UseAfterDate=1996/12/11
UseAfterTime=12:00
Password=Hursley
ShowIcon=yes
PromptForConnection=no

[component_1]
ComponentType=QueueManager
Name=Sample_QM
Description=Sample queue manager
Replace=yes

[component_2]
ComponentType=Connection
Name=Sample_Connection
Description=Sample connection
QueueManagerName=Sample_QM
HasChannelGroup=no
Replace=yes
AutoStart=yes
```

Figure 9. The file CREATEMQ.MQD supplied with the Compact version

Figure 10 on page 62 shows the supplied file MARS.MQD; this creates the queue manager MARS that is used in Chapter 13, “Running the sample programs on two workstations” on page 133. When it creates the MARS queue manager, MQ runs the MQSC command file MARS.TST to create the queues and channels that MARS uses. The MQD file also creates a channel group and a connection. The channel group contains the two channels defined by MARS.TST. The connection starts automatically when MQ starts.

Processing MQD

```
[Component_1]
ComponentType=QueueManager
Name=MARS
Description=Queue manager to communicate with VENUS
LoadUserMQSC_1=\Program Files\MQSeries for Windows\Samples\mars.tst
Replace=yes

[Component_2]
ComponentType=ChannelGroup
Name=MARSGroup
Description=Channel group to communicate with VENUS
QueueManagerName=MARS
AllUserChannels=no
Channel_1=MARS.TO.VENUS
Channel_2=VENUS.TO.MARS
Replace=yes

[Component_3]
ComponentType=Connection
Name=Mars_Connection
Description=Connection to use on MARS machine
QueueManagerName=MARS
HasChannelGroup=yes
ChannelGroupName=MARSGroup
Replace=yes
AutoStart=yes
```

Figure 10. The supplied file MARS.MQD

MQSeries for Windows also supplies a file named SAMPLE.MQD, which shows all the keywords you can use in an MQD file. You can use this as a template for your own MQD files; for more information, see "Creating MQD files" on page 63.

How MQ processes the MQD file

When MQSeries for Windows processes an MQD file, it writes any warning or error messages to a log file named CREATEMQ.LOG in the \Program Files\MQSeries for Windows\Data directory. This file is overwritten each time MQ processes a file. If any errors or warnings occur during the processing of the MQD file, the way MQ warns you depends on which version of the product you are using:

Compact version
MQ warns you that it will use the previous MQD file. MQ deletes all components it has created and restores any components it has replaced to their former state.

Complete version
MQ gives you the choice of:

- Viewing the log file
- Reverting to the previous MQD file (MQ deletes all components it has created and restores any components it has replaced to their former state)
- Using the new MQD file, even though it has errors

Creating MQD files

When you want to create an MQD file, you may find it easier to copy and edit the file SAMPLE.MQD that is supplied (in a default installation) in the \Program Files\MQSeries for Windows\Samples directory. This file shows all the keywords you can use in an MQD file: all you have to do is remove the comments from the keywords you want to use and add your own data. Alternatively, you can use any editor to create your MQD file.

Note: When you edit an MQD file, make sure you save it as a text (ASCII) file (that is, not in a word-processing format). If you open an MQD file using the Windows Explorer, Windows opens the file using WordPad and loads it as a text file.

Whichever method you use, remember either to rename your file to CREATEMQ.MQD and put it in the default directory, or use the MQW_MQDPATH environment variable to specify the name of the file and its full path.

Start by defining the connections your users will need. This will help you to specify the queue managers, channel groups, and other MQ objects you must define. You can then go on to define the options you want to use to control how MQ appears on your users' workstations.

Defining a connection

An MQ connection comprises a queue manager and the objects the application needs in order to communicate from that queue manager to another. Even if the application can run when the workstation is not connected to a queue manager on a server, you still need an MQ *standalone connection*.

Define each connection in a separate Connection section in your MQD file. Remember that the only MQ objects an application user should see are connections. Users have to choose an MQ connection when MQ starts (unless you set one connection to start automatically every time), so you need to give your connections recognizable names.

Creating MQD files

When you define your connection:

- If the user does not need to communicate with another queue manager, include only a queue manager.
- If the user will use a LAN to communicate with another queue manager, include a queue manager and a channel group.
- If the user will use a telephone for the communication, include a queue manager, a channel group, and a phonebook entry.

You must define the components the connection uses as follows:

- Define the queue manager in the QueueManager section of the MQD file.
- Define the channel group in the ChannelGroup section of the MQD file.
- The phonebook entry must already exist on the user's workstation. Create it using the dial-up networking features provided by Windows.

You must number these sections so that MQ processes them ***before*** the connection definition.

MQSeries for Windows allows only one queue manager to run at a time, so you may choose to define only a single queue manager. But this means that all the user's data is stored on the queues owned by this queue manager, so you may instead want to define a separate queue manager for each type of data. You can administer each queue manager separately, viewing and changing its characteristics.

The single queue manager that is running is determined by the connection that the user chooses. So you need to define a separate connection for each combination of queue manager and access type. For example:

- Connection A for using queue manager X on a LAN
- Connection B for using queue manager X with a telephone link
- Connection C for using queue manager Y on a LAN
- Connection D for using queue manager Y with a telephone link
- Connection E for using queue manager Z with a telephone link

You can define one connection to start automatically when MQ starts. Do this for the connection the user will use most often, and certainly if there is only one connection. The user can override this by setting the option in their MQSeries Properties dialog box to select a connection from a list when MQ starts.

When you define a connection, you can also specify that the command server starts whenever the connection starts. For information on this, see "Controlling the command server" on page 65.

For information on the keywords you can use in a Connection section in your MQD file, see "The Connection section" on page 73.

Controlling the command server

In the definition of a connection, you can specify that MQ runs the command server whenever the connection is running. The command server runs on the queue manager that the connection owns. It gets PCF command messages from the administration command queue. You need the command server running on the user's workstation if you want to administer that workstation from an administration application running on your server. You can create, change, and delete objects using PCF commands in an administration application, monitor MQ activity using MQ events, and diagnose problems (using events and display commands).

But the command server is a heavy user of system resources, so start it through the MQD file only if you want to use it regularly. If you want to use the command server only occasionally, ask the user to start it using the MQSeries Properties dialog box when you are ready; in the Compact version, the user must first enable the administration features. The user can stop the command server at any time using the MQSeries Properties dialog box. Also, the command server stops if the queue manager stops, so if the user stops the connection you cannot use PCF commands or MQ events.

Defining a queue manager

Define a QueueManager section for each queue manager you need in order to make the connections you defined in “Defining a connection” on page 63. You must give the queue manager a name. If you want all your users to use a queue manager of the same name, you can give them all the same MQD file. However, you may want them to use queue managers with unique names but having otherwise identical properties. You can still give all your users the same MQD file if you specify a question mark (?) for the name of the queue manager. When MQ runs the MQD file, it prompts the user to type the name of their own queue manager. In the MQD file, you can define the wording of the dialog that the users see when they have to type the name of their queue manager.

For information on the keywords you can use in a QueueManager section in your MQD file, see “The QueueManager section” on page 82.

Creating queues and channels

A queue manager is of no use without other MQ objects. It needs at least a local queue, and if it is going to communicate with other queue managers it needs a transmission queue, a local definition of a remote queue, and some channels. If you want to create channels and queues using the MQD file, you must do this as part of the definition of the queue manager. You must define these objects using MQSC DEFINE commands in one or more MQSC command files. (For more information on this, see “Writing MQSC command files” on page 111.) Specify in the QueueManager section of the MQD file the names of these command files. You can specify many MQSC files, so you can separate your MQSC commands in any way you want. MQ runs the MQSC command files when it has created the queue manager.

Creating MQD files

Defining a channel group for the queue manager

Define a ChannelGroup section for each channel group you need in order to make the connections you defined in “Defining a connection” on page 63. A channel group is simply a collection of channels, so all you have to do is give the group a name and specify which channels it includes. Note that a channel can belong to more than one group.

To make it easy for you to define a channel group, you can use the AllUserChannels keyword to include all the channels you create for the queue manager. This excludes the default channels that MQ defines for each queue manager, and any receiver channels. If you do not want to include all the channels (you may want to exclude some if there are a lot of them), you must list each channel name separately. You can include a maximum of 8 channels in the group, one of which can be the channel listener program.

For more information on channels and channel groups, see “Why you need channels” on page 38. For information on the keywords you can use in a ChannelGroup section in your MQD file, see “The ChannelGroup section” on page 69.

Defining the appearance of MQ

After you have defined in your MQD file all the MQ objects your users will need, you should think about how you want to control the appearance of MQ on the user’s workstation.

In the Process section of the MQD file you can specify keywords that determine the appearance of MQ (the user can change these options using the Options page of the MQSeries Properties dialog box):

- Whether the MQ status icon appears on the Windows taskbar
- Whether the user has to select from a list of connections when MQ starts
- Whether MQ uses small or large icons in the MQSeries Properties dialog box
- The interval between automatic refreshes of the component status displays
- The maximum size of the channel log file

Table 7 on page 67 and Table 8 on page 67 explain how the option to display a list of connections affects which connection MQ starts.

Table 7. How MQ displays the list of connections

	One connection is defined to start automatically	**No connection is defined to start automatically**
No connections defined in the file	Not applicable	MQ issues a warning message when it starts.
One connection defined in the file	MQ displays the Connections dialog and highlights the single connection.	MQ displays the Connections dialog and highlights the single connection.
More than one connection defined in the file	MQ displays the Connections dialog and highlights the connection that is marked for automatic starting.	MQ displays the Connections dialog and highlights the first connection in the list.

Table 8. Which connection MQ starts when the user is not prompted to select one

	One connection is defined to start automatically	**No connection is defined to start automatically**
No connections defined in the file	Not applicable	MQ does not start a connection.
One connection defined in the file	MQ starts the connection that is defined.	MQ does not start a connection.
More than one connection defined in the file	MQ starts the connection that is defined for automatic starting.	MQ does not start a connection.

You can also set the password that allows the user to access the administration features of the Compact version. You may want to do this to prevent the user accidentally changing their MQ installation. If you need to divulge this password to the user (for example, if they find a problem and they must make changes), you can change this password by giving them a new MQD file. For more information, see "Using an MQD file to change an existing installation" on page 68.

For information on the keywords you can use in a Process section in your MQD file, see "The Process section" on page 79.

Enabling LAN verification of MQ

If you want your users to use the Verify feature in LAN mode (that is, to send test messages to a remote server), you must enable this in the Process section of the MQD file. If you do not, the LAN option remains grayed out on the Verify page of the user's MQSeries Properties dialog box.

For more information, see:

- "Verifying your installation" on page 20
- "The Process section" on page 79
- "Configuring for verification using a LAN" on page 102

Creating MQD files

Using an MQD file to change an existing installation

You can use your MQD files for more than just creating objects for your users. You can also change existing objects or delete them. To do this, you must create a new MQD file and give the new file to your users. After they copy this new file to their workstations, MQ runs the file next time it starts or they can force MQ to run it at any time.

Alternatively, you can delay the use of the file by specifying in its Process section a date and time before which MQ will not run it. Even if the user tries to force MQ to run the MQD file by selecting the **Run MQD file now** option from the menu of the MQ taskbar icon, MQ still tests the file to see if you have specified a time delay.

For information on the keywords you can use in a Process section in your MQD file, see “The Process section” on page 79.

To change connections, queue managers, and channel groups, use the Replace keyword in your MQD definitions to force MQ to replace the objects of the same name. You must specify all the keywords you want to use for each new object, not just the keywords you want to change. This is because MQ deletes the existing object before it creates the new one. Note that if you replace a queue manager, MQ deletes the queues (including any messages on them) and channels that the queue manager owns. To delete these objects, you need to specify only the object name and the two delete keywords in the component sections of your MQD file. The second keyword is simply to make you check that you really want to delete the object.

For information on the keywords you can use to change these objects in your MQD file, see

- “The ChannelGroup section” on page 69
- “The Connection section” on page 73
- “The QueueManager section” on page 82

To change queues and channels, you cannot use an MQD file. Instead you must issue MQSC commands using the MQSeries Properties dialog box (see Chapter 10, “Using MQSC commands” on page 107).

Testing your MQD file

When you have created your MQD file, you can test it by selecting the **Run MQD file now** option from the menu that appears when you click with mouse button 2 on the MQ taskbar icon. You must first stop your active connection and close the MQSeries Properties dialog box.

If there are any errors in your file, MQ writes messages in the log file named CREATEMQ.LOG. In a default installation, this log file is in the \Program Files\MQSeries for Windows\Data directory. Note that this file is overwritten each time MQ processes a file.

The ChannelGroup section

To define a channel group, use a section that starts with the lines:

```
[Component_n]
ComponentType=ChannelGroup
```

You can define zero or more ChannelGroup sections in an MQD file.

For advice on how to create a ChannelGroup section, see “Defining a channel group for the queue manager” on page 66.

The following list explains the keywords you can use within the section:

Use this keyword	***To do this***

Name=	Specify the name of the channel group you want to create.
	
	You can use a maximum of 48 characters. If you use a nonvalid character, MQ replaces it with a period (.) and logs a warning. For more information on the characters you can use in the name of a channel group, see “Naming MQ objects” on page 96.
	
	If you do not specify a name, MQ selects the first name that has not already been used from the following list:
	`ChannelGroup`
	`ChannelGroup_1`
	`ChannelGroup_2`
	⋮
	`ChannelGroup_n`

Description=	Describe the channel group. Use this keyword to add a text description of the channel group. You can use a maximum of 64 characters. The default value is all blanks.

ChannelGroup section

QueueManagerName=	Specify the name of the queue manager to which the channel group is to belong. Each queue manager can own a maximum of 16 channel groups. If the queue manager you specify does not exist, MQ logs an error. If you do not specify the name of a queue manager, and MQ has created one during its current operation, it uses that name; otherwise it uses the name of any queue manager it can find. If there is no queue manager on the workstation, MQ logs an error.
StartListener=	Specify whether the listener is started when the channel group is started. Specify either `yes` or `no`. The default value is `no`. You need the listener if the channel group contains a channel that acts as a responder (that is, a channel that has to be started by requests from a queue manager other than the one that owns the channel group).
AllUserChannels=	Specify whether to include in the channel group all the user-defined channels that are defined for the owning queue manager. This excludes the MQ default channels that are defined in the AMQSCOMW.TST file and any receiver channels. Specify either `yes` or `no`. The default value is `no`. This means you must use Channel_*n* keywords to specify one or more channel names. If you specify `AllUserChannels=yes`, this saves you having to specify the names of individual channels.
Channel_*n*=	Specify the name of a channel to be included in the channel group. You can specify this keyword a maximum of 8 times, substituting successive integers for *n*, starting with 1. The entries are read in numeric order. If MQ cannot find the specified channel, it logs an error and continues processing the next component. If you specify `AllUserChannels=yes`, MQ ignores all Channel_*n* keywords.
Replace=	If a channel group of this name already exists, specify whether to replace it with this one. Specify either `yes` or `no`. The default value is `no`.

Delete=

Specify whether to delete the channel group defined by the Name keyword.

Specify either `yes` or `no`. The default value is `no`. If you specify `yes`, you must also:

- Specify `ConfirmDelete=yes`; this is to attempt to prevent accidental deletions.
- Use the QueueManagerName keyword to specify the name of the queue manager to which the channel group belongs.

If you want to delete a channel group, you must first delete the connections that use it. To do this, give the sections of the MQD file that delete the connections a lower component number than the section that deletes the channel group.

If you specify both `Delete=yes` and `ConfirmDelete=yes`, the channel group specified by the Name keyword is deleted and the other keywords in the section are ignored. If the channel group does not exist, a warning is logged and MQ continues processing the remainder of the MQD file.

ConfirmDelete=

Specify whether to confirm that you want to delete the channel group defined by the Name keyword.

Specify either `yes` or `no`. The default value is `no`.

If you specify both `Delete=yes` and `ConfirmDelete=yes`, the channel group specified by the Name keyword is deleted and the other keywords in the section are ignored. If the channel group does not exist, a warning is logged and MQ continues processing the remainder of the MQD file.

Example ChannelGroup components

The first example defines a channel group named MY_CHANNEL_GROUP on the queue manager named MY_QUEUE_MANAGER, using default values. This creates a channel group that includes all the user-defined channels, but not the listener. If there is already a channel group of that name defined on the workstation, MQ does not replace it. The channel group has no description.

```
[Component_2]
ComponentType=ChannelGroup
Name=MY_CHANNEL_GROUP
QueueManagerName=MY_QUEUE_MANAGER
AllUserChannels=yes
```

ChannelGroup section

The second example is from MARS.MQD, the sample file that creates the components you can use to run the sample programs across two workstations (see Chapter 13, “Running the sample programs on two workstations” on page 133). When MQ creates the channel group, it includes only two channels (named MARS.TO.VENUS and VENUS.TO.MARS). If there is already a channel group named MARSGroup on the workstation, MQ replaces it.

```
[Component_2]
ComponentType=ChannelGroup
Name=MARSGroup
Description=Channel group to communicate with VENUS
QueueManagerName=MARS
AllUserChannels=no
Channel_1=MARS.TO.VENUS
Channel_2=VENUS.TO.MARS
Replace=yes
```

The last example deletes MY_CHANNEL_GROUP from MY_QUEUE_MANAGER:

```
[Component_1]
ComponentType=ChannelGroup
Name=MY_CHANNEL_GROUP
QueueManagerName=MY_QUEUE_MANAGER
Delete=yes
ConfirmDelete=yes
```

The Connection section

To define a connection, use a section that starts with the lines:

```
[Component_n]
ComponentType=Connection
```

Note: You can define zero or more Connection sections in the MQD file, but to run an MQSeries application there must be at least one connection running. So you should always define a Connection component in an MQD file, unless you are using the file to make changes to an existing installation that already has a connection defined.

For advice on how to create a Connection section, see “Defining a connection” on page 63.

The following list explains the keywords you can use within the section:

Use this keyword	***To do this***

Name=	Specify the name of the connection you want to create.
	
	The name of the connection must be unique on the workstation. It can have a maximum of 48 characters.
	
	If you use a nonvalid character, MQ replaces it with a period (.) and logs a warning. For more information on the characters you can use in the name of a connection, see “Naming MQ objects” on page 96.
	
	If you do not specify a name, MQ selects the first name that has not already been used from the following list:
	`Connection`
	`Connection_1`
	`Connection_2`
	⋮
	`Connection_n`

Description=	Describe the connection. Use this keyword to add a text description of the connection. You can use a maximum of 64 characters. The default value is all blanks.

Connection section

QueueManagerName=	Specify the name of the queue manager to be started when this connection is started. If this queue manager does not already exist, you can define it in a QueueManager section of the MQD file (see "The QueueManager section" on page 82). You must define the QueueManager section before the Connection section in the file (that is, give it a lower value of the Component_*n* keyword). If the queue manager you specify does not exist, an error is logged. If you do not specify the name of a queue manager, and MQ has created one during its current operation, it uses that one; otherwise it uses the name of the first queue manager it finds. If there are no queue managers defined, MQ logs an error.
HasChannelGroup=	Specify that the connection uses a channel group. Specify either `yes` or `no`. The default value is `no`. You need a channel group for a LAN or dial-up connection. If you omit this keyword or specify `HasChannelGroup=no`, MQ ignores the ChannelGroupName keyword. If you do not specify the name of a channel group using the ChannelGroupName keyword, and MQ has created one during its current operation, it uses that one; otherwise it uses the first channel group it finds.
ChannelGroupName=	Specify the name of the channel group to be started when this connection is started. If this channel group does not already exist, you can define it in a ChannelGroup section of the MQD file (see "The ChannelGroup section" on page 69). You must define the ChannelGroup section before the Connection section in the file (that is, give it a lower value of the Component_*n* keyword). If the channel group you specify does not exist, an error is logged. If you omit this keyword, and you specify `HasChannelGroup=no`, the connection MQ creates will be of type standalone. No channel group is started.
HasPhonebookEntry=	Specify that the connection uses a phonebook entry. Specify either `yes` or `no`. The default value is `no`. If you omit this keyword or specify `HasPhonebookEntry=no`, MQ ignores the PhonebookEntryName keyword. You need a phonebook entry for a dial-up connection only. The phonebook entry must already exist on the workstation

	that runs the MQD file; you create phonebook entries using the dial-up networking support provided by Windows. If you specify `HasPhonebookEntry=yes`, but you do not specify the name of a phonebook entry using the PhonebookEntryName keyword, MQ displays a list of the phonebook entries defined on the workstation when it processes the MQD file. The user can select one of these. If you specify `HasPhonebookEntry=yes` and there are no phonebook entries defined on the workstation when MQ processes the MQD file, MQ logs a warning and processes the next section of the file.
PhonebookEntryName=	Specify the name of the phonebook entry you want MQ to dial when this connection is started. The phonebook entry must already exist on the workstation that runs the MQD file. If the phonebook entry you specify does not exist, an error is logged. Alternatively, you can make MQ prompt the user to select a phonebook entry from a list of those defined on their workstation. MQ displays the list of phonebook entries when it runs the MQD file. To do this, specify the PhonebookEntryName keyword in the following way: `PhonebookEntryName=?` If you omit this keyword, and you specify `HasPhonebookEntry=no`, the connection MQ creates will be of type standalone or LAN.
AutoStart=	Specify whether the connection is started automatically when MQ is started. Specify either `yes` or `no`. The default value is `no`. **Note:** Only one connection can be started automatically. If you specify `AutoStart=yes` for more than one connection, the last one you define is started automatically. Specify `AutoStart=yes` for the connection the user will use most often. The user can override this by setting the option to select a connection when MQ starts, but the connection you specify with this keyword will be highlighted.
RunCommandServer=	Specify whether to run the command server when the connection is running. Specify either `yes` or `no`. The default value is `no`. If you specify `yes`, the command server starts when the connection starts and ends when the connection ends.

The command server is a heavy user of system resources, so start it using this keyword only if you want to use it frequently. For more information on starting the command server, see "Controlling the command server" on page 65.

Replace=
: If a connection of this name already exists, specify whether to replace it with this one.

  Specify either `yes` or `no`. The default value is `no`.

Delete=
: Specify whether to delete the connection defined by the Name keyword.

  Specify either `yes` or `no`. The default value is `no`. If you specify `yes`, you must also specify `ConfirmDelete=yes`; this is to attempt to prevent accidental deletions.

  If you specify both `Delete=yes` and `ConfirmDelete=yes`, the connection specified by the Name keyword is deleted and the other keywords in the section are ignored. If the connection does not exist, a warning is logged and MQ continues processing the remainder of the MQD file.

ConfirmDelete=
: Specify whether to confirm that you want to delete the connection defined by the Name keyword.

  Specify either `yes` or `no`. The default value is `no`.

  If you specify both `Delete=yes` and `ConfirmDelete=yes`, the connection specified by the Name keyword is deleted and the other keywords in the section are ignored. If the connection does not exist, a warning is logged and MQ continues processing the remainder of the MQD file.

Example Connection components

The first example defines a connection named MY_CONNECTION using the default values. This connection uses the queue manager that is defined in the QueueManager section of the MQD file. The connection does not start automatically, and if there is already a connection of this name, MQ ignores this section.

```
[Component_1]
ComponentType=Connection
Name=MY_CONNECTION
```

The second example is from MARS.MQD, the sample file that creates the components you can use to run the sample programs across two workstations (see Chapter 13, “Running the sample programs on two workstations” on page 133). The Mars_Connection comprises the MARS queue manager and the MARSGroup channel group. This connection starts automatically when MQ starts.

```
[Component_3]
ComponentType=Connection
Name=Mars_Connection
Description=Connection to use on MARS machine
QueueManagerName=MARS
HasChannelGroup=yes
ChannelGroupName=MARSGroup
Replace=yes
AutoStart=yes
```

The third example deletes MY_CONNECTION:

```
[Component_1]
ComponentType=Connection
Name=MY_CONNECTION
Delete=yes
ConfirmDelete=yes
```

The Controls section

With MQSeries for Windows Version 2.0, you could use a Controls section to define some options. This section started with the line:

```
    [Controls]
```

MQSeries for Windows Version 2.1 ignores the Controls section. No warnings or errors are logged.

The Process section

To define options that control how the MQD file is run and how MQ appears on the user's workstation, use a section that starts with the line:

```
    [Process]
```

You can omit the Process section, or you can define a single Process section in an MQD file.

For advice on how to create a Process section, see "Defining the appearance of MQ" on page 66 and "Using an MQD file to change an existing installation" on page 68.

The following list explains the keywords you can use within the section:

Use this keyword	***To do this***
UseAfterDate=	Specify the date after which you want MQ to run the MQD file. The date must be in YYYY/MM/DD format. There is no default. If you do not specify a date, or you specify a nonvalid date, MQ logs a warning; MQ runs the MQD file next time MQ starts or when the user instructs it.
UseAfterTime=	Specify the time (on the UseAfterDate day) after which you want MQ to run the MQD file. The time must be in HH:MM 24-hour format. If you omit this keyword, the default value is 00:00; that is, MQ runs the file next time MQ starts on or after the date specified in the UseAfterDate keyword. MQ ignores this keyword if you do not specify a date using the UseAfterDate keyword.
ShowIcon=	Specify whether the MQ status icon appears on the Windows taskbar. Specify either `yes` or `no`. The default value is `yes`.
PromptForConnection=	Specify whether to show the Connections dialog when MQ starts. Specify either `yes` or `no`. The default value is `no`. If you specify `yes`, MQ displays the Connections dialog every time it starts. In this dialog, the user can select which connection they want to start. For more information, see Table 7 on page 67.

If you specify `no`, MQ automatically starts the connection you have defined it should start (see the AutoStart keyword in "The Connection section" on page 73). If there is no such connection, MQ does not start a connection. For more information, see Table 8 on page 67.

Password= Specify or change the administration-access password.

A user of the Compact version must type this password before they can see the Administration page of the MQSeries Properties dialog box. If you do not specify a password, the user can use the administration page without typing a password. If you omit this keyword, any existing password still applies. If you want to remove an existing password, specify `Password=<>`

The password is case sensitive, so if your application user has to type the password, make sure they type it exactly as you specify it in this keyword.

The maximum length of the password is 12 characters.

When the Complete version processes the MQD file, it ignores this keyword. Users of the Complete version can use administration features without entering a password.

RefreshRate= Specify the interval (in seconds) between automatic refreshes of the status of MQ components in the MQSeries Properties dialog box.

Specify one of the following values:

- 0
- 5
- 30
- 60
- 600

To prevent automatic refreshes of the component status information, specify `RefreshRate=0`. When you do this, the user must use the F5 function key to manually refresh the status information.

If you specify any value other than one of those listed, MQ does not change the refresh interval. It logs a warning in the MQD log file.

The default value is 600 seconds.

UseSmallIcons= Choose whether to display small icons on the pages of the MQSeries Properties dialog box.

Specify either `yes` or `no`. The default value is `yes`.

If you specify the value `no`, MQ uses large icons.

EnableLANVerify=	Choose whether to allow the user to verify the operation of MQ using a remote server.
	Specify either `yes` or `no`. The default value is `no`.
	If you specify the value `no`, or you omit this keyword, the option to verify MQ using a remote server is not available to the user.
	For more information, see "Configuring for verification using a LAN" on page 102.
ChannelLogSize=	Specify the maximum number of entries the channel log can hold.
	You can specify any number, but note that each entry in the log file is about 300 bytes long. The default value for this keyword is 100.
	If you want to stop channel logging, specify the value 0 for this keyword. If you change the value of this keyword, MQ discards the current log.

Example Process sections

The first example specifies that MQ runs the MQD file only after 9 p.m. on 30 November 1997. It sets the administration password to `Hursley`, so after 9 p.m., the user of the Compact version must use this new password to gain access to the administration features. The example also makes MQ display its status icon on the Windows taskbar, and prompts the user to choose a connection when MQ starts.

```
[Process]
UseAfterDate=1997/11/30
UseAfterTime=21:00
ShowIcon=yes
PromptForConnection=yes
Password=Hursley
```

The second example removes the existing password. It shows the MQ status icon on the Windows taskbar and does not prompt the user to choose a connection when MQ starts; these are the default values.

```
[Process]
Password=<>
```

The QueueManager section

To define a queue manager, use a section that starts with the lines:

```
[Component_n]
ComponentType=QueueManager
```

Note: You can define zero or more QueueManager sections in the MQD file, but to run an MQSeries application, there must be one queue manager running. So you should always define a QueueManager component in an MQD file, unless you are using the file to make changes to an existing installation that already has a queue manager defined.

For advice on how to create a QueueManager section, see "Defining a queue manager" on page 65.

The following list explains the keywords you can use within the section:

Use this keyword	***To do this***
Name=	Specify the name of the queue manager you want to create. The name of the queue manager can have a maximum of 48 characters. If you use a nonvalid character, MQ replaces the character with a period (.) and logs a warning. For more information on the characters you can use in the name of a queue manager, see "Naming MQ objects" on page 96. The name of the queue manager must be unique on the workstation. If you want each of your users to have a queue manager with a different name, but all the other definitions are the same, you can still create only one MQD file. Specify the Name keyword in the following way: `Name=?` Now, when the MQD file runs on the user's workstation, MQ prompts them to type the name they want to use for their own queue manager. You can control the wording of the dialog that the user sees by using the NamePrompt and NameInformationText keywords.

If you do not specify a name, MQ selects the first name that has not already been used from the following list:

```
QueueManager
QueueManager_1
QueueManager_2
  ⋮
QueueManager_n
```

NamePrompt= Specify the text that appears above the Queue Manager Name field if the user has to type the name.

This is valid only when you specify `Name=?` in the Name keyword. You can use a maximum of 60 characters.

The default text is:

```
Enter the name of the queue manager
```

NameInformationText= Specify the text that appears in the information line at the bottom of the window if the user has to type the name of the queue manager.

This is valid only when you specify `Name=?` in the Name keyword. You can use a maximum of 60 characters.

The default text is all blanks.

Description= Describe the queue manager.

Use this keyword to add a text description of the queue manager. You can use a maximum of 64 characters.

The default value is all blanks.

LoadSamplesMQSC= Specify whether to load the MQSC command file for the sample programs when the queue manager is created.

Specify either `yes` or `no`. The default value is `no`.

The MQSC command file for the sample programs is named AMQSCOSW.TST.

LoadUserMQSC_*n*= Specify the name of an MQSC command file you want loaded when the queue manager is created.

You can specify this keyword many times, substituting successive integers for *n*, starting with 1. The entries are read in numeric order.

You can use these MQSC command files to define MQSeries objects (such as queues and channels) that you want MQ to create when it creates the queue manager. You can also use these commands to change or delete MQ objects.

QueueManager section

If MQ cannot find the specified file, it logs an error and continues by processing the next component.

If an error occurs while running an MQSC command file, MQ puts an entry in the MQD log file, and that entry refers you to an MQSC log file.

Replace=
If a queue manager of this name already exists, specify whether to replace it with this one.

Specify either `yes` or `no`. The default value is `no`.

Note: If you specify `Replace=yes`, all the queues, channels, and messages associated with the first queue manager are destroyed. If you have defined any queues and channels in an MQSC command file that you specify in the LoadUserMQSC_*n* keyword, those queues and channels are created for the new queue manager.

Delete=
Specify whether to delete the queue manager defined by the Name keyword.

Specify either `yes` or `no`. The default value is `no`. If you specify `yes`, you must also specify `ConfirmDelete=yes`; this is to attempt to prevent accidental deletions.

If you want to delete a queue manager, you must first delete the connections that use it. To do this, give the sections of the MQD file that delete the connections a lower component number than the section that deletes the queue manager.

If you specify both `Delete=yes` and `ConfirmDelete=yes`, the queue manager specified by the Name keyword is deleted and the other keywords in the section are ignored. If the queue manager does not exist, a warning is logged and MQ continues processing the remainder of the MQD file.

ConfirmDelete=
Specify whether to confirm that you want to delete the queue manager defined by the Name keyword.

Specify either `yes` or `no`. The default value is `no`.

If you specify both `Delete=yes` and `ConfirmDelete=yes`, the queue manager specified by the Name keyword is deleted and the other keywords in the section are ignored. If the queue manager does not exist, a warning is logged and MQ continues processing the remainder of the MQD file.

Example QueueManager components

The first example defines a queue manager named MY_QUEUE_MANAGER using the default values. If there is already a queue manager of this name, MQ ignores this section.

```
[Component_1]
ComponentType=QueueManager
Name=MY_QUEUE_MANAGER
```

The second example is from MARS.MQD, the sample file that creates the components you can use to run the sample programs across two workstations (see Chapter 13, “Running the sample programs on two workstations” on page 133). When MQ creates the queue manager, it runs the file MARS.TST. This file defines the objects the queue manager needs when it communicates with the queue manager named VENUS. If there is already a queue manager named MARS on the workstation, MQ replaces it with this one.

```
[Component_1]
ComponentType=QueueManager
Name=MARS
Description=Queue manager to communicate with VENUS
LoadUserMQSC_1=\Program Files\MQSeries for Windows\Samples\MARS.TST
Replace=yes
```

The third example defines a queue manager using the name the user supplies. When MQ creates the queue manager, it displays a window in which the user is prompted to type the name of the queue manager they have been given by their head office. MQ replaces an existing queue manager of the same name.

```
[Component_1]
ComponentType=QueueManager
Name=?
NamePrompt=Type the name given to you by Head Office
NameInformationText=Type the name, then press OK
Description=Queue manager for payroll application
LoadUserMQSC_1=A:\PAYROLL.TST
Replace=yes
```

QueueManager section

The last example deletes MY_QUEUE_MANAGER:

```
[Component_1]
ComponentType=QueueManager
Name=MY_QUEUE_MANAGER
Delete=yes
ConfirmDelete=yes
```

Chapter 9. Working with the components of MQSeries for Windows

This chapter describes the following tasks:

- "Creating MQ components individually" on page 88
- "Deleting MQ components individually" on page 97
- "Viewing and changing object attributes" on page 98
- "Viewing the status of an MQ object" on page 101
- "Configuring for verification using a LAN" on page 102

There are two ways to create, change, and delete MQ components:

- You can define components in a definition (MQD) file. If this file has changed since the last time MQ was started, MQ runs the file automatically when it starts. Alternatively, you can run the file at any time using the **Run MQD file now** item on the menu of the MQ icon on the Windows taskbar.

  You can use an MQD file to create new components, to replace existing components, and to delete components. Use this method if you want to create the same components many times while you are testing an application, or if you want many users to use identical components on their own workstations without them having to define the components themselves. You can give each of your users a copy of the MQD file; when MQ runs this file, it performs all the definitions you specify in the file.

  For information on how to create a definition file, see Chapter 8, "Creating an MQD file" on page 57.

- If you have installed the Complete version of MQSeries for Windows, you can use the Components page of the MQSeries Properties dialog box to create, change, or delete components individually. Use this method if you want to work on the components on your own workstation. Only MQ administrators are advised to use this method.

  For information on how to use the MQSeries Properties dialog box, see:

  – "Creating MQ components individually" on page 88
  – "Deleting MQ components individually" on page 97

In addition, you can work with some MQ objects using MQSC and programmable command format (PCF) commands:

- You can issue MQSC commands either by:

  – Running an MQSC command file when you create the queue manager using the MQSeries Properties dialog box (see "Creating a queue manager" on page 88).

  – Using the MQSC page of the MQSeries Properties dialog box (this allows you to enter commands individually or by running an MQSC command file). To find out more about the MQSC page, see Chapter 10, "Using MQSC commands" on page 107.

- You can use PCF commands in an administration application in the same way you do for other MQ products. For a list of the PCF commands that MQSeries for Windows supports, see “PCF commands supported by MQSeries for Windows” on page 121. For information on how to use PCF commands, see the *MQSeries Programmable System Management* manual.

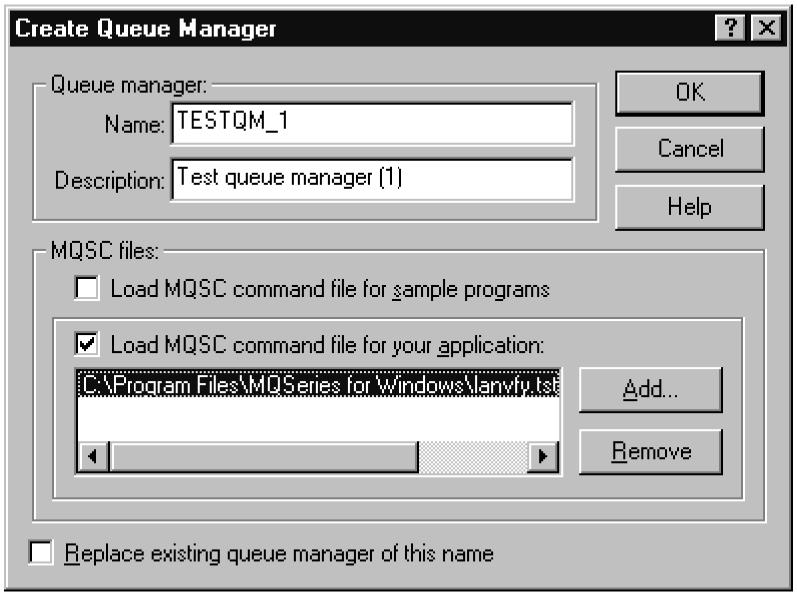
Creating MQ components individually

To create MQ components individually, use the Components page of the MQSeries Properties dialog box. This page appears only if you have installed the Complete version.

Note: If you have a connection running, you must stop it before you try to create a new component.

Creating a queue manager

To create a queue manager, open the Components page of the MQSeries Properties dialog box and click on the **Create** push button. Select **Queue Manager** from the menu. MQ opens the window shown in Figure 11.



Name: TESTQM_1

Description: Test queue manager (1)

☐ Load MQSC command file for sample programs

☑ Load MQSC command file for your application:

C:\Program Files\MQSeries for Windows\lanvfy.tst

☐ Replace existing queue manager of this name

Figure 11. The Create Queue Manager window

Complete the fields in the Create Queue Manager window as follows:

Queue Manager Name
Specify the name of the queue manager you want to create.

When you run MQSeries applications, you need to use this name to identify this queue manager. You can use a maximum of 48 characters. For more information on the characters you can use in the name of a queue manager, see “Naming MQ objects” on page 96.

Note: The name of a queue manager is case sensitive, so if you type the name in uppercase letters when you create the queue manager, you must always use uppercase letters whenever you type the name.

Queue Manager Description
Specify a text description of the function or purpose of this queue manager.

This field is optional. If you do not give a description, all subsequent windows show: `(none)`. Remember that, at a later date, a description could help you to identify the queue manager you want to work with. You can change it at any time by changing the attributes of the queue manager (see “Changing the attributes of a queue manager, queue, or channel” on page 101).

You can type any text you like in this field.

Load MQSC Command File for Sample Programs
If you want to run the supplied sample programs after you have created your queue manager, make sure this check box is marked. When MQ creates the queue manager, it runs the MQSC command file that generates the queues used by the sample programs.

The sample file is named AMQSCOSW.TST; for more information on it, see “Objects for running the sample programs on one workstation” on page 204.

Load MQSC Files for your Application
Select the names of any MQSC command files you want to run when you create the queue manager.

This field is optional. If you are using MQ for the first time, or just running the samples, you can leave this field blank.

You can use one or more MQSC command files to create other MQ objects (such as queues and channels). To find out more about MQSC command files, see Chapter 10, “Using MQSC commands” on page 107.

If there is an existing queue manager with the name you have chosen, you can replace it by checking the Replace box in the Create Queue Manager window.

When you have completed the input fields in this window, click on **OK**. When MQ has created the queue manager, it displays a confirmation message.

When you create a queue manager, it always runs one MQSC command file automatically. This is AMQSCOMW.TST, which defines default and system queues and channels. For more information on it, see “Default and system objects” on page 203.

If an error occurs when any of the MQSC files are run, you are prompted to look at the MQSC log to find out more about the error. MQ creates the log file in a separate directory for each queue manager. For example, for a queue manager named TEST, the log file is \Program Files\MQSeries for Windows\QMgrs\TEST\MQSC.LOG (if you installed MQSeries for Windows in the default directory). For information about these error messages, see Appendix E, “Error messages” on page 213.

After you have created the queue manager, you must add it to an MQ connection. You can do this either:

- When you create the connection (see “Creating a connection” on page 92)
- Or by changing the attributes of an existing connection (see “Changing the attributes of an MQ connection” on page 100).

If you do not do this, you cannot see or work with the queue manager using the MQSeries Properties dialog box.

Creating a channel group

You must create a channel group if you want to create a LAN or dial-up connection.

To create a channel group, use the Components page of the MQSeries Properties dialog box. Remember that a channel group contains channels, so you must create the channels first. Typically you will create the channels using an MQSC command file and specify the name of that file when you create the queue manager. If you do not do this, you must create the channels as described in “Creating a channel” on page 95. However, you can create a channel group that contains just the listener.

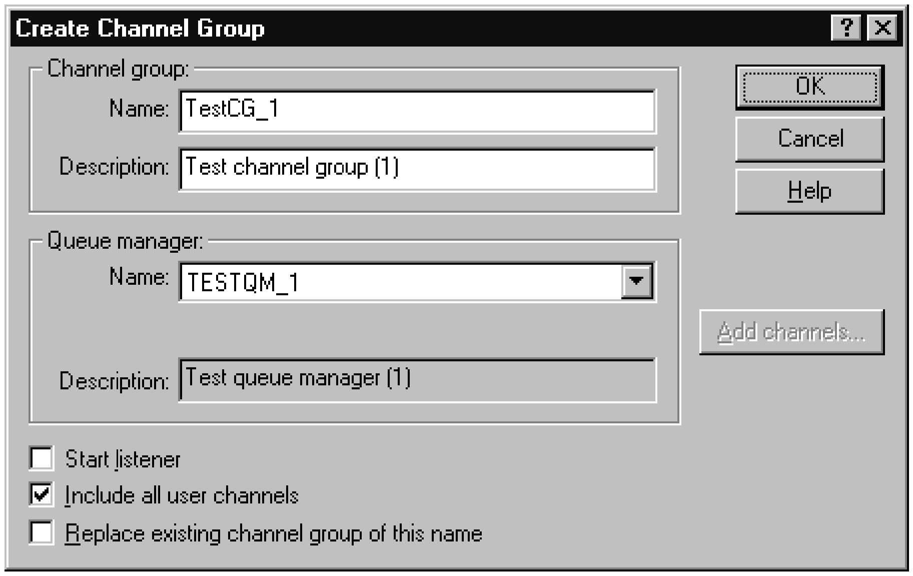
Open the Components page of the MQSeries Properties dialog box and click on the **Create** push button. Select **Channel Group** from the menu. MQ opens the window shown in Figure 12 on page 91.

When you complete the fields in the Create Channel Group window, you must:

- Specify the name of the channel group you want to create.
- Select the queue manager for which you want to create a channel group. This queue manager will own the channel; each queue manager can own a maximum of 16 channel groups.

For more information on the characters you can use in the name of a channel group, see “Naming MQ objects” on page 96.

To select which existing channels you want to add to the channel group, click on the **Add Channels** push button. Note you cannot add receiver channels to a channel group.



Channel group:
Name: TestCG_1
Description: Test channel group (1)

Queue manager:
Name: TESTQM_1
Description: Test queue manager (1)

☐ Start listener
☑ Include all user channels
☐ Replace existing channel group of this name

Figure 12. The Create Channel Group window

Alternatively, you can automatically add all the user-defined channels the queue manager owns. This excludes the receiver channels and the MQ default channels. This saves you having to add individual channels to the channel group.

You can have a maximum of 8 channels (or 7 and the listener) in a channel group.

If you need to add the listener to this channel group, make sure you mark the **Start Listener** check box.

If there is an existing channel group with the name you have chosen, you can replace it by checking the Replace box in the Create Channel Group window.

After you have created the channel group, you must add it to an MQ connection. You can do this either:

- When you create the connection (see “Creating a connection” on page 92)
- Or by changing the attributes of an existing connection (see “Changing the attributes of an MQ connection” on page 100).

If you do not do this, you cannot see or work with the channel group using the MQSeries Properties dialog box. Note that a connection can include only one channel group.

If you want to add more channels to the group after you have created it, see “Changing the attributes of a channel group” on page 100.

Creating components

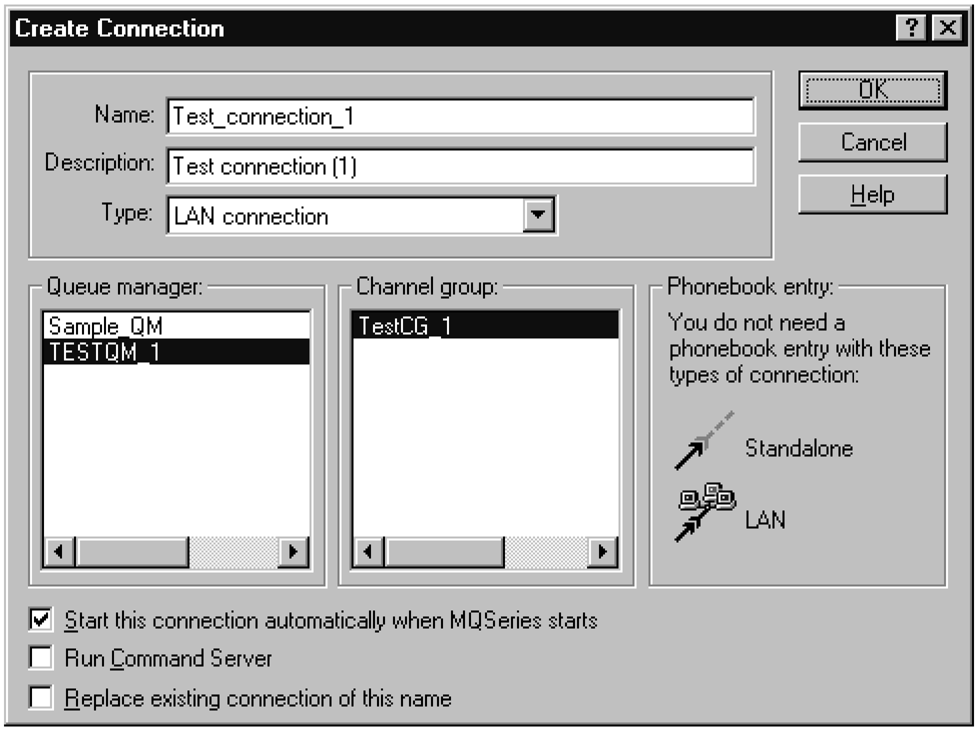
Creating a connection

Before you create a connection, you must:

- For a standalone connection, create a queue manager
- For a LAN connection, create a queue manager and a channel group
- For a dial-up connection, create a queue manager, a channel group, and a phonebook entry

A dial-up connection uses the built in dial-up networking support provided by the Windows operating system. If this is not installed on you workstation, you cannot create a dial-up connection. You must first install the dial-up networking support, then use it to create what Windows calls *dial-up networking connections*; MQ calls these *phonebook entries*. These entries are like a paper phone book; they relate a person's name to a telephone number. When you have defined the phonebook entry, you can use the name to access the telephone number.

To create a connection, open the Components page of the MQSeries Properties dialog box and click on the **Create** push button. Select **Connection** from the menu. MQ opens the window shown in Figure 13.


Create Connection

Name: Test_connection_1
Description: Test connection (1)
Type: LAN connection

Queue manager:
Sample_QM
TESTQM_1

Channel group:
TestCG_1

Phonebook entry:
You do not need a phonebook entry with these types of connection:
Standalone
LAN

[✓] Start this connection automatically when MQSeries starts
[ ] Run Command Server
[ ] Replace existing connection of this name

Figure 13. The Create Connection window

Complete the fields in the Create Connection window as follows:

Connection Name
Specify the name of the connection you want to create.

When you run MQSeries applications, you need to use this name to identify this connection. You can use a maximum of 48 characters. For more information on the characters you can use in the name of a connection, see “Naming MQ objects” on page 96.

Note: The name of a connection is case sensitive, so if you type the name in uppercase letters when you create the connection, you must always use uppercase letters whenever you type the name.

Connection Description
Specify a text description of the function or purpose of this connection.

This field is optional. If you do not give a description, all subsequent windows show: `(none)`. Remember that, at a later date, a description could help you to identify the connection you want to work with. You can change it at any time by changing the attributes of the connection (see “Changing the attributes of an MQ connection” on page 100).

You can type any text you like in this field.

Connection Type
Select from the list the type of connection you want to create:

Standalone Use a standalone connection when you want to work with MQ without establishing a connection to another queue manager. For example, you use a standalone connection when you just want to enter data without sending it to another computer. You might do this when you are working with a laptop computer in a customer’s office.

LAN Use a LAN connection when you want to communicate with another queue manager that is running on a computer you can access through a local area network (LAN). For example, you could use a LAN connection when you want to use the application in your own office.

Dial-up Use a dial-up connection when you want to communicate with another queue manager that is running on a computer you can access through a dial-up telephone link. For example, you could use a dial-up connection when you want to use the application in your own home or in a hotel room.

You must always select the queue manager you want to use for the new connection (select a queue manager from the list of those available). In addition:

- If you are creating a LAN connection, you must select a single channel group from the list of those available.
- If you are creating a dial-up connection, you must select a single channel group and a single phonebook entry from the lists of those available.

Creating components

You can check a box to specify that your new connection is started automatically whenever MQ starts. But there can be only one connection marked for automatic starting; if there is already a connection marked, and you mark the new connection you create, MQ removes the mark from the previous connection. When MQ starts, and you have chosen the option to select a connection, MQ displays the Connections window and highlights the connection you have marked for automatic starting.

You can check a box to specify that MQ runs the command server automatically whenever this connection is active. The command server runs on the queue manager that the connection owns. The command server stops when the connection ends. But the command server is a heavy user of system resources, so set it to start automatically only if you want to use it regularly. If you want to use the command server only occasionally, start it using the MQSeries Properties dialog box when you are ready. You can stop the command server at any time using the MQSeries Properties dialog box.

If there is an existing connection with the name you have chosen, you can replace it by checking the Replace box in the Create Connection window. If you want to change the composition or properties of a connection after you have created it, see “Changing the attributes of an MQ connection” on page 100.

Creating a queue

This section tells you how to create a queue using the MQSeries Properties dialog box. You can also create queues using MQSC and PCF commands.

To create a queue using the MQSeries Properties dialog box, open the Components page and click on the **Create** push button. Select **Queue** from the menu. MQ opens the window shown in Figure 14 on page 95.

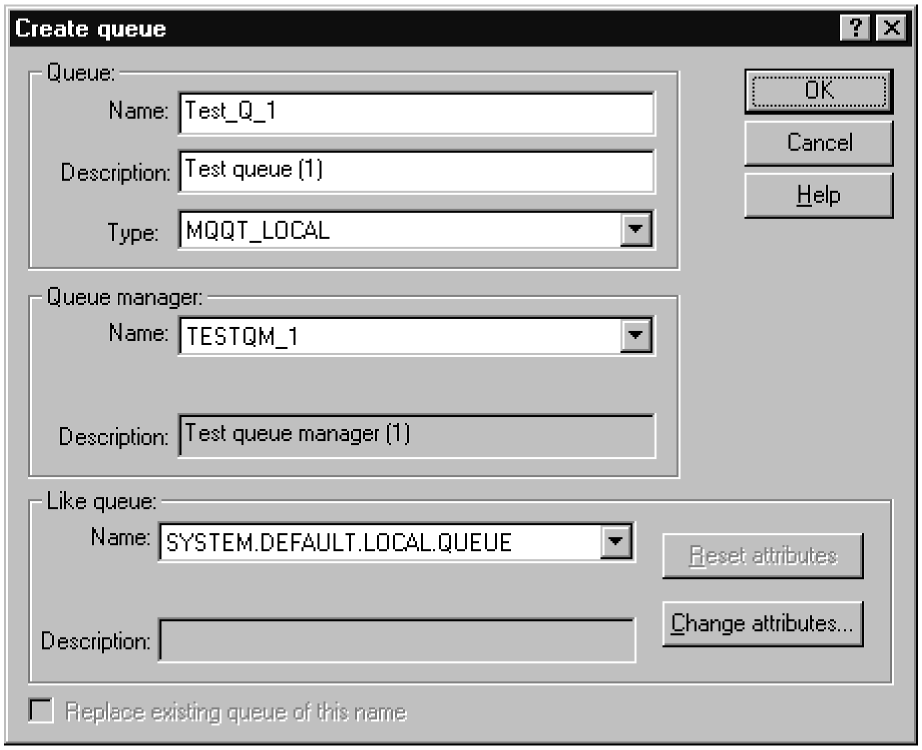
When you complete the fields in the Create Queue window, you must:

- Specify the name of the queue you want to create.
- Specify the type of queue you want to create.
- Select the queue manager for which you want to create a queue.
- If you want to set any attributes for the new queue, click on the **Change attributes** push button.
- Select an existing queue from which MQ can copy any queue attributes you do not specify. You will usually use the system default queue for this.

For more information on the characters you can use in the name of a queue, see “Naming MQ objects” on page 96.

If there is an existing queue with the name you have chosen, you can replace it by checking the Replace box in the Create Queue window.

Note: You cannot see the queue in the MQSeries Properties dialog box until you start the connection that uses it.


Create queue

Queue:

Name: Test_Q_1

Description: Test queue (1)

Type: MQQT_LOCAL

Queue manager:

Name: TESTQM_1

Description: Test queue manager (1)

Like queue:

Name: SYSTEM.DEFAULT.LOCAL.QUEUE

Description:

☐ Replace existing queue of this name

Figure 14. The Create Queue window

Creating a channel

This section tells you how to create a channel using the MQSeries Properties dialog box. You can also create channels using MQSC and PCF commands.

The procedure for creating a channel using the MQSeries Properties dialog box is similar to that for creating a queue. Open the Components page and click on the **Create** push button. Select **Channel** from the menu, then complete the fields in the Create Channel window. You must:

- Specify the name of the channel you want to create
- Specify the type of channel you want to create
- Select the queue manager for which you want to create a channel (this queue manager will own the channel)
- If you want to set any attributes for the new channel, click on the **Change attributes** push button. For some types of channel, you must specify some attributes; see Table 9 on page 96.
- Select an existing channel from which MQ can copy any channel attributes you do not specify. You will usually use the system default channel for this.

Creating components

For more information on the characters you can use in the name of a channel, see "Naming MQ objects" on page 96.

If there is an existing channel with the name you have chosen, you can replace it by checking the Replace box in the Create Channel window.

Note: You cannot see the channel in the MQSeries Properties dialog box until you start a connection that uses it.

Table 9. The channel attributes you must always set

Channel type	Required attributes
Receiver	None
Requester	ConnectionName
Sender	ConnectionName and XmitQName
Server	XmitQName

Naming MQ objects

Table 10 shows the maximum number of characters you can use in the name when you create MQ objects.

Table 10. Maximum number of characters in the names of MQ objects

MQ object	Maximum number of characters
MQ connection	48
Channel group	48
Queue manager	48
Queue	48
Channel	20

You can use any of the following characters in the name:

- Uppercase letters A through Z
- Lowercase letters a through z
- Numerics 0 through 9
- Period (.)
- Underscore (_)
- Forward slash (/) (see Note 1)
- Percent (%) (see Note 1)

Notes:

1. Forward slash (/) and percent (%) are special characters. If you use either of these characters in a name, you must enclose the name in double quotation marks whenever you use it.
2. You cannot use leading or embedded blank characters.
3. You cannot use national language characters.

Deleting MQ components individually

Each MQ component you create uses system resources, so you should delete a component you no longer use. However, deleting some components has an effect on others, so read through the following sections before you delete a component:

- "Deleting a connection"
- "Deleting a queue manager"
- "Deleting a queue"
- "Deleting a channel"
- "Deleting a channel group" on page 98

Deleting a connection

When you delete an MQ connection, no other MQ components are affected. However, this may leave a queue manager or channel group on your workstation that is not owned by a connection. You cannot work with these components using the MQSeries Properties dialog box until you create a new connection that uses these components.

Deleting a queue manager

You cannot delete a queue manager if it is owned by a connection. If you want to delete the queue manager, you must first change the attributes of its owning connection so that the connection owns another queue manager. Alternatively, you could delete the connection.

When you delete a queue manager, any queues and channels it owns are deleted, and the messages on the queues are deleted.

Deleting a queue

When you delete a queue, be warned that the messages on it are deleted.

Deleting a channel

If there is only one channel in a channel group and the group does not include the listener, MQ deletes the channel group if you delete that channel. This is because the channel group is now empty.

Changing attributes

Deleting a channel group

You cannot delete a channel group if it is owned by a connection. If you want to delete the channel group, you must first change the attributes of its owning connection so that the connection no longer owns the channel group. Alternatively, you could delete the connection.

When you delete a channel group, the channels that belong to the group are ***not*** deleted.

How to delete a component

To delete a component:

1. Open the Components page of the MQSeries Properties dialog box.
2. Click on the **Delete** push button.
3. Select from the menu the type of component you want to delete.
4. In the resulting Delete window, select the components you want to delete. You can select as many components as you want. Figure 15 on page 99 shows the Delete Queue window.
5. You are prompted to confirm that you want to delete the components.

When the component has been deleted, you can reuse its name when you create another component.

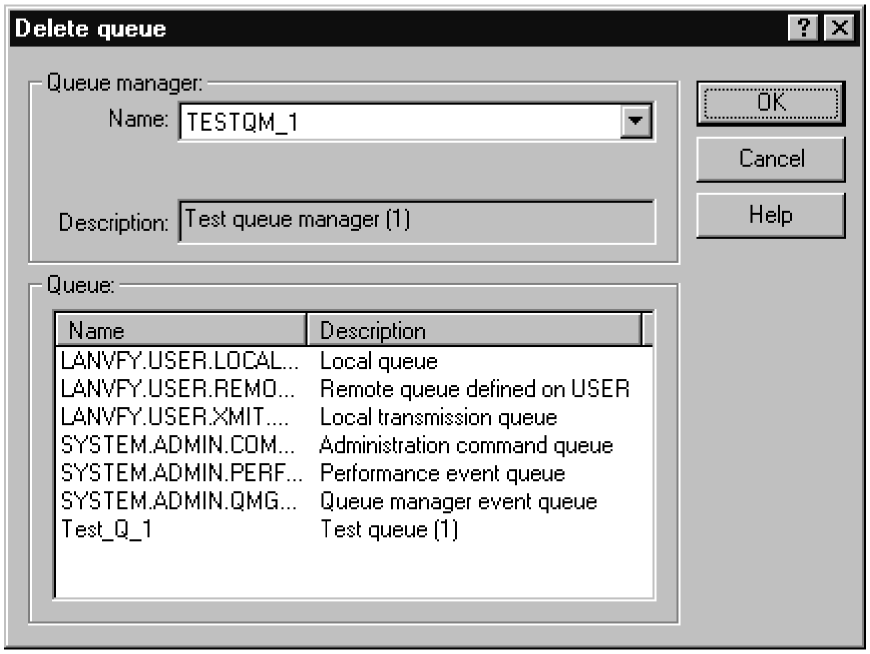
If you uninstall MQSeries for Windows, all MQ components are deleted.

Viewing and changing object attributes

On the Compact version of MQSeries for Windows, your application users can see a tree view that shows the status of all the MQ components defined on their workstations. If they enable the administration features of the product, they can also view and change the attributes of MQ objects. On the Complete version of MQSeries for Windows, you can change attributes directly from the Components page of the MQSeries Properties dialog box.

All MQ components have attributes (or properties) that control their behavior. To see the attributes of a component, select the component on the Components page of the MQSeries Properties dialog box and click on the **Attributes** push button. In the resulting window you can select each attribute to see its value. For more information on these attributes, see the online help.

You can change some of the attributes of a component, but the connection must be in the correct state; see Table 11 on page 99.



Name: TESTQM_1

Description: Test queue manager (1)

Name	Description
LANVFY.USER.LOCAL...	Local queue
LANVFY.USER.REMO...	Remote queue defined on USER
LANVFY.USER.XMIT....	Local transmission queue
SYSTEM.ADMIN.COM...	Administration command queue
SYSTEM.ADMIN.PERF...	Performance event queue
SYSTEM.ADMIN.QMG...	Queue manager event queue
Test_Q_1	Test queue (1)

Figure 15. The Delete Queue window

Table 11. State of connection for changing component attributes

Component	**State of connection for changing component attributes**
Connection	Stopped
Channel group	Stopped
Queue manager	Active
Queue	Active
Channel	Active

Note: If you want to change the properties of a phonebook entry, you must use the dial-up networking features of the operating system.

For more information, see:

- "Changing the attributes of an MQ connection" on page 100
- "Changing the attributes of a channel group" on page 100
- "Changing the attributes of a queue manager, queue, or channel" on page 101

Changing attributes

Changing the attributes of an MQ connection

When an MQ connection is active, you can view (but not change) its attributes. When the connection is stopped, you can change:

- The description if you want to make the purpose of the connection clearer
- The queue manager that the connection uses
- The channel group that the connection uses
- The phonebook entry that the connection uses
- Whether the connection starts automatically when MQ starts
- Whether the command server runs when the connection is active

For example, if you want MQ to start a different connection automatically:

1. Stop the connection.
2. Open the Components page of the MQSeries Properties dialog box.
3. Select the connection you want to start automatically.
4. Click on the **Attributes** push button.
5. In the resulting window, select the Autostart attribute. The current value of this attribute is displayed.
6. Change this value to Yes.
7. Click on the **OK** push button.

MQ displays a message to confirm that it has changed the connection that is started automatically.

Changing the attributes of a channel group

When an MQ connection is active, you can view (but not change) the attributes of its channel group. When the connection is stopped, you can change:

- The description of the channel group if you want to make the purpose of the channel group clearer
- Which channels are included in the channel group
- Whether the listener belongs to the channel group

For example, if you want to add more channels to a channel group (up to the maximum of 8 channels, or 7 plus the listener), change the attributes of the channel group as follows:

1. Stop the connection.
2. Open the Components page of the MQSeries Properties dialog box.
3. Expand the tree view until you can see the channel group you want to change, then select the channel group.
4. Click on the **Attributes** push button.
5. In the resulting window, select the Channels attribute. The list of available channels is displayed.

6. Select all the channels you want in the channel group.
7. Click on the **OK** push button.

If there is only one channel in the channel group and the group does not include the listener, MQ deletes the channel group if you delete that channel. This is because the channel group is now empty.

Changing the attributes of a queue manager, queue, or channel

The attributes of queue managers, queues, and channels are part of the Message Queue Interface (MQI). For information on the attributes that MQSeries for Windows supports, see the online help. Also:

- The queue manager attributes are listed in Table 20 on page 181.
- The queue attributes are listed in Table 21 on page 182.
- The channel attributes are listed in Table 22 on page 183.

Before you change these attributes, you must start a connection that uses the queue manager you want to change.

For example, if you want to block a queue so that applications cannot put messages on it while you fix a problem:

1. Start a connection that uses the queue manager that owns the queue.
2. Open the Components page of the MQSeries Properties dialog box.
3. Expand the tree view until you can see the queue you want to change, then select the queue.
4. Click on the **Attributes** push button.
5. In the resulting window, select the InhibitPut attribute. The current value of this attribute is displayed.
6. Select the value PUT_INHIBITED from the list box.
7. Click on the **OK** push button.

Viewing the status of an MQ object

We have already seen two ways in which you can get information about an MQ object:

- The icons in the tree view on the Components page of the MQSeries Properties dialog box give an indication of the status of an object (they show you whether the object is active or stopped).
- The **Attributes** push button on the Components page of the MQSeries Properties dialog box shows you the values of the attributes of an object (such as the maximum message length of a queue).

In addition, MQ reports transient status information about its objects. An example of this status information is the number of bytes sent or received on a channel. To see this information, select the object on the Components page of the MQSeries Properties dialog box, then click the **Status** push button. In the resulting window, you can select each status attribute to see its value. For more information on these status attributes, see the online help. You cannot change these status attributes because they report the current state of the object.

Configuring for verification using a LAN

You may want your users to use the LAN option of the Verify function (see "Verifying your installation" on page 20) to confirm that their system is configured correctly to transmit messages to a queue manager on a server. Before they can do this, you must configure their workstation and the server for LAN-mode verification.

Configuring the workstation

You must configure the user's workstation to enable verification using a connection to a server on a LAN. You can do this on the installation media before you install MQSeries for Windows on the workstation.

1. Enable LAN verification in the user's MQD file. See the description of the EnableLANVerify keyword in "The Process section" on page 79.
2. Edit the supplied file LANVFY.TST to add the TCP/IP address of the server to which you want Verify to connect. You must add the address to the definition of both the USER.TO.ADMIN sender channel and the ADMIN.TO.USER requester channel.

   Alternatively, supply your own MQSC command file that defines the MQ objects you need. If you do this, you must set environment variable MQW_LANVFYPATH on the user's workstation to specify the name and location of this file. You could put this file on your server.

Configuring the server

To enable users to run Verify in LAN mode, you must configure a server to which Verify can connect. You can run any MQSeries queue manager product on this server, but you must configure it using the definitions the Verify program uses. If the server is running MQSeries for Windows, you must perform some extra configuration steps.

Configure the server as follows:

1. Create a queue manager.

   You must name the queue manager SYSTEM.LANVFY.ADMIN.QUEUE.MANAGER.
2. Create a transmission queue, a remote queue, a server channel, and a receiver channel.

   MQSeries for Windows supplies the object definitions you need in the MQSC command file named LANVFY.TST, but the definitions are commented out. Copy the commented-out section (labeled 'ADMINISTRATION machine') from this file into a new file, remove the comment characters, then run that file on the queue manager you have created on your server.

3. Start the listener or, if the server is running MQSeries for Windows, create a channel group that includes only the listener.

   When you create the channel group, you must:

   - Specify its owning queue manager as SYSTEM.LANVFY.ADMIN.QUEUE.MANAGER
   - Add the listener to it (for example, by marking the Start Listener box in the Create Channel Group window)

4. If the server is running MQSeries for Windows, create a connection.

   When you create the connection, you must:

   - Specify the name of its queue manager as SYSTEM.LANVFY.ADMIN.QUEUE.MANAGER
   - Specify the name of the channel group you have created

If you want to use objects with names that are different from those listed here, you must provide your own definitions. However, if you do this, you must make sure the objects are compatible with those defined on the user's workstation.

Objects required for verifying in LAN mode

Figure 16 shows the MQ objects you need for verifying in LAN mode. The supplied file LANVFY.TST creates the queues and channels. MQ creates the user's queue manager; you must create the server queue manager yourself.

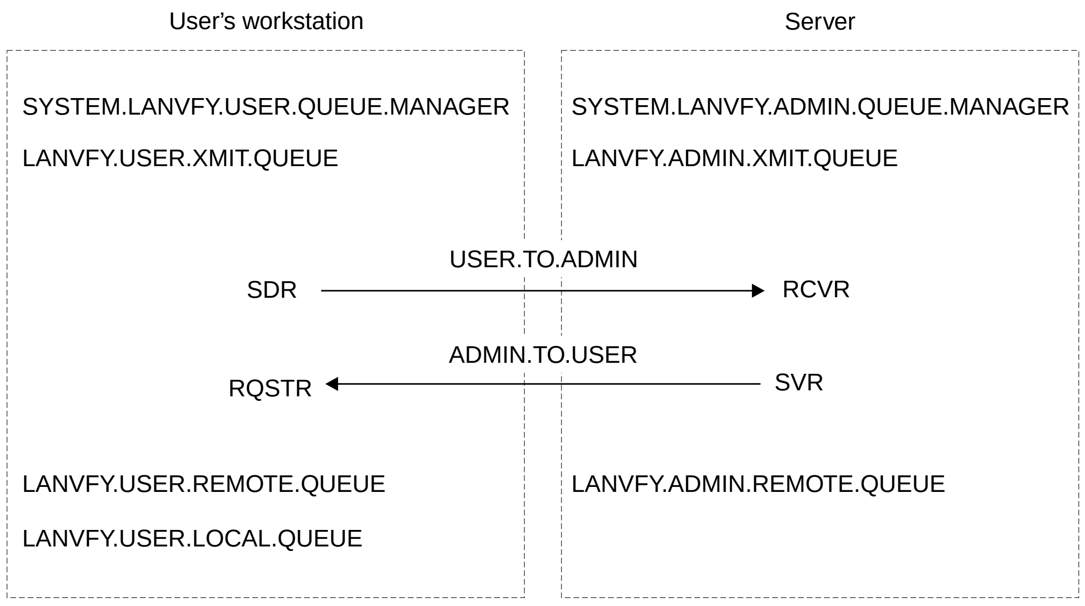


Figure 16. The objects defined in the file LANVFY.TST

LAN verification

How LAN-mode verification works

In LAN mode, Verify uses MQ objects defined in the MQSC command file named LANVFY.TST. Verify can run in LAN mode only if there is no active connection.

In LAN mode, Verify:

1. Creates the test connection (unless it exists already)
2. Starts the connection
3. Opens the remote queue
4. Puts a test message on the remote queue
5. Closes the remote queue
6. Opens the local queue
7. Gets the message from the local queue
8. Closes the local queue
9. Stops the connection

The queues are defined so that the remote queue on which Verify puts the test message resolves to a queue on the remote server. The queue on the server is itself a remote queue which resolves to a local queue on the user's workstation. This means that when Verify puts the test message on the remote queue, MQ moves the message from the workstation to the server, then back to the local queue on the workstation.

If LAN-mode verification fails

The most likely reasons for failure of a LAN verification are:

- The TCP/IP connection to the server is broken.

  You can confirm this by trying to ping the workstation from the server. You must reestablish this connection before Verify can send messages to the server.

  If the workstation cannot connect to the server, look at the errors recorded in the channel log. To do this, open the Channel Logs tab on the Service page of the MQSeries Properties dialog box, then select the SYSTEM.LANVFY.USER.QUEUE.MANAGER queue manager.

- A queue manager named SYSTEM.LANVFY.ADMIN.QUEUE.MANAGER already exists on the server.

  If you have created a queue manager of this name, you must delete it before you run Verify.

- A queue manager named SYSTEM.LANVFY.USER.QUEUE.MANAGER already exists on the workstation.

  If you have created a queue manager of this name, you must delete it before you run Verify.

- A connection named SYSTEM.LANVFY.USER.CONNECTION already exists on the workstation.

  If you have created a connection of this name, you must delete it before you run Verify.

- The server is not set up correctly for LAN verification.

LAN verification

Make sure you set up the server as described in “Configuring for verification using a LAN” on page 102, using either the supplied file LANVFY.TST or your own object definitions.

- Verify cannot get messages from the queue.

  Try increasing the wait interval on the Verify tab of the Service page of the MQSeries Properties dialog box.

- If you still find problems:

  1. Delete the queue manager named SYSTEM.LANVFY.USER.QUEUE.MANAGER from the workstation.
  2. Reset the sequence number at the server end of the channel by issuing the MQSC command RESET CHANNEL on the server.
  3. Run the verification again.

LAN verification

Chapter 10. Using MQSC commands

MQSeries provides commands (known as *MQSC commands*) with which you can create, change, and delete MQSeries objects. This chapter describes how to use those commands on MQSeries for Windows. It contains the following sections:

MQSC commands are common across all MQSeries platforms, but MQSeries for Windows does not support all the commands available on other MQSeries products:

- If you are used to using MQSC commands on other operating systems, see "MQSC commands supported by MQSeries for Windows" on page 116 to see if the commands you want to use are available on Windows.
- For a description of each MQSC command you can use on MQSeries for Windows, see the online *MQSeries for Windows Command Reference*.

Issuing MQSC commands

To issue MQSC commands, use the MQSC page of the MQSeries Properties dialog box. This page performs and extends the functions of the RUNMQSC command that other MQSeries products provide. You can type MQSC commands one at a time, recall commands you have issued previously, or you can save many commands in a file and run that file.

Note: The MQSC page is grayed out if there is no active connection. You must start a connection that uses the queue manager on which you want to issue your MQSC commands. If you are using the Compact version, you may find that the Administration page of the MQSeries Properties dialog box is protected by a password (it is in a default installation).

In addition, you can run MQSC command files:

- When you create a queue manager using the Components page of the MQSeries Properties dialog box of the Complete version
- When you use an MQD file to create a queue manager (list the names of the command files in the MQD file)

Specifying MQSeries object names

When you are issuing MQSC commands, you need specify only the local name of the object on which you are operating. In the examples showing MQSC commands in this chapter, the queues have names such as `SAMPLE.VENUS.REMOTE`. They have this format simply to show the function of the queue. You do not need to use names of this form, but you are recommended to use a naming scheme of some sort.

MQSC commands

You can write MQSC commands and their attributes in uppercase or lowercase. But the names of MQSeries objects are case sensitive, so always ensure you type them in the same way in which they were defined.

Quotation marks are significant in the names of MQSeries objects:

- If you enclose a name in single quotation marks, MQ leaves the name as it is.
- If you ***do not*** enclose a name in single quotation marks, MQ changes the name to uppercase.

So, for example, MQ treats the following as the same name:

```
hursley
Hursley
HURSLEY
```

But MQ treats the following as different names:

```
'hursley'
'Hursley'
'HURSLEY'
```

The MQSC page

The MQSC page of the MQSeries Properties dialog box allows you to type MQSC commands and to run MQSC command files. The page is shown in Figure 17 on page 110. You can:

Edit and rerun a command
If a command results in an error, you can edit the command and rerun it without having to type the whole command again.

Run an MQSC command file
You can run an MQSC command file. If there are any errors, they are displayed on the MQSC page. You can then correct them and run the file again.

Use the Windows clipboard
You can use the clipboard to transfer commands to and from other files. You can save a complex command in the clipboard so that you can load it next time you use MQSC (provided you have not added something else to the clipboard or restarted your workstation).

Recall commands
MQ stores the last 32 successful commands until you close the MQSeries Properties dialog box. You can recall these commands using the scroll buttons in the Command window, then rerun any of them.

Log commands in a window
MQ logs in the Results window of the MQSC page the result of each command it processes, and a summary of every MQSC file it runs. There is one log message for each command issued and each file run. Each log message also contains an icon, which provides a visual representation of the success or failure of the command.

Log commands in a file
MQ logs in a file all the commands it processes from MQSC command files. The log file is called MQSC.LOG and it is stored in the directory of the active queue manager. Each command is appended to the end of the existing log file. To view the log file, click the **View MQSC.LOG** push button on the MQSC page of the MQSeries Properties dialog box.

Save user preferences
When you stop using MQSC, MQ saves the name of the directory containing the last MQSC file you ran.

Look up descriptions of MQSC commands
The MQSC commands that MQSeries for Windows supports are described in the online *MQSeries for Windows Command Reference*. If you are using the Complete version, you can open this book by clicking the **Open Command Reference** push button on the MQSC page of the MQSeries Properties dialog box.

Differences from other platforms

The main differences between the MQSC page of the MQSeries Properties dialog box and the RUNMQSC utility provided by other MQSeries products are:

- You cannot specify the name of a queue manager to run the MQSC commands against. This is because MQSeries for Windows allows only one queue manager to run at a time, so it runs your commands against the active queue manager.
- There are some small differences in how you must write commands in an MQSC command file; see "The format of MQSC files" on page 112.
- You cannot issue MQSC commands on an MQSeries for Windows queue manager to run on another queue manager (known as indirect mode). Also, you cannot issue MQSC commands on another queue manager (using the `runmqsc /w` command) to run on an MQSeries for Windows queue manager.

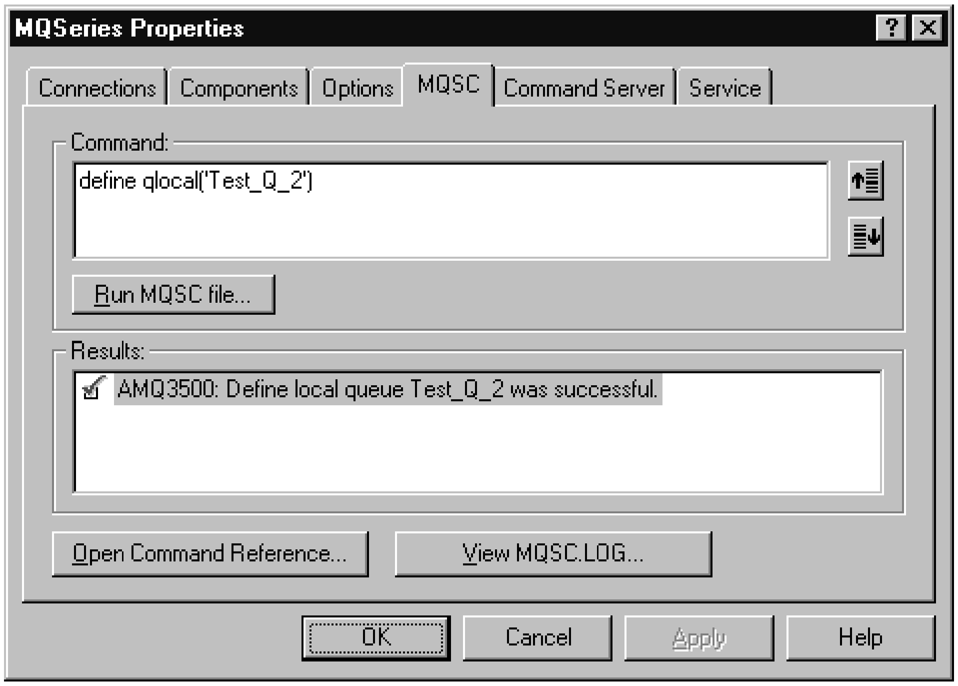
MQSC commands

Issuing MQSC commands

You can issue MQSC commands either interactively (by typing them in a field of the MQSC page of the MQSeries Properties dialog box) or by saving them in an MQSC command file and running that file.

Issuing commands interactively

To issue an MQSC command by typing its parameters, type it in the Command field of the MQSC page of the MQSeries Properties dialog box. The page is shown in Figure 17. You can split the command across multiple lines by ending each line except the last one with a plus (+) sign. When you have finished typing the command, press the Enter key to submit the command. A message indicating the success or failure of the command is displayed in the Results field.



```
define qlocal('Test_Q_2')
```

```
AMQ3500: Define local queue Test_Q_2 was successful.
```

Figure 17. The MQSC page

If the command is successful, the command is selected in the Command field. If you press any keys other than the direction keys, the Command field is cleared. The command is saved, and you can recall it using the scroll buttons.

If the command is unsuccessful, it is not selected. You can correct it and resubmit it without having to retype the whole command.

Running a command file

To run an MQSC command file, click the **Run MQSC File** push button on the MQSC page of the MQSeries Properties dialog box. This displays a window that allows you to select the file you want to run.

To view an MQSC file, select the file in the Run MQSC File dialog, click mouse button 2, then select **Open** from the menu. This loads the file into the Windows WordPad program. You can view and edit the file using WordPad.

When you have chosen the file you want to use, click the **Run** push button in the Run MQSC File dialog. This runs the file against the active queue manager. If any errors or warnings occur, MQSeries for Windows provides an option to view the log file. A message indicating the success or failure of the command is displayed in the Results field.

Writing MQSC command files

Write MQSC command files in plain (ASCII) text. By convention, command files have a file-name extension of TST. If you open a TST file using the Windows Explorer, Windows opens the file using WordPad and loads it as a text file.

Figure 18 is an extract from the MQSC command file VENUS.TST that is supplied in the \Program Files\MQSeries for Windows\Samples directory (if you installed MQSeries for Windows in the default directory). This extract shows the command DEFINE QLOCAL and its attributes.

```
 ⋮
    DEFINE QLOCAL('SAMPLE.VENUS.LOCAL') REPLACE  +
           DESCR('Local queue') +
           DEFPSIST(YES) +
           SHARE
```

Figure 18. An extract from VENUS.TST

MQSC command files

The format of MQSC files

If you are using MQSC files you have copied from other MQSeries platforms, be aware that they may contain unrecognizable control characters, which the MQSC interpreter will reject. You can prevent this happening by editing and resaving the .TST file before you use it.

Most command files from other systems will work with MQSeries for Windows, but Table 12 shows differences you should note.

For more information on the format of command files, see the online *MQSeries for Windows Command Reference*.

Table 12. The format of MQSC command files

On MQSeries for Windows	On other MQSeries platforms
You ***cannot*** use commas as separators.	You can use commas as separators.
You ***cannot*** use a minus sign (–) as a continuation character.	You can use a minus sign (–) as a continuation character.
The continuation character ***must*** be at the end of a keyword, data value, or quoted string.	The continuation character can appear anywhere within the command.
The description you specify using the DESCR keyword cannot contain characters from a double-byte character set (DBCS).	The DESCR keyword can use DBCS characters.

Understanding errors in the MQSC log files

When you run an MQSC command file, the queue manager creates a log file called MQSC.LOG in the directory for the active queue manager. For example, if your queue manager is called TEST, the log file is \Program Files\MQSeries for Windows\QMgrs\TEST\MQSC.LOG (if you installed MQSeries for Windows in the default directory).

To look at the contents of the log file, click the **View MQSC.LOG** push button on the MQSC page of the MQSeries Properties dialog box while the queue manager is running.

If an MQSC command that is run when you are creating a queue manager generates an error, you are prompted to view the MQSC log. The messages in the log file explain the error (for example, the syntax of the command is incorrect). If you need more information about these errors, see Appendix E, “Error messages” on page 213.

Note: MQSeries parses MQSC commands from left to right, and it stops parsing when it finds the first error. If your command still does not run after you have fixed the reported errors, the command contains more errors following those first reported.

If a command fails even when its syntax is correct, this means the queue manager was unable to run the command. In this case, the command returns a code that shows the reason for its failure; Appendix D, “Return codes” on page 207 lists these codes.

If you cannot view the log file, this may be because the log file is empty or incomplete. This can happen if there is not enough free disk space to write the file. You should move some data or applications to another disk to free some working space on the disk on which MQSeries for Windows is installed. You can use the Service page of the MQSeries Properties dialog box to see how much disk space you have available.

Examples of MQSC command files

The following figures show simple MQSC command files that are supplied with MQSeries for Windows:

- Figure 19 on page 114 shows the file MARS.TST
- Figure 20 on page 115 shows the file VENUS.TST

Both of these files are supplied in directory \Program Files\MQSeries for Windows\Samples (if you installed MQSeries for Windows in the default directory).

MQSC command files

```
* Define a local transmission queue - messages will be put here
* before being sent to the remote queue manager.
DEFINE QLOCAL('SAMPLE.MARS.XMIT') REPLACE +
       DESCR('Local transmission queue')  +
       USAGE(XMITQ)

* Define the remote queue.
* The sample application should put messages on this queue.
DEFINE QREMOTE('SAMPLE.MARS.REMOTE') REPLACE  +
       DESCR('Remote queue defined on MARS') +
       DEFPSIST(YES) +
* This is the name of the local queue on the remote machine
       RNAME('SAMPLE.VENUS.LOCAL') +
* This is the name of the queue manager on the remote machine
       RQMNAME('VENUS') +
* This is the name of the local transmission queue to be used
       XMITQ('SAMPLE.MARS.XMIT')

* Define the channel that will remove messages from the transmission
* queue SAMPLE.MARS.XMIT and send them to the machine specified
* by CONNAME.
* Change CONNAME to the TCP/IP name of the machine where
* the remote queue manager is running.
*
DEFINE CHANNEL ('MARS.TO.VENUS')  CHLTYPE(SDR) TRPTYPE(TCP) +
       XMITQ('SAMPLE.MARS.XMIT') +
       CONNAME('VENUS TCP/IP machine name') +
       DESCR('Sender channel for messages to queue manager VENUS') +
       REPLACE

* Define the channel that will accept messages from the remote
* queue manager on the machine specified by CONNAME.
* Change CONNAME to the TCP/IP name of the machine where
* the remote queue manager is running.
*
DEFINE CHANNEL ('VENUS.TO.MARS') CHLTYPE(RQSTR) TRPTYPE(TCP) +
       CONNAME('VENUS TCP/IP machine name') +
       DESCR('Requester channel for messages from queue manager VENUS') +
       REPLACE

* Define the local queue where the remote machine will put its
* messages.  The sample application should get messages from this queue.
DEFINE QLOCAL('SAMPLE.MARS.LOCAL') REPLACE +
       DESCR('Local queue') +
       DEFPSIST(YES)  +
       SHARE
```

Figure 19. The supplied file MARS.TST

```
* Define a local transmission queue - messages will be put here
* before being sent to the remote queue manager
DEFINE QLOCAL('SAMPLE.VENUS.XMIT') REPLACE +
       DESCR('Local transmission queue')  +
       USAGE(XMITQ)


* Define the remote queue.
* The sample application should put messages on this queue
DEFINE QREMOTE('SAMPLE.VENUS.REMOTE') REPLACE  +
       DESCR('Remote queue defined on VENUS') +
       DEFPSIST(YES) +
* This is the name of the local queue on the remote machine
       RNAME('SAMPLE.MARS.LOCAL') +
* This is the name of the queue manager on the remote machine
       RQMNAME('MARS') +
* This is the name local transmission queue to be used
       XMITQ('SAMPLE.VENUS.XMIT')


* Define the channel that will remove messages from the transmission
* queue SAMPLE.VENUS.XMIT and send them to the remote queue
* manager.
DEFINE CHANNEL ('VENUS.TO.MARS') CHLTYPE(SVR) TRPTYPE(TCP) +
       XMITQ('SAMPLE.VENUS.XMIT') +
       DESCR('Server channel for messages to queue manager MARS') +
       REPLACE


* Define the channel that will accept messages from the remote
* queue manager.
DEFINE CHANNEL ('MARS.TO.VENUS')  CHLTYPE(RCVR) TRPTYPE(TCP) +
       DESCR('Receiver channel for messages from queue manager MARS') +
       REPLACE


* Define the local queue where the remote machine will place
* its messages.
* The sample application should get messages from this queue
DEFINE QLOCAL('SAMPLE.VENUS.LOCAL') REPLACE +
       DESCR('Local queue') +
       DEFPSIST(YES)  +
       SHARE
```

Figure 20. The supplied file VENUS.TST

MQSC commands supported by MQSeries for Windows

MQSeries for Windows supports a subset of the MQSeries commands (MQSC). This subset is shown in Table 13. Version 2.1 of MQSeries for Windows adds support for some commands that Version 2.0 does not support.

For a description of the syntax of each command, see the online *MQSeries for Windows Command Reference*.

Table 13 (Page 1 of 3). MQSC commands and MQSeries for Windows

Command	**Available in V2.0**	**Available in V2.1**
ALTER CHANNEL	Yes	Yes
ALTER NAMELIST	No	No
ALTER PROCESS	No	No
ALTER QALIAS	Yes	Yes
ALTER QLOCAL	Yes	Yes
ALTER QMGR	Yes	Yes
ALTER QMODEL	Yes	Yes
ALTER QREMOTE	Yes	Yes
ALTER SECURITY	No	No
ALTER TRACE	No	No
ARCHIVE LOG	No	No
CLEAR QLOCAL	Yes	Yes
DEFINE BUFFPOOL	No	No
DEFINE CHANNEL	Yes	Yes
DEFINE MAXSMSGS	No	No
DEFINE NAMELIST	No	No
DEFINE PROCESS	No	No
DEFINE PSID	No	No
DEFINE QALIAS	Yes	Yes
DEFINE QLOCAL	Yes	Yes
DEFINE QMODEL	Yes	Yes
DEFINE QREMOTE	Yes	Yes
DEFINE STGCLASS	No	No
DELETE CHANNEL	Yes	Yes
DELETE NAMELIST	No	No
DELETE PROCESS	No	No
DELETE QALIAS	Yes	Yes
DELETE QLOCAL	Yes	Yes

Table 13 (Page 2 of 3). MQSC commands and MQSeries for Windows

Command	**Available in V2.0**	**Available in V2.1**
DELETE QMODEL	Yes	Yes
DELETE QREMOTE	Yes	Yes
DISPLAY CHANNEL	No	Yes
DISPLAY CHSTATUS	No	No
DISPLAY DQM	No	No
DISPLAY CMDSERV	No	No
DISPLAY MAXSMSGS	No	No
DISPLAY NAMELIST	No	No
DISPLAY PROCESS	No	No
DISPLAY QMGR	No	Yes
DISPLAY QUEUE	No	Yes
DISPLAY SECURITY	No	No
DISPLAY STGCLASS	No	No
DISPLAY THREAD	No	No
DISPLAY TRACE	No	No
DISPLAY USAGE	No	No
PING CHANNEL	No	No
PING QMGR	No	No
RECOVER BSDS	No	No
REFRESH SECURITY	No	No
RESET CHANNEL	Yes	Yes
RESOLVE CHANNEL	Yes	Yes
RESOLVE INDOUBT	No	No
RVERIFY SECURITY	No	No
START CHANNEL	Yes	Yes
START CHINIT	No	No
START CMDSERV	No	No
START LISTENER	No	No
START QMGR	No	No
START TRACE	No	No
STOP CHANNEL	Yes	Yes
STOP CHINIT	No	No
STOP CMDSERV	No	No
STOP LISTENER	No	No
STOP QMGR	No	No

MQSC commands

Table 13 (Page 3 of 3). MQSC commands and MQSeries for Windows		
Command	**Available in V2.0**	**Available in V2.1**
STOP TRACE	No	No

Chapter 11. Making changes for a user

If you do not want to use MQSC commands to change the MQ components on a user's workstation, you can make changes using an administration application running on a server to which the workstation can connect. Such an application sends MQSeries programmable command format (PCF) command messages to create, change, and delete MQ objects. However, you cannot use PCF commands to change MQ connections or channel groups.

This chapter uses the following sections to explain how to use PCF commands:

- "Controlling the command server"
- "Making changes using a PCF application" on page 120
- "PCF commands supported by MQSeries for Windows" on page 121
- "Restrictions on using PCF commands" on page 122
- "Controlling access to the administration features" on page 125

Controlling the command server

Before PCF commands can run on a workstation:

- There must be an active connection on the workstation.
- The command server must be running on that workstation.

The active connection can be any connection that uses the queue manager you want to change. You can use PCF commands to change any of the objects that this queue manager owns.

Users of the Compact version of MQSeries for Windows can start the command server from the Administration page of the MQSeries Properties dialog box. If you have defined a password in their MQD files, they must type this password before they can open the Administration page.

Alternatively, when you create an MQD file, you can define a connection in such a way that the command server always runs when that connection is active. This way, you do not have to give the user the password to the administration features. However, the command server is a heavy user of system resources, so you should allow it to run only with a connection used for administration purposes, not with one that is used frequently.

Users can stop the command server in two ways:

- If they have access to the Administration page of the MQSeries Properties dialog box, they can click the **Stop** push button on the Command Server tab.
- They can stop the active connection; this also stops the command server.

PCF commands

Making changes using a PCF application

To understand how to use a PCF application to make changes to a user's workstation, read the following example.

A travelling salesman, named John, uses MQSeries for Windows applications to send his sales orders to an office server. We saw how he might do this in the example described in "Choosing an MQ connection" on page 24. But he is finding that he is having to type more data than he used to for each order. He has told his MQ administrator, named Janet, that he will soon exceed the limits he was told for the application he uses. When Janet investigates this, she finds that, although the application will accept more data, the queue she defined for John to use on his laptop cannot hold messages greater than 500 KB in size. This means that some of the data that John types could be lost.

Janet realizes that she can easily increase the size of John's queue using the PCF application she uses to administer her other MQ users. This is how she does it:

1. Janet telephones John, and tells him to start the Administration connection on his laptop computer. This is a dial-up connection that she defined so he can dial in to her server. She also tells him to start the command server on his laptop; for this, she has to tell him the administration password she has set up on the Compact version of MQSeries for Windows he has on his laptop.
2. John powers on his laptop computer. This starts Windows 95 and MQSeries for Windows.
3. John connects his laptop computer to his modem so the Windows 95 dial-up networking software can connect to Janet's server.
4. On the Connections page of MQSeries for Windows, John selects the Administration connection
5. When the MQ icon on his Windows taskbar shows that the dial-up connection is ready, John opens the Administration page of the MQSeries Properties dialog box. He has to type the password Janet tells him.
6. On the Command Server tab of the Administration page, John clicks on the push button to start the command server.
7. Janet starts the MQ administration application she uses to send PCF command messages. She uses this application to change the queue attribute that specifies the maximum size of messages on John's queue.
8. When Janet stops her application, John stops the command server on his laptop, then powers off his laptop.

This simple example shows how you can make changes to a user's workstation without having to sit in front of it yourself. If Janet wanted to make the changes by sitting at the laptop herself, she could do this using MQSC commands. Alternatively, she could give John instructions over the telephone on how to type the MQSC commands.

Now that Janet has told John the password that opens the Administration page on his MQSeries Properties dialog box, she may want to change the password to hide that page again. She must do this in the MQD file that John uses. John is a mobile worker, so he uses an MQD file that is physically located on his laptop computer, rather than one located on a server. This means Janet must send him a new MQD file (containing keywords to change the password) for him to copy on to his laptop. MQ runs this file automatically next time John powers on his laptop.

PCF commands supported by MQSeries for Windows

MQSeries for Windows Version 2.1 supports a subset of the MQSeries Programmable Command Format (PCF) commands. This subset is shown in Table 14.

For a description of each command, see the *MQSeries Programmable System Management* manual. However, MQSeries for Windows does not support all the parameters and values of the commands; for more information, see “Restrictions on using PCF commands” on page 122.

Table 14 (Page 1 of 2). PCF commands and MQSeries for Windows

Command	**Available in V2.1**
Change Channel	Yes
Change Process	No
Change Queue	Yes
Change Queue Manager	Yes
Clear Queue	Yes
Copy Channel	Yes
Copy Process	No
Copy Queue	Yes
Create Channel	Yes
Create Process	No
Create Queue	Yes
Delete Channel	Yes
Delete Process	No
Delete Queue	Yes
Escape	No
Inquire Channel	Yes
Inquire Channel Names	Yes
Inquire Channel Status	Yes
Inquire Process	No
Inquire Process Names	No
Inquire Queue	Yes

PCF commands

Table 14 (Page 2 of 2). PCF commands and MQSeries for Windows

Command	**Available in V2.1**
Inquire Queue Manager	Yes
Inquire Queue Names	Yes
Ping Channel	No
Ping Queue Manager	Yes
Reset Channel	Yes
Reset Queue Statistics	Yes
Resolve Channel	Yes
Start Channel	Yes
Start Channel Initiator	No
Start Channel Listener	No
Stop Channel	Yes

Restrictions on using PCF commands

MQSeries for Windows does not support the following MQ features so you cannot specify values related to them in PCF commands:

- Client connection or server connection channel types
- Application triggering
- Data conversion
- Dead-letter queues
- Authority events

Also, you cannot use PCF commands to work with MQ connections and channel groups.

For more information, see:

- "Restrictions on the Change, Copy, and Create Queue commands" on page 123
- "Restrictions on the Change Queue Manager command" on page 124
- "Restrictions on the Change, Copy, and Create Channel commands" on page 124
- "Restrictions on the Delete Channel command" on page 124
- "Restrictions on the Inquire Channel and Inquire Channel Names commands" on page 124

In addition, MQSeries for Windows does not support the new command parameters marked with vertical bars in the left margin of the -05 edition of the *MQSeries Programmable System Management* book (SC33-1482). However, MQSeries for Windows does support fast nonpersistent messages, so on the following commands you can use any of the values of the NonPersistentMsgSpeed parameter:

- Change Channel
- Copy Channel
- Create Channel
- Inquire Channel
- Inquire Channel Status

When you write a PCF application that you will run on an MQSeries for Windows queue manager, you must use the C programming language. MQSeries for Windows supplies definitions of the data types and constants you need in the include file CMQCFC.H. If you installed MQSeries for Windows in the default directory, this include file is supplied in the \Program Files\MQSeries for Windows\Include directory.

MQSeries for Windows does not do any data conversion. If the data you are sending in a PCF command to an MQSeries for Windows queue manager requires conversion to a different machine encoding, integer representation, or coded character set, the channel that sends the data must do it. To make the channel perform the data conversion, use the keyword CONVERT=YES in its definition.

Restrictions on the Change, Copy, and Create Queue commands

MQSeries for Windows does not support triggering, so although you can set the following parameters of the queue commands to any value, MQSeries for Windows does not use that value:

- InitiationQName
- ProcessName
- TriggerData
- TriggerDepth
- TriggerMsgPriority
- TriggerType

Also, you can set the following parameters only to the value shown:

- Scope must be set to MQSCO_Q_MGR
- TriggerControl must be set to MQTC_OFF

PCF commands

Restrictions on the Change Queue Manager command

MQSeries for Windows does not support triggering, so although you can set the TriggerInterval parameter of the Change Queue Manager command to any value, MQSeries for Windows does not use that value.

Also, MQSeries for Windows does not support a dead-letter queue or authority events, so you can set the following parameters only to the value shown:

- DeadLetterQName must be set to all blanks
- AuthorityEvent must be set to MQEVR_DISABLED

Restrictions on the Change, Copy, and Create Channel commands

You can set the following parameters of the Change, Copy, and Create Channel commands to any value, but MQSeries for Windows does not use that value:

- MCAName
- ModeName
- MsgRetryUserData
- Password
- QMgrName
- TpName
- UserIdentifier

Also note the restrictions on the values you can specify in the following parameters:

- ChannelType cannot be set to MQCHT_CLNTCONN or MQCHT_SVRCONN
- DataConversion must be set to MQCDC_NO_SENDER_CONVERSION
- MCAType must be set to MQMCAT_PROCESS
- MsgRetryCount must be set to 0
- MsgRetryExit must be set to all blanks
- MsgRetryInterval must be set to 0
- TransportType must be set to MQXPT_TCP

Restrictions on the Delete Channel command

MQSeries for Windows does not support client connection or server connection channels, so in the ChannelTable parameter of the DeleteChannel command, you cannot use the value MQCHTAB_CLNTCONN.

Restrictions on the Inquire Channel and Inquire Channel Names commands

MQSeries for Windows does not support client connection or server connection channels, so in the ChannelType parameter of the Inquire Channel and Inquire Channel Names commands, you cannot use the values MQCHT_CLNTCONN or MQCHT_SVRCONN.

Controlling access to the administration features

If you decide to make changes by sitting in front of the workstation yourself, you may have to type the password to open the Administration page of the MQSeries Properties dialog box. To hide this page again, simply close the MQSeries Properties dialog box.

If you want to change the password (maybe after telling it to the user so they can start the command server), either change it in their MQD file, or give them a new MQD file that changes the password. If you give them a new MQD file, you can tell them to run it immediately, but MQ runs the file next time it starts anyway.

Chapter 12. Running the sample programs on one workstation

Use the procedures described in this chapter if you want to run the supplied sample programs on a single workstation. The samples are:

The Putting Messages sample
This sample puts a message on a specified queue.

The Browsing Messages sample
This sample browses (that is, copies without removing) a message on a specified queue. By specifying the same queue that you used with the Putting Messages sample, you can browse the messages you put on the queue.

The Getting Messages sample
This sample gets a message from a specified queue. By specifying the same queue that you used with the Putting Messages sample, you can retrieve the messages you put on the queue.

Note that to run the sample programs, you must have installed the Complete version of MQSeries for Windows.

When you know how to run the samples, you may want to move on to the following:

- Chapter 13, “Running the sample programs on two workstations” on page 133 explains how to use the sample programs to exchange messages between two workstations.
- Chapter 17, “Understanding the sample programs” on page 185 explains the design of the sample programs.

Creating your test connection

The definition file CREATEMQ.MQD that MQSeries for Windows supplies creates a connection you can use to run the sample programs. Unless you have replaced this file with one of your own, MQ creates the test connection when you install the product.

The supplied MQD file:

- Creates a connection named Sample_Connection
- Creates a queue manager named Sample_QM
- Runs the MQSC command file named AMQSCOSW.TST, which creates a sample local queue
- Starts the sample connection

Running the samples

To run the sample programs, the sample connection must be active. If it is not, start it as follows:

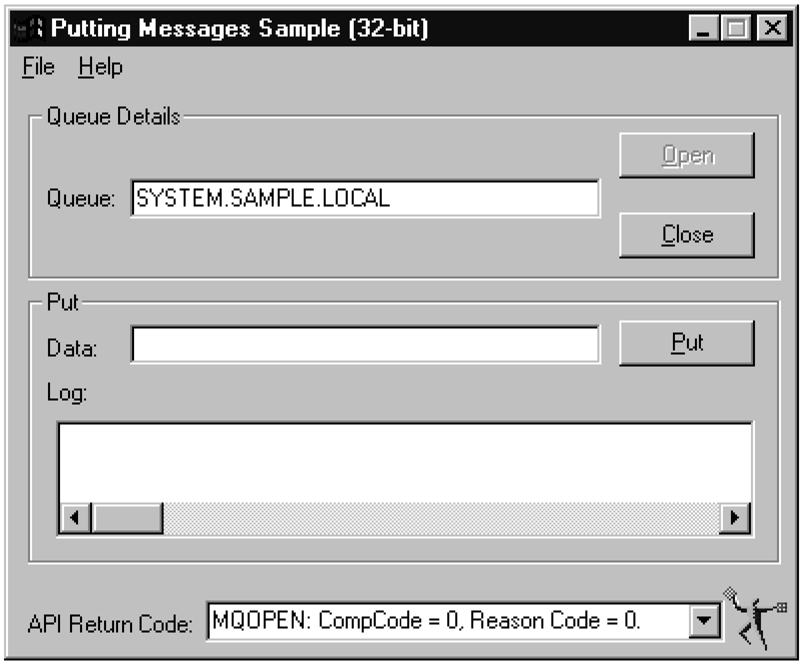
1. Double-click on the MQ icon on the Windows taskbar or in the Control Panel. This displays the MQSeries Properties dialog box.
2. On the Connections page of the MQSeries Properties dialog box, double-click on the Sample connection, or select it and click on the **Start** push button.

You can see that the connection has started because a green check mark appears beside its name on the Connections page and (if you have set the option to display the MQ icon) on the MQ icon on the Windows taskbar.

You can then use the procedures described in the remainder of this chapter to put some messages on the queue, and, with the connection still running, browse or get the messages from the queue.

Putting messages on a queue

To start the Putting Messages sample, double-click on the name of its executable file. The file is named AMQSPUTW.EXE; use the Windows Explorer to find this file in the \MQSeries for Windows\Samples\C directory. This starts the program and connects it to the active queue manager. Figure 21 shows the window that appears.



Queue: SYSTEM.SAMPLE.LOCAL

API Return Code: MQOPEN: CompCode = 0, Reason Code = 0.

Figure 21. The Putting Messages Sample window

If you want to run the Visual Basic version of the file, you must build an executable file first; see “Building the Visual Basic samples” on page 191.

To put messages on a queue:

1. In the **Queue** field of the Putting Messages Sample window, type the following name in uppercase letters:

   ```
   SYSTEM.SAMPLE.LOCAL
   ```

2. Click the **Open** push button to open the queue.
3. Check that the reason code for the MQOPEN call displayed in the **API Return Code** field is zero; if it is not, the workstation beeps.

   If the reason code is not zero, the program has not opened the queue. The reason codes are listed in Appendix D, “Return codes” on page 207. You are most likely to see the following reason codes:

   2018 or 2059 This means the queue manager is not running. If this happens:

   a. Close the sample by selecting **Exit** from the File menu.
   b. Start the Sample connection by double-clicking on its entry in the Connections page of the MQSeries Properties dialog box.
   c. Restart the Putting Messages sample.

   2085 This means the queue does not exist. If this happens, check that you have typed the name correctly, using uppercase letters.

4. Type some message text in the **Data** field.
5. Click the **Put** push button.

   MQ puts the message on the queue. The message text also appears in the **Log** list box.

6. Check that the reason code for the MQPUT call displayed in the **API Return Code** field is zero.
7. Repeat steps 4 and 5 as many times as you want to put more messages on the queue.

You can leave the Putting Messages sample running so you can continue putting messages after you have used the other samples to browse or get the messages.

When you have finished using the Putting Messages sample program:

1. Click the **Close** push button.

   This closes the queue and disconnects the sample from the queue manager.

2. Close the sample by selecting **Exit** from the File menu.

Browsing messages on a queue

To start the Browsing Messages sample, double-click on the name of its executable file. The file is named AMQSBCGW.EXE; use the Windows Explorer to find this file in the \MQSeries for Windows\Samples\C directory. This starts the program and connects it to the active queue manager.

If you want to run the Visual Basic version of the file, you must build an executable file first; see "Building the Visual Basic samples" on page 191.

To browse (that is, view) the messages on a queue:

1. In the **Queue** field of the Browsing Messages Sample window, type the following name in uppercase letters:

   ```
   SYSTEM.SAMPLE.LOCAL
   ```

2. Click the **Open** push button to open the queue.
3. Check that the reason code for the MQOPEN call displayed in the **API Return Code** field is zero; if it is not, the workstation beeps.

   If the reason code is not zero, the program has not opened the queue. This is likely to be for one of the reasons explained in "Putting messages on a queue" on page 128.

4. Click the **Browse** push button.

   MQ finds the oldest message on the queue and displays it:

   - The **Data** field shows the message data you typed when you put the message on the queue.
   - The **Length** field shows the number of characters in the message data.
   - The **Header** list box shows the fields of the message descriptor structure (MQMD) that MQ adds when it puts the message on the queue.

5. Check that the reason code for the MQGET call displayed in the **API Return Code** field is zero.

   You are most likely to see the following reason code:

   2033 This means there are no messages on the queue, or you have browsed all the messages and you are at the end of the queue.

   The reason codes are listed in Appendix D, "Return codes" on page 207.

6. You can keep repeating step 4 to browse any other messages on the queue.

   The sample steps through the messages until it reaches the end of the queue. After it has displayed the last message, if you click on **Browse** again, the workstation beeps and reason code 2033 is displayed, showing you there are no more messages on the queue. If you want to start browsing at the start of the queue again, click the **Close** push button, then open the queue again and browse as before.

You can leave the program running so you can continue browsing messages after you have used the Putting Messages sample program to put some more on the queue.

When you have finished using the Browsing Messages sample program:

1. Click the **Close** push button.

   This closes the queue and disconnects the sample from the queue manager.

2. Close the sample by selecting **Exit** from the File menu.

Getting messages from a queue

To start the Getting Messages sample, double-click on the name of its executable file. The file is named AMQSGETW.EXE; use the Windows Explorer to find this file in the \MQSeries for Windows\Samples\C directory. This starts the program and connects it to the active queue manager.

If you want to run the Visual Basic version of the file, you must build an executable file first; see "Building the Visual Basic samples" on page 191.

To get messages from a queue:

1. In the **Queue** field of the Getting Messages Sample window, type the following name in uppercase letters:

   ```
   SYSTEM.SAMPLE.LOCAL
   ```

2. Click the **Open** push button to open the queue.

3. Check that the reason code for the MQOPEN call displayed in the **API Return Code** field is zero; if it is not, the workstation beeps.

   If the reason code is not zero, the program has not opened the queue. This is likely to be for one of the reasons explained in "Putting messages on a queue" on page 128.

4. Click the **Get** push button.

   MQ removes the oldest message from the queue and displays it:

   - The **Data** field shows the message data you typed when you put the message on the queue.
   - The **Length** field shows the number of characters in the message data.

   If there are no messages on the queue, the sample waits for 15 seconds, then it returns reason code 2033. Use the Putting Messages sample to put some more messages on the queue.

5. Check that the reason code for the MQGET call displayed in the **API Return Code** field is zero.

   You are most likely to see the following reason code:

2033 This means there are no messages on the queue. If this happens, you can use the Putting Messages sample to put more messages on the queue.

The reason codes are listed in Appendix D, “Return codes” on page 207.

6. You can keep repeating step 4 on page 131 to get any other messages from the queue.

   The sample removes each message until it reaches the end of the queue. After you have removed the last message, if you click on **Get** again, the sample waits 15 seconds for a new message to arrive. If a message does not arrive in this time, the workstation beeps and the sample displays reason code 2033 to show you there are no more messages on the queue. You must put another message on the queue before you can use the Getting Messages sample again.

You can leave the program running so you can continue getting messages after you have used the Putting Messages sample program to put some more on the queue.

When you have finished using the Getting Messages sample program:

1. Click the **Close** push button.

   This closes the queue and disconnects the sample from the queue manager.

2. Close the sample by selecting **Exit** from the File menu.

Notes about the sample programs

When you are using the sample programs, remember:

- The Browsing Messages sample does not remove messages from the queue; it just displays them.
- The Getting Messages sample ***removes*** messages from the queue.
- The Browsing Messages and Getting Messages samples retrieve messages in the same order in which the messages were put on the queue.
- The put, browse, and get operations are independent, so the sample programs can operate at the same time, or one at a time.

Chapter 13. Running the sample programs on two workstations

This chapter describes how to set up connections on two workstations so they can send messages to each other. It then describes how to use the sample programs supplied with MQSeries for Windows to exchange messages between the two workstations.

In the examples, the two connections are named Mars_Connection and Venus_Connection. MQSeries for Windows supplies files that contain definitions for these two connections, and for the components they use. They use queue managers named MARS and VENUS. The channels are initiated from MARS, but you can send messages from either of the queue managers.

The procedure involves:

1. Setting up the two connections.

   If both of your workstations are to run MQSeries for Windows, see "Setting up the two connections."

   If one of your workstations is to run another MQSeries product, see "When one of your workstations is on a different platform" on page 141.

2. Running the sample programs.

   This is described in "Running the sample programs on VENUS and MARS" on page 140.

Setting up the two connections

Before you can set up your two connections, install MQSeries for Windows on each of the two workstations you are going to use. Make sure you install the Complete version of MQSeries for Windows on each one.

It is then advisable to use the TCP/IP **ping** command to test the TCP/IP connection between them.

Running the samples

Using the ping command

Before you can use the **ping** command, you need to find the internet protocol (IP) address of each workstation. This is also known as the network address, computer name, host name, or machine name. The IP address has a dotted-decimal format (for example, 152.78.108.4). But if your TCP/IP uses a domain name service, you can create a text alias for your IP address (note that this alias is case sensitive). To find the IP address of your own workstation, look in the Windows Control Panel.

When you know the addresses, try pinging one workstation by typing the **ping** command at a DOS command prompt on the other workstation; for example:

```
ping 152.78.108.4
```

If the ping is successful, the command displays messages that show the time taken for the test message to be delivered. If the ping is not successful, you need to establish a TCP/IP connection between the two workstations before you proceed.

You are now ready to use the supplied definition (MQD) files to create a connection, a queue manager, its queues, and its channels on each workstation.

Note: If you do not want to use the MQD files to set up your queue managers, see "Setting up the two connections without using MQD files" on page 138.

Setting up VENUS

Set up VENUS as follows:

1. By default, MQSeries for Windows is installed in the directory \Program Files\MQSeries for Windows. If you have installed it elsewhere, edit the file VENUS.MQD to change the line (highlighted below) that refers to this directory. (You will find VENUS.MQD in the Samples subdirectory.)

```
[Component_1]
ComponentType=QueueManager
Name=VENUS
Description=Queue manager to communicate with MARS
LoadUserMQSC_1=\Program Files\MQSeries for Windows\Samples\venus.tst
Replace=yes
```

2. Make a backup copy of the supplied file named CREATEMQ.MQD.

   For example, from the directory \Program Files\MQSeries for Windows, type at a DOS command prompt:

```
copy CREATEMQ.MQD CREATEMQ.OLD
```

3. From the directory \Program Files\MQSeries for Windows\Samples (or the directory in which the samples are installed), copy the file VENUS.MQD to replace the supplied file CREATEMQ.MQD:

   ```
   copy VENUS.MQD CREATEMQ.MQD
   ```

   This is because by default, MQ uses the file named CREATEMQ.MQD.

4. Run the MQD file.

   To do this:

   a. Stop the active connection.
   b. Close the MQSeries Properties dialog box.
   c. Click with mouse button 2 on the MQ icon on the Windows taskbar.
   d. Click on the **Run MQD file now** item on the menu that appears.

   When MQ runs the MQD file (supplied as VENUS.MQD), it uses the definitions in the file to:

   - Create a connection named Venus_Connection.
   - Create a queue manager named VENUS.
   - Run the MQSC command file named VENUS.TST. This creates the three queues and two channel definitions on VENUS that are shown in Figure 22 on page 136.
   - Create a channel group named VENUSGroup.

5. Start Venus_Connection from the Connections page of the MQSeries Properties dialog box. When the connection starts, the listener starts automatically.

If the connection starts successfully, a check mark appears beside its icon on the Connections page of the MQSeries Properties dialog box. If you have the taskbar icon visible, that also has a check mark. If the listener starts successfully, a check mark appears beside the channel group icon on the Connections page.

VENUS is now ready to respond to incoming channel-connect requests.

Setting up MARS

On your second workstation, set up MARS as follows:

1. By default, MQSeries for Windows is installed in the directory \Program Files\MQSeries for Windows. If you have installed it elsewhere, edit the file MARS.MQD to change the line (highlighted below) that refers to this directory. (You will find MARS.MQD in the Samples subdirectory.)

```
[Component_1]
ComponentType=QueueManager
Name=MARS
Description=Queue manager to communicate with VENUS
LoadUserMQSC_1=\Program Files\MQSeries for Windows\Samples\mars.tst
Replace=yes
```

Running the samples

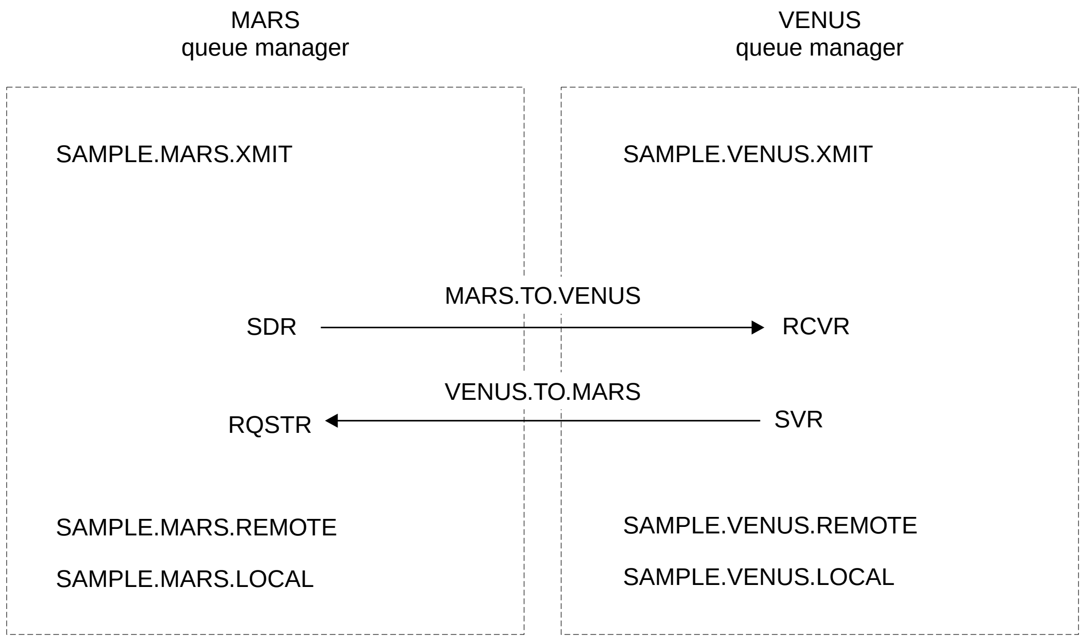


Figure 22. The objects that VENUS.TST and MARS.TST create

2. Make a backup copy of the supplied file named CREATEMQ.MQD.

   For example, from the directory \Program Files\MQSeries for Windows, type at a DOS command prompt:

   ```
   copy CREATEMQ.MQD CREATEMQ.OLD
   ```

3. From the directory \Program Files\MQSeries for Windows\Samples (or the directory in which the samples are installed), copy the file MARS.MQD to replace the supplied file CREATEMQ.MQD:

   ```
   copy MARS.MQD CREATEMQ.MQD
   ```

   This is because by default, MQ runs the file named CREATEMQ.MQD.

4. Edit the MQSC command file named MARS.TST.

   This file is in the \Program Files\MQSeries for Windows\Samples subdirectory in a default installation. Change the CONNAME attribute on both the Sender and Requester channel definitions to the IP address of the VENUS workstation:

```
DEFINE CHANNEL ('MARS.TO.VENUS')  CHLTYPE(SDR) TRPTYPE(TCP) +
       XMITQ('SAMPLE.MARS.XMIT') +
       CONNAME('VENUS TCP/IP machine name') +
       DESCR('Sender channel for messages to queue manager VENUS') +
       REPLACE

DEFINE CHANNEL ('VENUS.TO.MARS') CHLTYPE(RQSTR) TRPTYPE(TCP) +
       CONNAME('VENUS TCP/IP machine name') +
       DESCR('Requester channel for messages from queue manager VENUS') +
       REPLACE
```

Note: The IP names you use for the CONNAME attribute are case sensitive.

5. Run the MQD file.

   To do this:

   a. Stop the active connection.
   b. Close the MQSeries Properties dialog box.
   c. Click with mouse button 2 on the MQ icon on the Windows taskbar.
   d. Click on the **Run MQD file now** item on the menu that appears.

   When MQ runs the MQD file (supplied as MARS.MQD), it uses the definitions in the file to:

   - Create a connection named Mars_Connection.
   - Create a queue manager named MARS.
   - Run the MQSC command file named MARS.TST. This creates the three queues and two channel definitions that are shown in Figure 22 on page 136.
   - Create a channel group named MARSGroup.

6. Start Mars_Connection from the Connections page of the MQSeries Properties dialog box.

When the MARSGroup channel group starts, it sends channel-connect requests to VENUS. If this is successful, a check mark appears beside the channel group icon on the Components page of the MQSeries Properties dialog box. If the check mark does not appear, or if an exclamation mark appears instead, the channels have not started. The two most likely reasons for this are:

- In the MARS.TST file, the CONNAME attribute does not contain the correct IP name for the VENUS workstation. For example, you may have typed the name using lowercase letters, but the name should be in uppercase letters.

  You can correct this error by changing the value of the ConnectionName attribute for the appropriate channel. Do this from the Components page of the MQSeries Properties dialog box.

- There is a TCP/IP problem. Restart the workstation and try again. You do not need to re-create the queue manager.

Running the samples

When the VENUS and MARS queue managers are running, you are ready to run the sample programs. Proceed to “Running the sample programs on VENUS and MARS” on page 140.

Setting up the two connections without using MQD files

Note

Read this section only if you do not want to use the supplied MQD files to set up your workstations. You may want to use the information supplied here to help you understand how to create your own MQ components.

Before you can set up your two workstations, install MQSeries for Windows on each of the two workstations you are going to use. Make sure you install the Complete version of MQSeries for Windows on each one.

It is then advisable to use the TCP/IP **ping** command to send a test message from one workstation to the other to ensure that there is a TCP/IP connection between them. If you need more information on how to do this, see “Setting up the two connections” on page 133.

Setting up VENUS yourself

Set up VENUS as follows, using the Components page of the MQSeries Properties dialog box:

1. Create a queue manager named VENUS.

   Specify the MQSC command file VENUS.TST. In a default installation, this is in the \Program Files\MQSeries for Windows\Samples subdirectory.

2. Create a channel group named VENUSGroup on the VENUS queue manager.

   Ensure that you mark the **Start Listener** box.

3. Create a connection named Venus_Connection.

   The connection must comprise:

   - The VENUS queue manager
   - The VENUSGroup channel group

Setting up MARS yourself

On your second workstation, set up MARS as follows:

1. Edit the MQSC command file named MARS.TST to add the IP address of the VENUS workstation. For information on how to do this, see “Setting up MARS” on page 135.

2. Create a queue manager named MARS, using the Components page of the MQSeries Properties dialog box.

   Specify the MQSC command file MARS.TST.

3. Create a channel group named MARSGroup on the MARS queue manager, using the Components page of the MQSeries Properties dialog box.

   Check the **Include all user channels** check box. You do not need the listener.

4. Create a connection named Mars_Connection, using the Components page of the MQSeries Properties dialog box.

   The connection must comprise:

   - The MARS queue manager
   - The MARSGroup channel group

Starting the two connections

When you have created Venus_Connection and Mars_Connection, and their queue managers and channel groups, you are ready to start them. Perform the following steps:

1. On your first workstation, use the Connections page of the MQSeries Properties dialog box to start Venus_Connection.

   If this is successful, a check mark appears beside the connection icon. VENUS is now ready to respond to incoming channel-connect requests.

2. On your second workstation, use the Connections page of the MQSeries Properties dialog box to start Mars_Connection.

   If this is successful, a check mark appears beside the connection icon.

   This sends channel-connect requests to the VENUS queue manager. If this is successful, a check mark appears beside the channel group icon on the Components page of the MQSeries Properties dialog box.

If the check marks do not appear, or an exclamation mark appears instead, there has been an error in starting the channels. The two main reasons for failure are:

- In the MARS.TST file, the CONNAME attribute does not contain the correct IP name for the VENUS workstation. For example, you may have typed the name using lowercase letters, but the name uses uppercase letters.

  Correct the error by changing the value of the ConnectionName attribute for the appropriate channel using the Components page of the MQSeries Properties dialog box.

- There is a TCP/IP problem. Restart the workstation and try again. It is not necessary to re-create the queue manager.

When the VENUS and MARS queue managers are running, you are ready to run the sample programs. Proceed to “Running the sample programs on VENUS and MARS” on page 140.

Running the sample programs on VENUS and MARS

When you have set up VENUS and MARS, you can use the sample programs supplied with MQSeries for Windows to send messages between them:

The Putting Messages sample
Run the Putting Messages sample on MARS to put messages on the local queue owned by VENUS.

The Browsing Messages sample
Run the Browsing Messages sample on VENUS to browse messages on the same queue.

The Getting Messages sample
Run the Getting Messages sample on VENUS to get messages from the same queue.

Sending messages to VENUS

On MARS, use the Putting Messages sample program to send messages to the queue named SAMPLE.VENUS.LOCAL, which is owned by the VENUS queue manager. Follow the same steps you used when you ran the Putting Messages sample on a single workstation (see “Putting messages on a queue” on page 128), but in the **Queue** field of the Putting Messages Sample window, type the following name in uppercase letters:

```
SAMPLE.MARS.REMOTE
```

Notes:

1. The queue SAMPLE.MARS.REMOTE is defined in the file MARS.TST. It is a remote queue.
2. This is not the true name of the destination queue—it is the name of the local definition (on MARS) of the remote queue. MARS.TST resolves this name to SAMPLE.VENUS.LOCAL. Figure 22 on page 136 shows these queues.

Browsing messages on VENUS

On VENUS, use the Browsing Messages sample program to browse messages on the queue named SAMPLE.VENUS.LOCAL. Follow the same steps you used when you ran the Browsing Messages sample on a single workstation (see “Browsing messages on a queue” on page 130), but in the **Queue** field of the Browsing Messages Sample window, type the following name in uppercase letters:

```
SAMPLE.VENUS.LOCAL
```

The queue SAMPLE.VENUS.LOCAL is defined in the file VENUS.TST. It is a local queue.

Getting messages on VENUS

On VENUS, use the Getting Messages sample program to get messages from the queue named SAMPLE.VENUS.LOCAL. Follow the same steps you used when you ran the Getting Messages sample on a single workstation (see "Getting messages from a queue" on page 131), but in the **Queue** field of the Getting Messages Sample window, type the following name in uppercase letters:

```
SAMPLE.VENUS.LOCAL
```

The queue SAMPLE.VENUS.LOCAL is defined in the file VENUS.TST. It is a local queue.

Sending messages to MARS

You should also test that you can send messages in the opposite direction, that is, from VENUS to MARS. The configuration tasks you performed in "Setting up the two connections" on page 133 allows you to do this without making any further changes. All you need to do is:

1. Start the Putting Messages sample on VENUS.
2. Put some messages on the queue named SAMPLE.VENUS.REMOTE (which is a remote queue whose name resolves to SAMPLE.MARS.LOCAL).
3. Start the Browsing Messages or Getting Messages sample on MARS.
4. Browse or get the messages from the queue named SAMPLE.MARS.LOCAL.

When one of your workstations is on a different platform

If the workstation you are using for the VENUS queue manager is running MQSeries on a platform (or operating system) other than Windows, use the following steps to verify that they are configured correctly.

1. Create and start the queue manager VENUS, following the instructions given in the documentation for the queue manager on the platform you are using.
2. Create all the queues and channels defined in the MQSC command file named VENUS.TST, following the instructions for the platform you are using.
3. Start the listener on VENUS, following the instructions for the platform you are using.
4. Set up the MARS queue manager on Windows by following the procedure described in "Setting up MARS" on page 135.
5. Run the sample programs as described in "Running the sample programs on VENUS and MARS" on page 140.

If the workstation you are using for the MARS queue manager is running MQSeries on a platform other than Windows, use the following steps to verify that they are configured correctly.

1. Create and start the queue manager MARS, following the instructions given in the documentation for the queue manager on the platform you are using.

2. Create all the queues and channels defined in the MQSC command file named MARS.TST, following the instructions for the platform you are using.

   You must edit MARS.TST to add the IP address of VENUS in the CONNAME attributes of the two channel definitions. For information on how to do this, see "Setting up MARS" on page 135.

3. Set up the VENUS queue manager on Windows by following the procedure described in "Setting up VENUS" on page 134.
4. Start the two channels VENUS.TO.MARS and MARS.TO.VENUS, following the instructions for the platform you are using.
5. Run the sample programs as described in "Running the sample programs on VENUS and MARS" on page 140.

Note: MQSeries for Windows does not perform any data conversion so, if the operating system on which your other queue manager is running uses a code page or integer representation that is different from that of Windows, you must make sure the other queue manager does the required conversion. To make it do this before it sends a message to your Windows queue manager, specify `CONVERT(YES)` in the definition of the channel at the sending end.

Other tests you might want to try

When you are satisfied that your queue managers are operating as you would expect, you might want to try some of these other tests:

Can I put messages when the destination queue manager, or the channel to it, is not running?
Yes. When the queue manager, and the channel to it, start running, MQSeries for Windows delivers the messages you have put.

Can I put messages at the same time as getting them?
Yes, you can run the Putting Messages and Getting Messages sample programs at the same time. They can both use the same queue, but note that the Getting Messages sample always gets the oldest message from the queue.

Also, you can put messages from both queue mangers at the same time. This is because there is a separate channel for messages flowing in each direction.

How can I put messages on the local queue?
You might want to put messages on a queue that is local to the queue manager that is running the Putting Messages sample, instead of sending them to the other queue manager. To do this on MARS, in the **Queue** field of the Putting Messages Sample window, type the name SAMPLE.MARS.LOCAL.

Chapter 14. Diagnosing problems

This chapter suggests reasons for problems you may have with MQSeries for Windows. You usually start with a symptom, or set of symptoms, and trace them back to their cause.

Problem diagnosis is not problem solving. However, the process of problem diagnosis often enables you to solve a problem. For example, if you find that the cause of the problem is an error in an application program, you can solve the problem by correcting that error.

You may not always be able to solve a problem after determining its cause. For example, a performance problem may be caused by a limitation of your hardware, or you may find that the cause of your problem is in MQSeries for Windows, in which case you need to contact your IBM Support Center for a solution.

This chapter first describes:

- “Preliminary checks”
- “Problems with queues and queue managers” on page 144
- “Problems with channels and channel groups” on page 145
- “Problems with messages” on page 147

It then goes on to describe the service tools that MQSeries for Windows provides:

- “Service information” on page 149
- “Service Trace” on page 150
- “Monitoring MQSeries events” on page 151
- “MQSeries events generated by MQSeries for Windows” on page 152

Preliminary checks

Before you start problem determination in detail, it is worth looking for an obvious cause of the problem, or a likely area in which to start your investigation.

Has MQSeries for Windows run successfully before?

Even if MQSeries for Windows has run successfully before, you should check that the installation is correct. You can verify the installation ***at any time*** by running the Verify program. This tests the basic functions of MQSeries for Windows, so it can be a useful first step for problem diagnosis. For information on how to do this, see “Verifying your installation” on page 20.

Is there enough disk space?

You may not have enough free disk space available to run MQSeries for Windows. Use the Service page of the MQSeries Properties dialog box to display the amount of free disk space on the installed drive. Each new queue manager you create uses approximately 10 KB of disk space to hold its configuration information (such as queue definitions), so consider deleting any queue managers you no longer use.

Is there enough memory?

You may not have enough physical memory available to run MQSeries for Windows. Use the Service page of the MQSeries Properties dialog box to display the amount of physical memory available.

Is the service information correct?

Check the tabs on the Service page of the MQSeries Properties dialog box. If any of them contain blank or corrupted fields, this indicates a problem with your installation. To correct this, uninstall then reinstall MQSeries for Windows.

Problems with queues and queue managers

This section outlines some possible problems you may have with your queue managers and queues.

Is the receiving queue full?

If the queue to which you are sending messages is full, you could increase the value of its MaxDepth attribute. To do this, use the MQSeries Properties dialog box, the MQSC command ALTER QUEUE, or the PCF command CHANGE QUEUE.

Are some of your queues failing?

If you suspect the problem occurs with only some of your queues, examine the local queues that you think are showing problems:

1. Display the attributes of each queue using the MQSeries Properties dialog box.
2. Make the following tests:
   - If the CurDepth attribute has the same value as the MaxDepth attribute, the queue is full and is not being processed. Check that the applications that process the queue are running normally.
   - If the value of the CurDepth attribute is less than that of the MaxDepth attribute, check the following queue attributes to ensure they are correct:
     - The Shareability attribute defines whether the queue can be shared by more than one program. If this attribute is set so that the queue cannot be shared, another application could already have opened the queue for input; this prevents other programs opening the queue.
     - The InhibitGet and InhibitPut attributes define whether programs can get and put messages on the queue. If these attributes are set to the value `INHIBITED`, programs cannot use the queue.
   - If there are no programs getting messages from the queue, determine why this is so. It may be that the programs need to be started, a communications link has been broken, or a program cannot open the queue.

     Check the OpenInputCount and OpenOutputCount queue attributes. These attributes indicate whether the queue has been opened for input or output. If a value is zero, it indicates that no operations of that type can occur. Note that the values may have changed—the queue may have been open, but it is now closed.

You need to check the status at the time you expect to put or get a message.

Have data files been lost?

MQSeries for Windows does not support media recovery. If a disk error or disk failure causes loss or corruption of the queue manager data files, you cannot recover this data. You must re-create the queue manager and all the other objects you need.

Problems with channels and channel groups

This section outlines some possible problems with channels and channel groups. For all problems with channels or channel groups, you should use the MQSeries Properties dialog box to check the status of the channels; for information on how to do this, see "Viewing the status of an MQ object" on page 101.

Are the channels working?

You can see if a channel is currently sending messages by looking at the status markers in the MQSeries Properties dialog box. You can get more information by checking the status of the channel. Among the status attributes you can check are:

- The number of messages processed by the channel
- The number of bytes sent
- The number of bytes received

Why does a channel stop request not work?

A problem can sometimes arise when the receiving end of a channel stops, but the sending end does not. When this happens and there are no messages waiting to be sent, the sending end of the channel is monitoring its transmission queue, and not the TCP/IP connection. Therefore it does not recognize that the receiving end of the channel has stopped. In this situation, you must use the MQSeries Properties dialog box to stop the connection.

Why does the channel group not start?

If the channel group icon shows an exclamation mark or a cross when you start the channel group, one or more of the channels in the group are not running correctly. You should check the status of each channel. The following list shows status messages and possible causes for each message:

Status	**Explanation**
User exit error	There is a problem associated with a user-written exit program.
Invalid code page	The code pages of the sending and receiving MCAs are different. MQSeries for Windows cannot convert between the two code pages.

Diagnosing problems

Configuration error	There is a problem with the channel definition; for example, the connection name is unknown.
Queue manager error	The MCA has received an unexpected return code from the queue manager.
TCP/IP error	The network is not available, the remote system is not responding, the listener is not running, or there is a problem with TCP/IP.
System error	There is either a resource error (for example, no memory available) or an internal error in MQSeries for Windows.
Error at remote MCA	The remote MCA has had a problem and has stopped the channel, or bad data has been received. This message can also occur if the remote MCA stops the channel when the user stops the connection.
Message sequence error	The channel has sent or received a message whose sequence number does not match the one expected.
Unknown error	Any situation not covered by the other status messages.

For each of the preceding status messages, you can do any of the following:

- Retry the operation
- Restart the workstation, then retry the operation
- Ask your MQ administrator for help

Unrecoverable system error

If you have an unrecoverable system error, for example a problem with the synchronization file, you may need to reset the channel. The synchronization file is named AMQRSYNA.DAT, and it is in the queue manager's directory. You may need to do one or more of the following, but only if you are an MQSeries administrator:

- Delete the synchronization file
- Reset the message sequence number using the MQSC command RESET CHANNEL
- Resolve the status of any in-doubt messages using the MQSC command RESOLVE CHANNEL

Problems with messages

This section outlines some possible problems with sending and receiving messages.

A message cannot be delivered

MQSeries for Windows does not support dead-letter queues, therefore a message that it cannot deliver can stop your system running.

If an MCA cannot deliver a message:

- The channel stops.
- An error status is set on the workstations at both ends of the channel.
- The unit of work is backed out, and the messages reappear on the transmission queue at the sending end of the channel. The sending channel is now blocked by the undeliverable message.

If a message cannot be delivered to a remote queue, you should check that the remote queue and its associated objects are defined to MQSeries for Windows.

Why are messages not being sent or received?

If your messages are not being sent or received, this indicates a problem with the channel. You can:

- Check the status of the channel in the Channel Status window of the MQSeries Properties dialog box
- Check that the queue, the transmission queue name (the `XMITQ` channel attribute), and the channel definitions are correct

Why do messages not appear on the queue?

If messages do not appear when you are expecting them, check for the following.

Has the message been put on the queue successfully?

If the message has not been put on the queue successfully, check the following:

- Has the queue been defined correctly? For example, is the MaxMsgLength attribute of the queue large enough to allow a message of the required size?
- Is the queue enabled for putting?
- Is the queue already full? This could mean that an application was unable to put the required message on the queue.
- Has another application got exclusive access to the queue?

Are you able to get any message from the queue?

If you cannot get any message from the queue, check the following:

- Can other applications get messages from the queue?
- Has another application got exclusive access to the queue?

Diagnosing problems

If you are developing an application, check the following:

- Do you need to take a syncpoint?

  If messages are being put or retrieved with syncpoint control, they are not available to other tasks until the unit of work has been committed.

- Is your wait interval long enough?

  You can set the wait interval as an option on the MQGET call. You should ensure that you are waiting long enough for a response.

- Are you waiting for a specific message that is identified by a message or correlation identifier (*MsgId* or *CorrelId*)?

  Check that you are waiting for a message with the correct *MsgId* or *CorrelId*. A successful MQGET call will set both these values to that of the message retrieved, so you may need to reset these values in order to get another message successfully.

  Also check if you can get another message from the queue.

- Was the message you are expecting defined as persistent?

  If not, and MQSeries for Windows has been restarted, the message will have been lost.

If many programs are serving the queue, they can conflict with one another. For example, suppose one program issues an MQGET call with a buffer length of zero to find out the length of the message, and then issues a specific MQGET call specifying the *MsgId* of that message. However, in the meantime, another program issues a successful MQGET call for that message, so the first program receives a completion code of MQRC_NO_MSG_AVAILABLE. Applications must be designed to cope with this situation.

Consider that the message could have been received, but that your application failed to process it in some way. For example, did an error in the expected format of the message cause your program to reject it?

The data is not converted

MQSeries for Windows does not convert data in messages to other code pages or integer representations. It converts only the message header. Therefore, any data conversion must be done by the queue manager that sends a message to an MQSeries for Windows queue manager.

Service information

The Service page of the MQSeries Properties dialog box provides information that may help you solve the problems of users of an MQSeries for Windows application. The information it provides includes:

- Information about the version of MQSeries for Windows you have installed, including the name of the directory in which it is installed
- The release levels of the MQSeries for Windows files, so you can determine if any maintenance fixes have been applied
- The amount of free disk space remaining on the drive on which MQSeries for Windows is installed and the amount of physical memory available
- Information about the TCP/IP product that is installed on the workstation
- Access to the channel logs (for more information, see “Channel logging”)

In addition, MQSeries for Windows writes the release levels of its files to a file named AMQLEVLW.LOG in the \Program Files\MQSeries for Windows\Data directory.

If you want to use the Verify function to test your MQ installation, start it from the Service page. For more information on this, see “Verifying your installation” on page 20.

Channel logging

On MQSeries for Windows, the message channel agents of each channel create a log file named CHANNEL.LOG. They put this file in the directory for the active queue manager. The log file records events such as the starting and stopping of the channel, and any errors that occur. Each entry in the log contains:

- A timestamp
- The name of the channel
- Upto three text strings
- Three hexadecimal codes

To view the channel log, open the Channel Logs tab on the Service page of the MQSeries Properties dialog box. To get more information about a log entry, click on the question button, then drag the question mark icon to that log entry.

The channel log has a fixed size, set by the ChannelLogSize keyword in the MQD file. When the log is full, MQ overwrites the oldest entry.

If you want to stop channel logging, specify the value 0 for the ChannelLogSize keyword. If you change the value of this keyword, MQ discards the current log.

Service trace

Service Trace

To trace the operation of an MQSeries for Windows application, use the Service Trace utility. This can provide trace information that includes:

- The time of each trace event
- The identifiers of the processes that are running
- The names of the MQI calls that have been issued
- Information associated with control and data flow

Note

The Service Trace utility is designed for use under the direction of IBM Service personnel. This chapter does not describe all the features of the trace output.

Starting tracing

Before you use the Service Trace utility, note that MQ traces only those processes that start ***after*** the Service Trace utility starts. Before using trace, ***close all MQSeries for Windows applications***. If you do not do this, the trace output will be incomplete.

You can start tracing in any of the following ways:

- To start tracing next time MQ starts (that is, next time you start your workstation), check the Start Service Trace box on the Options page of the MQSeries Properties dialog box. Remember to clear this check box if you do not want to use tracing next time you start MQ.
- To start tracing immediately, click on the Service Trace shortcut in the MQSeries for Windows folder.
- To start tracing immediately by typing a command, select **Run** from the Start menu on the Windows desktop. In the Run window, type the command:

  ```
  AMQXTRCW
  ```

  If you want to set some options before tracing starts, type the following command to open the Trace Utility Properties window:

  ```
  AMQXTRCW /S
  ```

Controlling the trace output

The trace output is always shown in the Service Trace Utility window, but you can also log it in a file. Do this by selecting an option from the File menu in the Service Trace Utility window. At any time, you can change whether the output is logged in a file.

In the Service Trace Utility window you can choose which trace points to trace, but you should do this only under the guidance of IBM service personnel.

If you log trace output in a file, by default it is written to the file named AMQTRACW.LOG in the trace output directory. In a default installation, this file is in the \Program Files\MQSeries for Windows\Data directory.

To log the trace output in a different file, you must supply the file name, and the path to it, when you type the start trace command. For example, to send trace output to the file NEWTRACE.LOG in the C:\MQTRACE directory:

1. On the Windows desktop, select **Run** from the Start menu.
2. In the Run window, type the following:

```
AMQXTRCW C:\MQTRACE\NEWTRACE.LOG
```

The new file name is displayed in the title bar of the Service Trace Utility window.

To set options for tracing, select **Properties** from the View menu of the Service Trace Utility window. You can set:

- How often the trace output is written to the log file.
- Whether the trace output is logged to one file continuously, or toggled between two files. If you choose two files, you can control the size of the files.
- Whether the existing log file is erased before you start logging.

If you choose to log the trace output frequently, this can make tracing run very slowly, especially during startup. If your application uses timers, they may time out unexpectedly, causing the application to fail. Use frequent logging only under the direction of IBM service personnel. You are advised ***not*** to log frequently when you are running applications in a production environment.

Stopping tracing

Stop tracing either by selecting **Exit** from the File menu of the Service Trace Utility window, or by pressing ALT+F4 in the Service Trace Utility window.

When you do this, all tracing is stopped and the Service Trace Utility window is closed.

Monitoring MQSeries events

MQSeries instrumentation events provide information about errors, warnings, and other significant occurrences on a queue manager or channel. You can use these events to monitor the operation of these objects.

This section tells you how to capture MQ events on MQSeries for Windows. If you need more information:

- For a list of the events that MQSeries for Windows generates, see “MQSeries events generated by MQSeries for Windows” on page 152.
- For more information on MQ events, see the *MQSeries Programmable System Management* manual.

MQ events

When an event occurs on a queue manager or channel, that object puts an *event message* on an *event queue*. For each queue manager, each category of event has its own event queue. All events in that category result in an event message being put on the same queue:

This event queue:	**Contains messages from:**
SYSTEM.ADMIN.QMGR.EVENT	Queue manager events
SYSTEM.ADMIN.PERFM.EVENT	Performance events
SYSTEM.ADMIN.CHANNEL.EVENT	Channel events

MQSeries for Windows defines the queue manager event queue and the performance event queue in the file AMQSCOMW.TST. In a default installation, this file is supplied in the \Program Files\MQSeries for Windows\QMgrs directory. MQ runs this file every time you create a queue manager so these two event queues are always available.

However, MQ ***does not*** define the channel event queue for you, so the default action is ***not to generate channel events***. This is because, once you have defined a channel event queue, you cannot stop channel event messages being generated. If you want MQ to generate channel events, you must define the channel event queue yourself using the name SYSTEM.ADMIN.CHANNEL.EVENT. To make it easier for you to define this queue, the supplied file AMQSCOMW.TST contains a commented-out definition of it. Either remove the comments from this file and run the file, or use the attributes described in the file when you create the queue using the MQSeries Properties dialog box.

You can stop channel events by disabling put operations on the channel event queue.

MQSeries events generated by MQSeries for Windows

MQSeries for Windows Version 2.1 generates a subset of the MQSeries instrumentation events. This subset is shown in Table 15.

For a description of the message generated by each event, see the *MQSeries Programmable System Management* manual.

Note: Version 2.0 of MQSeries for Windows does not generate MQSeries events.

Table 15 (Page 1 of 2). MQSeries events

Event	**Generated by V2.1**
Alias Base Queue Type Error	Yes
Bridge Started	No
Bridge Stopped	No
Channel Activated	No
Channel Auto-definition Error	No
Channel Auto-definition OK	No
Channel Conversion Error	Yes (1)

Table 15 (Page 2 of 2). MQSeries events

Event	**Generated by V2.1**
Channel Not Activated	Yes (1)
Channel Started	Yes (1)
Channel Stopped	Yes (1)
Default Transmission Queue Type Error	Yes
Default Transmission Queue Usage Error	Yes
Get Inhibited	Yes
Not Authorized (type 1)	No
Not Authorized (type 2)	No
Not Authorized (type 3)	No
Not Authorized (type 4)	No
Put Inhibited	Yes
Queue Depth High	Yes
Queue Depth Low	Yes
Queue Full	Yes
Queue Manager Active	Yes
Queue Manager Not Active	Yes
Queue Service Interval High	Yes
Queue Service Interval OK	Yes
Queue Type Error	Yes
Remote Queue Name Error	Yes
Transmission Queue Type Error	Yes
Transmission Queue Usage Error	Yes
Unknown Alias Base Queue	Yes
Unknown Default Transmission Queue	Yes
Unknown Object Name	Yes
Unknown Remote Queue Manager	Yes
Unknown Transmission Queue	Yes

Note: (1) If you want MQ to generate channel events, you must create the channel event queue yourself. For more information, see “Monitoring MQSeries events” on page 151.

MQ events

Part 3. For application programmers

Chapter 15. Writing applications using the MQI on Windows

When you write an MQ application to run on MQSeries for Windows, you need the following information:

Information on how to design an application
For this information, see the *MQSeries Application Programming Guide*.

Information about the MQI
This chapter describes the MQI calls, data types, and structures in the programming languages that MQSeries for Windows supports. It has the following sections:

- "Using the C programming language"
- "Using the Visual Basic programming language" on page 168

Information about how to migrate an existing application to Windows
If you have written MQSeries applications before, or you are migrating an existing MQSeries application to Windows from another operating system, note that MQSeries for Windows does not support the full MQI. Make sure you read Chapter 16, "How the MQI differs on MQSeries for Windows" on page 175 to understand the differences.

Examples of existing applications
Chapter 17, "Understanding the sample programs" on page 185 describes the design of the sample programs supplied with MQSeries for Windows.

You can write applications for MQSeries for Windows using either of the following programming languages:

- C
- Visual Basic

For a list of the compilers you can use, see "Required software" on page 12.

Using the C programming language

This section describes the support that MQSeries for Windows provides for the C programming language. It contains:

- "Considerations for the C language" on page 158
- "MQI calls in C" on page 161
- "Elementary data types in C" on page 165
- "Structure data types in C" on page 166

The MQI in C

Considerations for the C language

This section gives information you need before you start to use the MQI in the C programming language.

Header files in C

Header files are provided as part of the definition of the MQI to assist with the writing of C application programs that use message queuing. These header files are summarized in Table 16.

Table 16. C header files

File name	Contents
CMQC.H	Call prototypes, data types, and named constants for the main MQI
CMQXC.H	Call prototypes, data types, and named constants for the channel exits
CMQCFC.H	Data types and named constants for the PCF commands

To improve the portability of applications, it is recommended that you code the name of the header file in lowercase on the **#include** preprocessor directive:

```
#include "cmqc.h"
```

Your applications must be 32-bit, so link your programs with the library MQM.LIB.

In a default installation, the include files (.H) are supplied in the \Program Files\MQSeries for Windows\Include subdirectory. The library files are supplied in the \Program Files\MQSeries for Windows\Lib subdirectory.

Parameters of the MQI calls

Parameters that are *input-only* and of type MQHCONN, MQHOBJ, or MQLONG are passed by value; for all other parameters, the *address* of the parameter is passed by value.

Not all parameters that are passed by address need to be specified every time a function is invoked. Where a particular parameter is not required, a null pointer can be specified as the parameter on the function invocation, in place of the address of the parameter data. Parameters for which this is possible are identified in the call descriptions.

No parameter is returned as the value of the function; in C terminology, this means that all functions return **void**.

The attributes of the function are defined by the MQENTRY macro variable; the value of this macro variable depends on the environment.

Parameters with undefined data type

The MQGET, MQPUT, and MQPUT1 functions each have one parameter that has an undefined data type; this is the *Buffer* parameter. This parameter is used to send and receive the application's message data.

Parameters of this sort are shown in the C examples as arrays of MQBYTE. It is valid to declare the parameters in this way, but it is usually more convenient to declare them as the particular structure that describes the layout of the data in the message. The function parameter is declared as a pointer-to-void, and so the address of any sort of data can be specified as the parameter on the function invocation.

Data types

All data types are defined by means of the C **typedef** statement. For each data type, the corresponding pointer data type is also defined. The name of the pointer data type is the name of the elementary or structure data type prefixed with the letter 'P' to denote a pointer. The attributes of the pointer are defined by the MQPOINTER macro variable; the value of this macro variable depends on the environment. The following illustrates how pointer data types are declared:

```
#define MQPOINTER _far *           /* depends on environment */
...
typedef MQLONG MQPOINTER PMQLONG;  /* pointer to MQLONG      */
typedef MQMD   MQPOINTER PMQMD;    /* pointer to MQMD        */
```

Manipulating binary strings

Strings of binary data are declared as one of the MQBYTE*n* data types. Whenever fields of this type are copied, compared, or set, the C functions **memcpy**, **memcmp**, or **memset** should be used; for example:

```
#include <string.h>
#include "cmqc.h"

MQMD MyMsgDesc;

memcpy(MyMsgDesc.MsgId,           /* set MsgId field to nulls     */
       MQMI_NONE,                 /* ... using named constant     */
       sizeof(MyMsgDesc.MsgId));

memset(MyMsgDesc.CorrelId,        /* set CorrelId field to nulls  */
       0x00,                      /* ... using a different method */
       sizeof(MQBYTE24));
```

Do not use the string functions **strcpy**, **strcmp**, **strncpy**, or **strncmp**, because these do not work correctly for data declared with the MQBYTE*n* data types.

Manipulating character strings

When the queue manager returns character data to the application, the queue manager always pads the character data with blanks to the defined length of the field; the queue manager *does not* return null-terminated strings. Therefore, when copying, comparing, or concatenating such strings, the string functions **strncpy**, **strncmp**, or **strncat** should be used.

Do not use the string functions, which require the string to be terminated by a null (**strcpy**, **strcmp**, **strcat**). Also, do not use the function **strlen** to determine the length of the string; use instead the **sizeof** function to determine the length of the field.

Initial values for structures

The header file CMQC defines various macro variables that may be used to provide initial values for the message queuing structures when instances of those structures are declared. These macro variables have names of the form MQ*xxx*_DEFAULT, where MQ*xxx* represents the name of the structure. They are used in the following way:

```
MQMD   MyMsgDesc = {MQMD_DEFAULT};
MQPMO  MyPutOpts = {MQPMO_DEFAULT};
```

For some character fields (for example, the *StrucId* fields which occur in most structures, or the *Format* field which occurs in MQMD), the MQI defines particular values that are valid. For each of the valid values, *two* macro variables are provided:

- One macro variable defines the value as a string whose length excluding the implied null matches exactly the defined length of the field. For example, for the *Format* field in MQMD the following macro variable is provided (the symbol "ƀ" represents a blank character):

  ```
  #define MQFMT_STRING "MQSTRƀƀƀ"
  ```

  Use this form with the **memcpy** and **memcmp** functions.

- The other macro variable defines the value as an array of characters; the name of this macro variable is the name of the string form suffixed with "_ARRAY". For example:

  ```
  #define MQFMT_STRING_ARRAY 'M','Q','S','T','R','ƀ','ƀ','ƀ'
  ```

  Use this form to initialize the field when an instance of the structure is declared with values different from those provided by the MQMD_DEFAULT macro variable.[1]

Initial values for dynamic structures

When a variable number of instances of a structure is required, the instances are usually created in main storage obtained dynamically using the **calloc** or **malloc** functions. To initialize the fields in such structures, the following technique is recommended:

1. Declare an instance of the structure using the appropriate MQ*xxx*_DEFAULT macro variable to initialize the structure. This instance becomes the "model" for other instances:

   ```
   MQMD Model = {MQMD_DEFAULT};  /* declare model instance */
   ```

[1] This is not always necessary; in some environments the string form of the value can be used in both situations. However, the array form is recommended for declarations, since this is required for compatibility with the C++ programming language.

The **static** or **auto** keywords can be coded on the declaration in order to give the model instance static or dynamic lifetime, as required.

2. Use the **calloc** or **malloc** functions to obtain storage for a dynamic instance of the structure:

   ```
   PMQMD Instance;
   Instance = malloc(sizeof(MQMD));  /* get storage for dynamic instance */
   ```

3. Use the **memcpy** function to copy the model instance to the dynamic instance:

   ```
   memcpy(Instance,&Model,sizeof(MQMD));  /* initialize dynamic instance */
   ```

Notational conventions

The sections that follow show how the:

- Calls should be invoked
- Parameters should be declared
- Various data types should be declared

In a number of cases, parameters are arrays whose size is not fixed. For these, a lowercase ‘n’ is used to represent a numeric constant. When the declaration for that parameter is coded, the ‘n’ must be replaced by the numeric value required.

MQI calls in C

MQBACK

```
MQBACK (Hconn, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;     /* Connection handle */
MQLONG   CompCode;  /* Completion code */
MQLONG   Reason;    /* Reason code qualifying CompCode */
```

MQCLOSE

```
MQCLOSE (Hconn, &Hobj, Options, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;     /* Connection handle */
MQHOBJ   Hobj;      /* Object handle */
MQLONG   Options;   /* Options that control the action of MQCLOSE */
MQLONG   CompCode;  /* Completion code */
MQLONG   Reason;    /* Reason code qualifying CompCode */
```

The MQI in C

MQCMIT

```
MQCMIT (Hconn, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;     /* Connection handle */
MQLONG   CompCode;  /* Completion code */
MQLONG   Reason;    /* Reason code qualifying CompCode */
```

MQCONN

```
MQCONN (Name, &Hconn, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQCHAR48  Name;      /* Name of queue manager */
MQHCONN   Hconn;     /* Connection handle */
MQLONG    CompCode;  /* Completion code */
MQLONG    Reason;    /* Reason code qualifying CompCode */
```

MQDISC

```
MQDISC (&Hconn, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;     /* Connection handle */
MQLONG   CompCode;  /* Completion code */
MQLONG   Reason;    /* Reason code qualifying CompCode */
```

MQGET

```
MQGET (Hconn, Hobj, &MsgDesc, &GetMsgOpts, BufferLength, Buffer,
      &DataLength, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;         /* Connection handle */
MQHOBJ   Hobj;          /* Object handle */
MQMD     MsgDesc;       /* Message descriptor */
MQGMO    GetMsgOpts;    /* Options that control the action of MQGET */
MQLONG   BufferLength;  /* Length in bytes of the Buffer area */
MQBYTE   Buffer[n];     /* Area to contain the message data */
MQLONG   DataLength;    /* Length of the message */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

MQINQ

```
MQINQ (Hconn, Hobj, SelectorCount, Selectors, IntAttrCount, IntAttrs,
      CharAttrLength, CharAttrs, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;          /* Connection handle */
MQHOBJ   Hobj;           /* Object handle */
MQLONG   SelectorCount;  /* Count of selectors */
MQLONG   Selectors[n];   /* Array of attribute selectors */
MQLONG   IntAttrCount;   /* Count of integer attributes */
MQLONG   IntAttrs[n];    /* Array of integer attributes */
MQLONG   CharAttrLength; /* Length of character attributes buffer */
MQCHAR   CharAttrs[n];   /* Character attributes */
MQLONG   CompCode;       /* Completion code */
MQLONG   Reason;         /* Reason code qualifying CompCode */
```

MQOPEN

```
MQOPEN (Hconn, &ObjDesc, Options, &Hobj, &CompCode,
       &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;     /* Connection handle */
MQOD     ObjDesc;   /* Object descriptor */
MQLONG   Options;   /* Options that control the action of MQOPEN */
MQHOBJ   Hobj;      /* Object handle */
MQLONG   CompCode;  /* Completion code */
MQLONG   Reason;    /* Reason code qualifying CompCode */
```

MQPUT

```
MQPUT (Hconn, Hobj, &MsgDesc, &PutMsgOpts, BufferLength, Buffer,
      &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;         /* Connection handle */
MQHOBJ   Hobj;          /* Object handle */
MQMD     MsgDesc;       /* Message descriptor */
MQPMO    PutMsgOpts;    /* Options that control the action of MQPUT */
MQLONG   BufferLength;  /* Length of the message in Buffer */
MQBYTE   Buffer[n];     /* Message data */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

MQPUT1

```
MQPUT1 (Hconn, &ObjDesc, &MsgDesc, &PutMsgOpts,
       BufferLength, Buffer, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;         /* Connection handle */
MQOD     ObjDesc;       /* Object descriptor */
MQMD     MsgDesc;       /* Message descriptor */
MQPMO    PutMsgOpts;    /* Options that control the action of MQPUT1 */
MQLONG   BufferLength;  /* Length of the message in Buffer */
MQBYTE   Buffer[n];     /* Message data */
MQLONG   CompCode;      /* Completion code */
MQLONG   Reason;        /* Reason code qualifying CompCode */
```

MQSET

```
MQSET (Hconn, Hobj, SelectorCount, Selectors, IntAttrCount, IntAttrs,
      CharAttrLength, CharAttrs, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;           /* Connection handle */
MQHOBJ   Hobj;            /* Object handle */
MQLONG   SelectorCount;   /* Count of selectors */
MQLONG   Selectors[n];    /* Array of attribute selectors */
MQLONG   IntAttrCount;    /* Count of integer attributes */
MQLONG   IntAttrs[n];     /* Array of integer attributes */
MQLONG   CharAttrLength;  /* Length of character attributes buffer */
MQCHAR   CharAttrs[n];    /* Character attributes */
MQLONG   CompCode;        /* Completion code */
MQLONG   Reason;          /* Reason code qualifying CompCode */
```

Elementary data types in C

Table 17. Elementary data types in C

Data type	Representation
MQBYTE	`typedef unsigned char MQBYTE;`
MQBYTE16	`typedef MQBYTE MQBYTE16[16];`
MQBYTE24	`typedef MQBYTE MQBYTE24[24];`
MQBYTE32	`typedef MQBYTE MQBYTE32[32];`
MQBYTE64	`typedef MQBYTE MQBYTE64[64];`
MQCHAR	`typedef char MQCHAR;`
MQCHAR4	`typedef MQCHAR MQCHAR4[4];`
MQCHAR8	`typedef MQCHAR MQCHAR8[8];`
MQCHAR12	`typedef MQCHAR MQCHAR12[12];`
MQCHAR16	`typedef MQCHAR MQCHAR16[16];`
MQCHAR28	`typedef MQCHAR MQCHAR28[28];`
MQCHAR32	`typedef MQCHAR MQCHAR32[32];`
MQCHAR48	`typedef MQCHAR MQCHAR48[48];`
MQCHAR64	`typedef MQCHAR MQCHAR64[64];`
MQCHAR128	`typedef MQCHAR MQCHAR128[128];`
MQCHAR256	`typedef MQCHAR MQCHAR256[256];`
MQHCONN	`typedef MQLONG MQHCONN;`
MQHOBJ	`typedef MQLONG MQHOBJ;`
MQLONG	`typedef long MQLONG;`
PMQLONG	`typedef MQLONG MQPOINTER PMQLONG;`

The MQI in C

Structure data types in C

MQGMO–Get-message options

```
typedef struct tagMQGMO {
  MQCHAR4   StrucId;       /* Structure identifier */
  MQLONG    Version;       /* Structure version number */
  MQLONG    Options;       /* Options that control the action of
                              MQGET */
  MQLONG    WaitInterval;  /* Wait interval */
  MQLONG    Signal1;       /* Signal */
  MQLONG    Signal2;       /* Signal message identifier */
  MQCHAR48  ResolvedQName; /* Resolved name of destination queue */
 } MQGMO;
```

MQMD–Message descriptor

```
typedef struct tagMQMD {
  MQCHAR4   StrucId;          /* Structure identifier */
  MQLONG    Version;          /* Structure version number */
  MQLONG    Report;           /* Report options */
  MQLONG    MsgType;          /* Message type */
  MQLONG    Expiry;           /* Expiry time */
  MQLONG    Feedback;         /* Feedback or reason code */
  MQLONG    Encoding;         /* Data encoding */
  MQLONG    CodedCharSetId;   /* Coded character set identifier */
  MQCHAR8   Format;           /* Format name */
  MQLONG    Priority;         /* Message priority */
  MQLONG    Persistence;      /* Message persistence */
  MQBYTE24  MsgId;            /* Message identifier */
  MQBYTE24  CorrelId;         /* Correlation identifier */
  MQLONG    BackoutCount;     /* Backout counter */
  MQCHAR48  ReplyToQ;         /* Name of reply-to queue */
  MQCHAR48  ReplyToQMgr;      /* Name of reply queue manager */
  MQCHAR12  UserIdentifier;   /* User identifier */
  MQBYTE32  AccountingToken;  /* Accounting token */
  MQCHAR32  ApplIdentityData; /* Application data relating to
                                 identity */
  MQLONG    PutApplType;      /* Type of application that put the
                                 message */
  MQCHAR28  PutApplName;      /* Name of application that put the
                                 message */
  MQCHAR8   PutDate;          /* Date when message was put */
  MQCHAR8   PutTime;          /* Time when message was put */
  MQCHAR4   ApplOriginData;   /* Application data relating to origin */
 } MQMD;
```

MQOD–Object descriptor

```
typedef struct tagMQOD {
  MQCHAR4   StrucId;          /* Structure identifier */
  MQLONG    Version;          /* Structure version number */
  MQLONG    ObjectType;       /* Object type */
  MQCHAR48  ObjectName;       /* Object name */
  MQCHAR48  ObjectQMgrName;   /* Object queue manager name */
  MQCHAR48  DynamicQName;     /* Dynamic queue name */
  MQCHAR12  AlternateUserId;  /* Alternate user identifier */
 } MQOD;
```

MQPMO–Put-message options

```
typedef struct tagMQPMO {
  MQCHAR4   StrucId;           /* Structure identifier */
  MQLONG    Version;           /* Structure version number */
  MQLONG    Options;           /* Options that control the action of
                                  MQPUT or MQPUT1 */
  MQLONG    Timeout;           /* Reserved */
  MQHOBJ    Context;           /* Object handle of input queue */
  MQLONG    KnownDestCount;    /* Reserved */
  MQLONG    UnknownDestCount;  /* Reserved */
  MQLONG    InvalidDestCount;  /* Reserved */
  MQCHAR48  ResolvedQName;     /* Resolved name of destination queue */
  MQCHAR48  ResolvedQMgrName;  /* Resolved name of destination queue
                                  manager */
 } MQPMO;
```

MQXQH–Transmission queue header

```
typedef struct tagMQXQH {
  MQCHAR4   StrucId;         /* Structure identifier */
  MQLONG    Version;         /* Structure version number */
  MQCHAR48  RemoteQName;     /* Name of destination queue */
  MQCHAR48  RemoteQMgrName;  /* Name of destination queue manager */
  MQMD      MsgDesc;         /* Original message descriptor */
 } MQXQH;
```

The MQI in Visual Basic

Using the Visual Basic programming language

This section describes the support that MQSeries for Windows provides for the Visual Basic programming language. It contains:

- “Considerations for the Visual Basic language”
- “MQI calls in Visual Basic” on page 169
- “Elementary data types in Visual Basic” on page 172
- “Structure data types in Visual Basic” on page 173

Considerations for the Visual Basic language

This section gives information you need before you start to use the MQI in the Visual Basic programming language.

Header files in Visual Basic

Header (or form) files are provided as part of the definition of the MQI to assist with the writing of Visual Basic application programs that use message queuing. These header files are summarized in Table 18.

Table 18. Visual Basic header files

File name	Contents
CMQB.BAS	Call declarations, data types, and named constants for the main MQI.
CMQXB.BAS	Call declarations, data types, and named constants for the channel exits.

In a default installation, the form files (.BAS) are supplied in the \Program Files\MQSeries for Windows\Include subdirectory. The library files are supplied in the \Program Files\MQSeries for Windows\Lib subdirectory.

Parameters of the MQI calls

Parameters that are *input-only* and of type MQHCONN, MQHOBJ, or MQLONG are passed by value; all other parameters are passed by address.

Initial values for structures

The supplied header files define various subroutines that may be invoked to initialize the message queuing structures with the default values. These subroutines have names of the form **MQxxx_DEFAULTS**, where **MQxxx** represents the name of the structure. They are used in the following way:

```
MQMD_DEFAULTS (MyMsgDesc)     'Initialize message descriptor'
MQPMO_DEFAULTS (MyPutOpts)    'Initialize put-message options'
```

Notational conventions

The sections that follow show how to:

- Invoke the calls
- Declare the parameters
- Declare the data types

In some cases, parameters are arrays whose sizes are not fixed. For these, a lowercase ‘n’ represents a numeric constant. When you code the declaration for that parameter, you must replace the ‘n’ with the numeric value you require.

MQI calls in Visual Basic

MQBACK

```
MQBACK Hconn, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Hconn    As Long 'Connection handle'
Dim CompCode As Long 'Completion code'
Dim Reason   As Long 'Reason code qualifying CompCode'
```

MQCLOSE

```
MQCLOSE Hconn, Hobj, Options, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Hconn    As Long 'Connection handle'
Dim Hobj     As Long 'Object handle'
Dim Options  As Long 'Options that control the action of MQCLOSE'
Dim CompCode As Long 'Completion code'
Dim Reason   As Long 'Reason code qualifying CompCode'
```

MQCMIT

```
MQCMIT Hconn, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Hconn    As Long 'Connection handle'
Dim CompCode As Long 'Completion code'
Dim Reason   As Long 'Reason code qualifying CompCode'
```

MQCONN

```
MQCONN Name, Hconn, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Name     As String*48 'Name of queue manager'
Dim Hconn    As Long      'Connection handle'
Dim CompCode As Long      'Completion code'
Dim Reason   As Long      'Reason code qualifying CompCode'
```

The MQI in Visual Basic

MQDISC

```
MQDISC Hconn, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Hconn    As Long 'Connection handle'
Dim CompCode As Long 'Completion code'
Dim Reason   As Long 'Reason code qualifying CompCode'
```

MQGET

```
MQGET Hconn, Hobj, MsgDesc, GetMsgOpts, BufferLength, Buffer,
      DataLength, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Hconn        As Long   'Connection handle'
Dim Hobj         As Long   'Object handle'
Dim MsgDesc      As MQMD   'Message descriptor'
Dim GetMsgOpts   As MQGMO  'Options that control the action of MQGET'
Dim BufferLength As Long   'Length in bytes of the Buffer area'
Dim Buffer       As String 'Area to contain the message data'
Dim DataLength   As Long   'Length of the message'
Dim CompCode     As Long   'Completion code'
Dim Reason       As Long   'Reason code qualifying CompCode'
```

MQINQ

```
MQINQ Hconn, Hobj, SelectorCount, Selectors, IntAttrCount, IntAttrs,
      CharAttrLength, CharAttrs, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Hconn          As Long   'Connection handle'
Dim Hobj           As Long   'Object handle'
Dim SelectorCount  As Long   'Count of selectors'
Dim Selectors      As Long   'Array of attribute selectors'
Dim IntAttrCount   As Long   'Count of integer attributes'
Dim IntAttrs       As Long   'Array of integer attributes'
Dim CharAttrLength As Long   'Length of character attributes buffer'
Dim CharAttrs      As String 'Character attributes'
Dim CompCode       As Long   'Completion code'
Dim Reason         As Long   'Reason code qualifying CompCode'
```

MQOPEN

```
MQOPEN Hconn, ObjDesc, Options, Hobj, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Hconn    As Long 'Connection handle'
Dim ObjDesc  As MQOD 'Object descriptor'
Dim Options  As Long 'Options that control the action of MQOPEN'
Dim Hobj     As Long 'Object handle'
Dim CompCode As Long 'Completion code'
Dim Reason   As Long 'Reason code qualifying CompCode'
```

MQPUT

```
MQPUT Hconn, Hobj, MsgDesc, PutMsgOpts, BufferLength, Buffer, CompCode,
      Reason
```

Declare the parameters as follows:

```
Dim Hconn        As Long   'Connection handle'
Dim Hobj         As Long   'Object handle'
Dim MsgDesc      As MQMD   'Message descriptor'
Dim PutMsgOpts   As MQPMO  'Options that control the action of MQPUT'
Dim BufferLength As Long   'Length of the message in Buffer'
Dim Buffer       As String 'Message data'
Dim CompCode     As Long   'Completion code'
Dim Reason       As Long   'Reason code qualifying CompCode'
```

MQPUT1

```
MQPUT1 Hconn, ObjDesc, MsgDesc, PutMsgOpts, BufferLength, Buffer,
       CompCode, Reason
```

Declare the parameters as follows:

```
Dim Hconn        As Long   'Connection handle'
Dim ObjDesc      As MQOD   'Object descriptor'
Dim MsgDesc      As MQMD   'Message descriptor'
Dim PutMsgOpts   As MQPMO  'Options that control the action of MQPUT1'
Dim BufferLength As Long   'Length of the message in Buffer'
Dim Buffer       As String 'Message data'
Dim CompCode     As Long   'Completion code'
Dim Reason       As Long   'Reason code qualifying CompCode'
```

The MQI in Visual Basic

MQSET

```
MQSET Hconn, Hobj, SelectorCount, Selectors, IntAttrCount, IntAttrs,
      CharAttrLength, CharAttrs, CompCode, Reason
```

Declare the parameters as follows:

```
Dim Hconn          As Long   'Connection handle'
Dim Hobj           As Long   'Object handle'
Dim SelectorCount  As Long   'Count of selectors'
Dim Selectors      As Long   'Array of attribute selectors'
Dim IntAttrCount   As Long   'Count of integer attributes'
Dim IntAttrs       As Long   'Array of integer attributes'
Dim CharAttrLength As Long   'Length of character attributes buffer'
Dim CharAttrs      As String 'Character attributes'
Dim CompCode       As Long   'Completion code'
Dim Reason         As Long   'Reason code qualifying CompCode'
```

Elementary data types in Visual Basic

Table 19. Elementary data types in Visual Basic

Data type	Representation
MQBYTE	`String*1`
MQBYTE24	`String*24`
MQBYTE32	`String*32`
MQCHAR	`String*1`
MQCHAR4	`String*4`
MQCHAR8	`String*8`
MQCHAR12	`String*12`
MQCHAR28	`String*28`
MQCHAR32	`String*32`
MQCHAR48	`String*48`
MQCHAR64	`String*64`
MQCHAR128	`String*128`
MQCHAR256	`String*256`
MQHCONN	`Long`
MQHOBJ	`Long`
MQLONG	`Long`

Structure data types in Visual Basic

MQGMO—Get-message options

```
Type MQGMO
  StrucId       As String*4  'Structure identifier'
  Version       As Long      'Structure version number'
  Options       As Long      'Options that control the action of MQGET'
  WaitInterval  As Long      'Wait interval'
  Signal1       As Long      'Signal'
  Signal2       As Long      'Signal message identifier'
  ResolvedQName As String*48 'Resolved name of destination queue'
End Type
```

MQMD—Message descriptor

```
Type MQMD
  StrucId              As String*4  'Structure identifier'
  Version              As Long      'Structure version number'
  Report               As Long      'Report options'
  MsgType              As Long      'Message type'
  Expiry               As Long      'Expiry time'
  Feedback             As Long      'Feedback or reason code'
  Encoding             As Long      'Data encoding'
  CodedCharSetId       As Long      'Coded character set identifier'
  Format               As String*8  'Format name'
  Priority             As Long      'Message priority'
  Persistence          As Long      'Message persistence'
  MsgId(23)            As Byte      'Message identifier'
  CorrelId(23)         As Byte      'Correlation identifier'
  BackoutCount         As Long      'Backout counter'
  ReplyToQ             As String*48 'Name of reply-to queue'
  ReplyToQMgr          As String*48 'Name of reply queue manager'
  UserIdentifier       As String*12 'User identifier'
  AccountingToken(31)  As Byte      'Accounting token'
  ApplIdentityData     As String*32 'Application data relating to identity'
  PutApplType          As Long      'Type of application that put the message'
  PutApplName          As String*28 'Name of application that put the message'
  PutDate              As String*8  'Date when message was put'
  PutTime              As String*8  'Time when message was put'
  ApplOriginData       As String*4  'Application data relating to origin'
End Type
```

MQOD—Object descriptor

```
Type MQOD
  StrucId         As String*4  'Structure identifier'
  Version         As Long      'Structure version number'
  ObjectType      As Long      'Object type'
  ObjectName      As String*48 'Object name'
  ObjectQMgrName  As String*48 'Object queue manager name'
  DynamicQName    As String*48 'Dynamic queue name'
  AlternateUserId As String*12 'Alternate user identifier'
End Type
```

MQPMO—Put-message options

```
Type MQPMO
  StrucId          As String*4  'Structure identifier'
  Version          As Long      'Structure version number'
  Options          As Long      'Options that control the action of MQPUT or'
                                'MQPUT1'
  Timeout          As Long      'Reserved'
  Context          As Long      'Object handle of input queue'
  KnownDestCount   As Long      'Reserved'
  UnknownDestCount As Long      'Reserved'
  InvalidDestCount As Long      'Reserved'
  ResolvedQName    As String*48 'Resolved name of destination queue'
  ResolvedQMgrName As String*48 'Resolved name of destination queue manager'
End Type
```

MQXQH—Transmission queue header

```
Type MQXQH
  StrucId        As String*4  'Structure identifier'
  Version        As Long      'Structure version number'
  RemoteQName    As String*48 'Name of destination queue'
  RemoteQMgrName As String*48 'Name of destination queue manager'
  MsgDesc        As MQMD      'Original message descriptor'
End Type
```

Chapter 16. How the MQI differs on MQSeries for Windows

The Message Queue Interface (MQI) is the MQSeries application programming interface. MQSeries for Windows supports nearly all the features of the MQI; this chapter describes that support. It uses the following sections:

- “Restrictions in the MQI calls”
- “Using signaling with the MQGET call” on page 176
- “Restrictions in the MQI structures” on page 178
- “MQI attributes on Windows” on page 180

For a description of the full MQI, see the *MQSeries Application Programming Reference*.

Restrictions in the MQI calls

For each MQI call, this section describes the differences in processing in MQSeries for Windows.

Pointers

MQSeries for Windows cannot always verify that parameter pointers are valid. So if, for example, the address you pass as the Buffer parameter on an MQGET call cannot be accessed for the entire length given by the BufferLength parameter, an exception or unpredictable result can occur. To avoid this problem, always ensure that any parameters you pass on an MQI call are valid.

The MQBEGIN call

MQSeries for Windows does not support the MQBEGIN call. Do not use it in an application you want to run on MQSeries for Windows.

The MQCONN (Connect queue manager) call

If your application specifies a blank name in the Name parameter of the MQCONN call, the request is serviced by the running queue manager. MQSeries for Windows allows only one queue manager to run at a time, so the concept of a default queue manager does not apply.

MQSeries for Windows does not support queue manager groups, so you cannot use an asterisk (*) in the Name parameter of the MQCONN call.

The MQCONNX call

MQSeries for Windows does not support the MQCONNX call. Do not use it in an application you want to run on MQSeries for Windows.

Signaling

The MQOPEN (Open object) call

MQSeries for Windows supports the following options on the MQOPEN call:

- MQOO_INPUT_AS_Q_DEF
- MQOO_INPUT_SHARED
- MQOO_INPUT_EXCLUSIVE
- MQOO_BROWSE
- MQOO_OUTPUT
- MQOO_INQUIRE
- MQOO_SET
- MQOO_SET_ALL_CONTEXT
- MQOO_SET_IDENTITY_CONTEXT

If you have an existing MQSeries application that uses the following options, you do not have to change the MQOPEN call because MQSeries for Windows ignores these options:

- MQOO_ALTERNATE_USER_AUTHORITY
- MQOO_FAIL_IF_QUIESCING

The MQGET (Get message) call

Version 2.1 of MQSeries for Windows includes support for signaling (Version 2.0 does not). For information on how to use this feature, see "Using signaling with the MQGET call."

For a list of the options that MQSeries for Windows supports for the MQGET call, see "Get-message options structure (MQGMO)" on page 179.

The MQPUT and MQPUT1 (Put message) calls

For a list of the options that MQSeries for Windows supports for the MQPUT and MQPUT1 calls, see "Put-message options structure (MQPMO)" on page 180.

The MQSET and MQINQ (Set and inquire attribute) calls

For a list of the attributes that MQSeries for Windows supports, see "MQI attributes on Windows" on page 180.

Using signaling with the MQGET call

Signaling is an option on the MQGET call to allow the operating system to notify (or *signal*) a program when an expected message arrives on a queue. This is similar to the "get with wait" function because it allows your program to continue with other work while waiting for the signal. However, if you use signaling, you can free the thread and rely on the operating system to notify the program when a message arrives.

Signaling on Version 2.1 of MQSeries for Windows works in a similar way to signaling on MQSeries for MVS/ESA. The difference is that on Windows, MQ signals by sending a Windows message to a window that you specify.

To request a signal, do the following in the MQGMO structure you use on your MQGET call:

1. Set the MQGMO_SET_SIGNAL option.
2. Set in the WaitInterval field the time (in milliseconds) for which you want MQ to monitor the queue. This sets the maximum life of the signal. You can use the MQWI_UNLIMITED value to specify an unlimited life.
3. In the Signal1 field, specify the handle of the window to which you want the signal sent.
4. In the Signal2 field, specify an identifier for the signal message. Use a RegisterWindowMessage to find a suitable identifier.

For information on what MQ can return to your application when it makes an MQGET call using signaling, see “Reason codes when signaling.”

When a suitable message arrives, MQ sends a Windows message to the window you specified in your Signal1 field. MQ puts a completion code in the WPARAM field of the Windows message (see “Completion codes when signaling” on page 178). Your application must then make another MQGET call to get the message. However, if there is already a suitable message on the queue, MQ retrieves the message immediately without setting a signal. If you close the queue while the signal is set, MQ cancels the signal without sending a Windows message.

For more information on how to set a signal, see the *MQSeries Application Programming Guide* and the description of the MQGMO structure in the *MQSeries Application Programming Reference*.

Reason codes when signaling

When you set a signal using an MQGET call, you might see any of the following reason codes in addition to those the MQGET call usually returns.

MQRC_SIGNAL_OUTSTANDING
(2069, X'815')

This means there is already a signal or wait instruction set using the queue handle you specified in the MQGET call. You cannot set more than one signal or wait instruction for a particular queue (but you can set one on each of many queues). If you want to set a signal on this queue, you must wait for the first one you set to be delivered.

MQRC_SIGNAL_REQUEST_ACCEPTED
(2070, X'816')

This means MQ has set the signal you requested and is monitoring the queue to watch for the message you specified. MQ will deliver the signal when the message arrives; your application should wait for this signal.

MQRC_SIGNAL1_ERROR
(2099, X'833')

This means the window handle you specified in the Signal1 field is not valid. Reissue the MQGET call using a valid window handle.

Completion codes when signaling

The Windows message that MQ sends to your application to deliver the signal contains one of the following completion codes in the WPARAM field:

MQEC_MSG_ARRIVED
This means the message you set the signal for has arrived on the queue. To get the message, you must issue another MQGET call. But note that another application could get the message in the time between you receiving the signal and you issuing another MQGET call.

MQEC_WAIT_INTERVAL_EXPIRED
This means the wait interval you set has expired and the message you set the signal for did not arrive on the queue. MQ has canceled the signal.

MQEC_WAIT_CANCELED
This means the signal has been canceled. This happens, for example, if the queue manager stops or the attribute of the queue is changed so that MQGET calls are no longer allowed.

Restrictions in the MQI structures

MQSeries for Windows does not support the following MQI structures:

- MQBO
- MQCNO
- MQDH
- MQDLH
- MQIIH
- MQMDE
- MQOR
- MQPMR
- MQRMH
- MQRR
- MQTM
- MQTMC2

The remainder of this section describes how MQSeries for Windows uses the other MQI structures in ways that are different from other MQSeries products.

Get-message options structure (MQGMO)

In the Options field of the MQGMO structure:

- Version 2.0 of MQSeries for Windows supports the following values:
  - MQGMO_WAIT
  - MQGMO_NO_WAIT
  - MQGMO_SYNCPOINT
  - MQGMO_NO_SYNCPOINT
  - MQGMO_BROWSE_FIRST
  - MQGMO_BROWSE_NEXT
  - MQGMO_BROWSE_MSG_UNDER_CURSOR
  - MQGMO_MSG_UNDER_CURSOR
  - MQGMO_ACCEPT_TRUNCATED_MSG
  - MQGMO_NONE
- Version 2.1 of MQSeries for Windows supports the following values:
  - The values that Version 2.0 supports
  - MQGMO_SET_SIGNAL
  - MQGMO_SYNCPOINT_IF_PERSISTENT
- MQSeries for Windows ignores the MQGMO_FAIL_IF_QUIESCING value.

MQSeries for Windows supports all the other fields of Version 1 of the MQGMO structure.

Message descriptor structure (MQMD)

MQSeries for Windows Version 2.1 supports all the fields of Version 1 of the MQMD structure. It uses the following default values:

- PutApplType = `MQAT_WINDOWS_NT`
- UserIdentifier = the Windows user ID of the logged-on user

Object descriptor structure (MQOD)

MQSeries for Windows supports all the fields of Version 1 of the MQOD structure, but it ignores the contents of the AlternateUserId field.

MQI attributes

Put-message options structure (MQPMO)

In the Options field of the MQPMO structure:

- MQSeries for Windows supports the following values:
  - MQPMO_SYNCPOINT
  - MQPMO_NO_SYNCPOINT
  - MQPMO_NO_CONTEXT
  - MQPMO_DEFAULT_CONTEXT
  - MQPMO_SET_IDENTITY_CONTEXT
  - MQPMO_SET_ALL_CONTEXT
  - MQPMO_NONE
- If you have an existing MQSeries application that uses the following options, you do not have to change the MQPUT call because MQSeries for Windows ignores these values:
  - MQPMO_ALTERNATE_USER_AUTHORITY
  - MQPMO_FAIL_IF_QUIESCING

MQSeries for Windows supports all the other fields of Version 1 of the MQPMO structure.

MQI attributes on Windows

Queue managers, queues, and channels have properties called attributes. MQSeries for Windows does not support all the attributes provided by other MQSeries products. This section describes the attributes that MQSeries for Windows supports; to see which ones you can change, see the online help.

- The queue manager attributes are listed in Table 20 on page 181.
- The queue attributes are listed in Table 21 on page 182.
- The channel attributes are listed in Table 22 on page 183.

MQSeries for Windows does not support process definitions.

For information about the attributes you can use with the MQSC commands, see the online *MQSeries for Windows Command Reference*.

Table 20. Attributes of queue managers on Windows

Attribute	**Default value**
AuthorityEvent	MQEVR_DISABLED (1)
CodedCharSetId	
CommandInputQName	SYSTEM.ADMIN.COMMAND.QUEUE
CommandLevel	MQCMDL_LEVEL_110 (= 110)
DefXmitQName	blanks
InhibitEvent	MQEVR_DISABLED
LocalEvent	MQEVR_DISABLED
MaxHandles	256
MaxMsgLength	4 194 304 bytes
MaxPriority	9
MaxUncommittedMsgs	10000
PerformanceEvent	MQEVR_DISABLED
Platform	MQPL_WINDOWS_NT (= 11)
QMgrDesc	
QMgrName	
RemoteEvent	MQEVR_DISABLED
StartStopEvent	MQEVR_DISABLED
SyncPoint	MQSP_AVAILABLE

Note: (1) You can set this attribute to the default value only.

MQI attributes

Table 21. Attributes of queues on Windows

Attribute	**Default value**
BackoutRequeueQName	blank
BackoutThreshold	0
BaseQName	blank
CreationDate	
CreationTime	
CurrentQDepth	
DefinitionType	
DefInputOpenOption	MQOO_INPUT_SHARED
DefPersistence	MQPER_NOT_PERSISTENT
DefPriority	0
HardenGetBackout	MQQA_BACKOUT_NOT_HARDENED
InhibitGet	MQQA_GET_ALLOWED
InhibitPut	MQQA_PUT_ALLOWED
MaxMsgLength	4 194 304
MaxQDepth	5000
MsgDeliverySequence	MQMDS_PRIORITY
OpenInputCount	
OpenOutputCount	
QDepthHighEvent	MQEVR_DISABLED
QDepthHighLimit	80
QDepthLowEvent	MQEVR_DISABLED
QDepthLowLimit	20
QDepthMaxEvent	MQEVR_DISABLED
QDesc	blank
QName	
QServiceInterval	999 999 999
QServiceIntervalEvent	MQQSIE_NONE
QType	
RemoteQMgrName	blank
RemoteQName	blank
RetentionInterval	999 999 999
Shareability	MQQA_SHAREABLE
Usage	MQUS_NORMAL
XmitQName	blank

Table 22. Attributes of channels on Windows

Attribute	Default value
BatchSize	50
ChannelName	
ChannelType	
ConnectionName	
Desc	
DiscInterval	6000
FastMessages	MQNPMS_FAST
LongRetryCount	999 999 999
LongRetryInterval	1200
MaxMsgLength	4 194 304
MCAUserIdentifier	
MsgExit	
MsgUserData	
ReceiveExit	
ReceiveUserExit	
SecurityExit	
SecurityUserData	
SendExit	
SendUserData	
SeqNumberWrap	999 999 999
ShortRetryCount	10
ShortRetryInterval	60
TransportType	MQXPT_TCP
XmitQName	

MQI attributes

Chapter 17. Understanding the sample programs

This chapter describes the design of the MQSeries for Windows sample programs. The aim of the samples is to demonstrate the use of MQI calls inside Windows programs. The sections in this chapter are:

For information on how to ***run*** the sample programs, see:

- Chapter 12, "Running the sample programs on one workstation" on page 127
- Chapter 13, "Running the sample programs on two workstations" on page 133

The sample programs are available in two programming languages:

C
MQSeries for Windows provides the source code and executable files.

Visual Basic
MQSeries for Windows provides only the source code. You must build executable files before you can run these versions of the samples; see "Building the Visual Basic samples" on page 191.

The sample programs are based on the MQSeries family samples:

The Putting Messages sample
The Putting Messages sample puts a message on a specified queue. The supplied executable file is named AMQSPUTW.EXE.

The Browsing Messages sample
The Browsing Messages sample browses (that is, copies and displays) a message and its header. By specifying the same queue that you used with the Putting Messages sample, you can browse the messages you put on that queue. The supplied executable file is named AMQSBCGW.EXE.

The Getting Messages sample
The Getting Messages sample gets a message from a specified queue. By specifying the same queue that you used with the Putting Messages sample, you can retrieve the messages you put on that queue. The supplied executable file is named AMQSGETW.EXE.

Design of samples

General design

Each sample program uses a single window, and the design of this window is similar in each program (the window is shown in Figure 21 on page 128):

- The top part of the window is for working with queues.
- The middle part of the window is specific to the function of the sample (that is, putting, getting, or browsing messages).
- The bottom part of the window is for displaying the completion codes and reason codes for each of the MQI calls issued by the sample. This allows you to see a log of all the MQI calls issued by the sample. The most recently issued completion and reason codes are displayed at the top of the list.

Table 23 shows which MQI calls each sample program demonstrates.

Table 23. MQI calls used in the MQSeries for Windows sample programs

MQI call	**Putting Messages**	**Browsing Messages**	**Getting Messages**
MQCONN	Yes	Yes	Yes
MQOPEN for output	Yes	No	No
MQOPEN for input	No	No	Yes
MQOPEN for browsing	No	Yes	No
MQPUT	Yes	No	No
MQGET	No	No	Yes
MQGET for browsing	No	Yes	No
MQCLOSE	Yes	Yes	Yes
MQDISC	Yes	Yes	Yes

Notes:

1. The sample programs do not contain much code to check Windows errors. This is to make it easier to understand the MQSeries code. But this means you should take care if you want to use these samples as a basis for your own application development.
2. The MQSeries logic (including the MQI calls) is contained within conditional compile directives:

```
    #ifdef MQSERIES_CALLS

    #endif
```

   This is to enable you to identify the relevant sections of code more easily. `MQSERIES_CALLS` itself is defined in the make (.MAK) files.

On MQSeries for Windows, there can be only one active queue manager, so there is no need for the user of the sample to specify the name of a queue manager. The connection to the active queue manager is done during the processing of the WM_INITDIALOG message using the MQCONN call. The disconnection is done using the MQDISC call during the processing of the WM_CLOSE message. This means that the queue manager must be running before the sample starts; otherwise the connect fails.

The design of the Putting Messages sample program

The Putting Messages sample program demonstrates putting short messages (a maximum of 256 bytes) on a queue you specify when you start the program.

When the window is displayed, you must first decide which queue to open to put the messages on. Type the name of the chosen queue and select the **Open** push button. The sample then tries to open the queue for output using the MQOPEN call. The completion and reason codes are displayed at the bottom of the window in the **API Return Code** field. If successful, the **Open** push button is disabled, and the **Close** and **Put** push buttons are enabled.

When the queue has been opened successfully, you can put a message on the queue (using the MQPUT call) by typing the message in the **Data** field and selecting the **Put** push button. The completion and reason codes are displayed in the **API Return Code** field. If successful, the message data is also displayed in the **Log list** box. This is useful for correlating messages you have put with those you have retrieved or browsed using the other two samples. You can continue putting as many messages as you want on the same queue.

If you want to put messages on another queue, select the **Close** push button. This closes the queue using the MQCLOSE call. The completion and reason codes are displayed in the **API Return Code** field. The **Close** and **Put** push buttons are now disabled and the **Open** push button is enabled. Type in the name of the new queue to be opened and select the **Open** push button. Put messages on this new queue using the same method as before. When you want to end the sample program, first close any open queue, then select the **Exit** menu item from the File menu.

The design of the Browsing Messages sample program

The Browsing Messages sample program demonstrates browsing (that is, viewing) messages on a queue you specify when you start the sample. You can browse only the first 256 bytes of each message. The messages are not removed from the queue.

When the window is displayed, you must first decide which queue to open to browse the messages from. Type the name of the chosen queue and select the **Open** push button. The sample then tries to open the queue for browsing using the MQOPEN call. The completion and reason codes are displayed at the bottom of the window in the **API Return Code** field. If successful, the **Open** push button is disabled and the **Close** and **Browse** push buttons are enabled.

Getting Messages sample

If you have successfully opened the queue, you can browse a message from that queue (using the MQGET call) by selecting the **Browse** push button. The completion and reason codes are displayed in the **API Return Code** field. If successful, the message header is displayed in the **Header list** box, the message data is displayed in the **Data** field, and the message length is displayed in the **Length** field. The **Data** field is useful for correlating messages browsed with messages put using the Putting Messages sample. You can continue browsing as many messages as there are on the opened queue.

If you want to browse messages from another queue, first select the **Close** push button. This closes the queue using the MQCLOSE call. The completion and reason codes are displayed in the **API Return Code** field. The **Close** and **Browse** push buttons are now disabled and the **Open** push button is enabled. Type the name of the new queue to be opened and select the **Open** push button. Browse messages on this new queue using the same method. When you want to close the sample, first close any open queue, then select **Exit** from the File menu.

The design of the Getting Messages sample program

The Getting Messages sample program demonstrates getting short messages from a queue you specify when you start the program. The messages are removed from the queue. If they are longer than 256 bytes, the messages are truncated and the remainder discarded.

When the window is displayed, you must first decide which queue to open to get the messages from. Type the name of the chosen queue and select the **Open** push button. The sample then tries to open the queue for input using the MQOPEN call. The completion and reason codes are displayed at the bottom of the window in the **API Return Code** field. If successful, the **Open** push button is disabled, and the **Close** and **Get** push buttons are enabled.

When you have successfully opened the queue, you can get a message from it (using the MQGET call) by selecting the **Get** push button. The completion and reason codes are displayed in the **API Return Code** field. If successful, the message data is displayed in the **Data list** box and the message length is displayed in the **Length** field. This is useful for correlating messages retrieved with messages put or browsed using the other two samples. You can continue getting as many messages as there are on the open queue.

If you want to get messages from another queue, select the **Close** push button. This closes the queue using the MQCLOSE call. The completion and reason codes are displayed in the **API Return Code** field. The **Close** and **Get** push buttons are now disabled and the **Open** push button is enabled. Type in the name of the new queue to be opened and select the **Open** push button. Get messages from this new queue using the same method. When you want to close the sample, first close any open queue, then select the **Exit** menu item from the File menu.

Building the executable files

This section lists the files used by the sample programs and describes how to build the samples. Before you can run the Visual Basic samples, you need to build them because MQ does not provide executable files. You also need to build your own executable files if you change any of the sample source code.

The names of the sample files are of the form AMQS*xxx*W, where *xxx* represents the sample function (for example, GET). When you install MQSeries for Windows using the default options, the files for the samples are put in the \Program Files\MQSeries for Windows\Samples directory. The files for the C and Visual Basic versions of the samples are in separate subdirectories.

The following tables list the files that each sample program uses.

Table 24. Files for the C-language version of the Putting Messages sample

File name	Purpose
AMQSPUTW.C	Source file
AMQSPUTW.DEF	Module definition file
AMQSPUTW.H	Header file
AMQSPUTW.MAK	Make file
AMQSPUTW.RC	Resource file
AMQSPUTW.EXE	Executable file

Table 25. Files for the Visual Basic version of the Putting Messages sample

File name	Purpose
AMQSPUTB.FRM	Form file
AMQSPUTB.FRX	Graphics file
AMQSPUTB.VBP	Project file

Table 26. Files for the C-language version of the Browsing Messages sample

File name	Purpose
AMQSBCGW.C	Source file
AMQSBCGW.DEF	Module definition file
AMQSBCGW.H	Header file
AMQSBCGW.MAK	Make file
AMQSBCGW.RC	Resource file
AMQSBCGW.EXE	Executable file

Building the samples

Table 27. Files for the Visual Basic version of the Browsing Messages sample	
File name	**Purpose**
AMQSBCGB.FRM	Form file
AMQSBCGB.FRX	Graphics file
AMQSBCGB.VBP	Project file

Table 28. Files for the C-language version of the Getting Messages sample	
File name	**Purpose**
AMQSGETW.C	Source file
AMQSGETW.DEF	Module definition file
AMQSGETW.H	Header file
AMQSGETW.MAK	Make file
AMQSGETW.RC	Resource file
AMQSGETW.EXE	Executable file

Table 29. Files for the Visual Basic version of the Getting Messages sample	
File name	**Purpose**
AMQSGETB.FRM	Form file
AMQSGETB.FRX	Graphics file
AMQSGETB.VBP	Project file

Building the C samples

If you want to make changes to one of the C-language samples, first make a copy of the original. When you have made the changes, use the following procedure to rebuild the sample executable file. If you rename a source file, you must edit the make file to refer to the new name. You must use the 32-bit Microsoft Visual C++ Compiler, Version 4.0.

To build an executable file:

1. Open a command-prompt window.
2. If you have not already done so, run VCVARS32.BAT (which is supplied with Microsoft Visual C++) to ensure that the build environment is set up.
3. Run MQVARS.BAT (which is supplied in the \Program Files\MQSeries for Windows\Samples directory) to set the path for the sample files.
4. At the command prompt, type the appropriate NMAKE command.

   For example, to build the Putting Messages sample, type:

   ```
   NMAKE /A AMQSPUTW.MAK
   ```

   This creates the executable file named AMQSPUTW.EXE.

Building the Visual Basic samples

To build an executable file for one of the Visual Basic samples, use the following procedure. If you want to make changes to a source file, first make a copy of the original. If you rename a source file, you must edit the project file to refer to the new name. You must use the 32-bit Microsoft Visual Basic Compiler, Version 4.0.

To build an executable file, start your Visual Basic compiler and use it to:

1. Open the project file (.VBP)
2. Make an executable file

Building the samples

Part 4. Appendixes

Appendix A. Differences from the other members of the MQSeries family

MQSeries for Windows is a leaf-node queue manager that runs on Microsoft Windows 95 and Windows NT. It is designed to minimize system requirements so that workstations with relatively modest specifications can use commercial messaging. This appendix 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.

Channels MQSeries for Windows does not support:

- Channel autodefinition
- Channel heartbeats
- Multiple instances of the same channel

Version 2.1 of MQSeries for Windows allows you to define channels that carry nonpersistent messages at either normal or fast speed. The default speed is fast. However, note that fast messages do not wait for a syncpoint, so they may be lost if there is a transmission failure or if the channel stops when it is carrying the messages.

Channel exits
On MQSeries for Windows, you can write channel exits using the C programming language only.

Command server
Version 2.1 or MQSeries for Windows provides a command server so you can administer an MQSeries for Windows queue manager from a remote computer using a PCF application. For information on the PCF commands that Version 2.1 supports, see Chapter 11, “Making changes for a user” on page 119.

Configuration files
MQSeries for Windows does not support the configuration files (also called .ini files) that other MQSeries queue managers use to configure, for example, logs, communications, or installable services.

Family differences

Context passing
MQSeries for Windows does not copy context information from messages it receives from other queue managers. This is because an MQSeries for Windows queue manager is intended to be a leaf node only; it is not intended to be a server or an intermediate node in a network of queue managers (in which 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 perform the functions of some of these commands; for example, to start and stop a queue manager. For a comparison with the MQSeries control commands, see Table 30 on page 201.

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 (but it does convert the message header). Also, it cannot run data conversion exits.

For the applications running on these two queue managers to understand each other's data, the nonWindows queue manager must perform the data conversion.

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 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.

Distribution lists
MQSeries for Windows does not support distribution lists.

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. To see which events it generates, see “MQSeries events generated by MQSeries for Windows” on page 152. If you want MQ to generate channel events, you must create the channel event queue yourself (see “Monitoring MQSeries events” on page 151).

Version 2.0 of MQSeries for Windows 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 channel agent (MCA)

On MQSeries for Windows, you cannot replace the supplied MCA program with another program.

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 Chapter 16, “How the MQI differs on MQSeries for Windows” on page 175.

Message retry

MQSeries for Windows does not support message retry.

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 “MQSC commands supported by MQSeries for Windows” on page 116.

Using MQSeries for Windows, you can type MQSC commands in a window (and test and reissue them). You can also run MQSC command files. This is described in Chapter 10, “Using MQSC commands” on page 107.

You cannot issue MQSC commands on an MQSeries for Windows queue manager to run on another queue manager (known as indirect mode). Also, you cannot issue MQSC commands on another queue manager (using the `runmqsc /w` command) to run on an MQSeries for Windows queue manager.

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.

Family differences

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 Format (PCF) command messages
Version 2.1 supports many of the MQSeries PCF commands. To see which commands, see "PCF commands supported by MQSeries for Windows" on page 121. You can write PCF applications using the C programming language only.

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. They are described in Chapter 17, "Understanding the sample programs" on page 185.

Security manager
See *object authority manager*.

Signaling Version 2.1 of MQSeries for Windows supports signaling, so you can use the MQGMO_SET_SIGNAL option with the MQGET call in Windows applications. For more information, see "Using signaling with the MQGET call" on page 176.

Version 2.0 does not support signaling.

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 single-phase commitment of MQ resources (that is, the queue manager can commit or back out units of work).

Family differences

Appendix B. MQSeries control commands

In other MQSeries products, you type control commands at a command prompt. In MQSeries for Windows, you use the user interface to perform the functions of some of these control commands. Table 30 shows the control commands and how MQSeries for Windows provides the functions.

Table 30 (Page 1 of 2). Control commands and MQSeries for Windows

MQSeries command	**Description**	**Support on MQSeries for Windows V2.1**
CRTMQCVX	Create data conversion exit	Function not supported.
CRTMQM	Create queue manager	This function is provided by the MQSeries Properties dialog box of the Complete version and by the MQD file.
DLTMQM	Delete queue manager	This function is provided by the MQSeries Properties dialog box of the Complete version and by the MQD file.
DSPMQAUT	Display authority	Function not supported.
DSPMQCSV	Display command server	This function is provided by the MQSeries Properties dialog box.
DSPMQFLS	Display MQSeries files	Function not supported.
DSPMQTRN	Display MQSeries transactions	Function not supported.
ENDMQCSV	End the command server	This function is provided by the MQSeries Properties dialog box.
ENDMQM	Stop queue manager	This function is provided by the MQSeries Properties dialog box.
ENDMQTRC	End MQSeries trace	This function is provided by the Service Trace utility. Also, you can clear the Start Service Trace option in the MQSeries Properties dialog box.
RCDMQIMG	Record media image	Function not supported.
RCRMQOBJ	Recreate object	Function not supported.
RSVMQTRN	Resolve MQSeries transactions	Function not supported.
RUNMQCHI	Run channel initiator	Function not supported.
RUNMQCHL	Run channel	This function is provided by the MQSeries Properties dialog box.
RUNMQDLQ	Run dead-letter queue handler	Function not supported.

Control commands

Table 30 (Page 2 of 2). Control commands and MQSeries for Windows

MQSeries command	**Description**	**Support on MQSeries for Windows V2.1**
RUNMQLSR	Run listener	This function is provided by the MQSeries Properties dialog box. You must use either the MQSeries Properties dialog box or the MQD file to create a channel group that contains the listener. You can add the listener to an existing channel group using the MQSeries Properties dialog box.
RUNMQSC	Run MQSeries commands	This function is provided by the MQSeries Properties dialog box. You can also run an MQSC command file when you create a queue manager using the MQSeries Properties dialog box or the MQD file.
RUNMQTMC	Start client trigger monitor	Function not supported.
RUNMQTRM	Start trigger monitor	Function not supported.
SCMMQM	Add or delete the queue manager to the Windows NT Service Control Manager	Function not supported.
SETMQAUT	Set authority	Function not supported.
STRMQCSV	Start command server	This function is provided by the MQSeries Properties dialog box.
STRMQM	Start queue manager	This function is provided by the MQSeries Properties dialog box.
STRMQTRC	Start MQSeries trace	This function is provided by the MQSeries Properties dialog box.

Appendix C. Predefined queues and channels

This appendix lists the queues and channels that the supplied MQSC command files define.

Default and system objects

The sample MQSC command file AMQSCOMW.TST defines the MQSeries for Windows default and system objects. Each object definition contains a complete set of attributes for that object. Each time you create a new object, that object inherits its attributes from the default object of the same type, except for the attributes you explicitly specify.

For example, SYSTEM.DEFAULT.LOCAL.QUEUE contains the default definitions for a local queue. Consider what happens if you create a local queue using this MQSC command:

```
DEFINE QLOCAL ('PINK.QUEUE') PUT(DISABLED)
```

The queue named PINK.QUEUE takes the attributes of the SYSTEM.DEFAULT.LOCAL.QUEUE, except that the PUT attribute has a value of DISABLED, whereas the default value is PUT(ENABLED).

Note: You can change the default attributes of MQ objects by editing the definitions of the objects in the file AMQSCOMW.TST. If you do this, remember to keep a copy of the original file.

The system objects are required for the operation of a queue manager or channel. Table 31 on page 204 lists the objects defined in the supplied AMQSCOMW.TST file. In a default installation, the file is supplied in the \Program Files\MQSeries for Windows\QMgrs directory.

MQ automatically creates the objects specified in AMQSCOMW.TST whenever you create a queue manager.

Default queues and channels

Table 31. Objects defined in AMQSCOMW.TST

Object name	**Description**
SYSTEM.DEFAULT.ALIAS.QUEUE	Default alias queue
SYSTEM.DEFAULT.LOCAL.QUEUE	Default local queue
SYSTEM.DEFAULT.MODEL.QUEUE	Default model queue
SYSTEM.DEFAULT.REMOTE.QUEUE	Default remote queue
SYSTEM.DEF.SENDER	Default sender channel
SYSTEM.DEF.SERVER	Default server channel
SYSTEM.DEF.RECEIVER	Default receiver channel
SYSTEM.DEF.REQUESTER	Default requester channel
SYSTEM.CHANNEL.SYNCQ	Channel synchronization queue
SYSTEM.ADMIN.COMMAND.QUEUE	Administration command queue
SYSTEM.ADMIN.QMGR.EVENT	Queue manager event queue
SYSTEM.ADMIN.PERFM.EVENT	Performance event queue
SYSTEM.ADMIN.CHANNEL.EVENT	Channel event queue (see note)

Note: The definition of the channel event queue is commented out in this file. If you want MQ to generate channel events, you must create the channel event queue yourself using the definitions shown in this file. For more information, see “Monitoring MQSeries events” on page 151.

Objects for running the sample programs on one workstation

The sample MQSC command file AMQSCOSW.TST defines a local queue you can use with the supplied sample programs. The queue is named SYSTEM.SAMPLE.LOCAL. MQ runs this file if you select the **Load MQSC file for the sample programs** option when you create a queue manager.

Table 32 shows the object defined in the supplied AMQSCOSW.TST file. In a default installation, the file is supplied in the directory \Program Files\MQSeries for Windows\Samples.

Table 32. Object defined in AMQSCOSW.TST

Object name	**Description**
SYSTEM.SAMPLE.LOCAL	Sample local queue

Objects for running the sample programs on two workstations

If you want to use the sample programs to send messages from one workstation to another (as described in Chapter 13, “Running the sample programs on two workstations” on page 133), you need to define extra objects. The files VENUS.TST and MARS.TST define these objects.

MQ runs these files if you specify them as user or application command files when you create a queue manager.

Table 33 lists the objects defined in the file VENUS.TST. Table 34 lists the objects defined in the file MARS.TST. In a default installation, both files are supplied in the \Program Files\MQSeries for Windows\Samples directory.

Table 33. Objects defined in VENUS.TST

Object name	Description
SAMPLE.VENUS.XMIT	Transmission queue
SAMPLE.VENUS.REMOTE	Remote queue
SAMPLE.VENUS.LOCAL	Local queue
VENUS.TO.MARS	Server channel
MARS.TO.VENUS	Receiver channel

Table 34. Objects defined in MARS.TST

Object name	Description
SAMPLE.MARS.XMIT	Transmission queue
SAMPLE.MARS.REMOTE	Remote queue
SAMPLE.MARS.LOCAL	Local queue
MARS.TO.VENUS	Sender channel
VENUS.TO.MARS	Requester channel

Default queues and channels

Appendix D. Return codes

MQI calls return two types of code:

- A completion code to show whether the call completed successfully, completed partially, or failed. Table 35 shows these codes.
- A numeric reason code to explain a warning or failure. MQSC commands also return reason codes. Table 36 shows these codes.

If you want more information about reason codes:

- For those with a value of the form `2`*nnn* (and a name that starts with the characters `MQRC_`), see the *MQSeries Application Programming Reference*.
- For those with a value of the form `3`*nnn* or `4`*nnn* (and a name that starts with the characters `MQRCCF_`), see the *MQSeries Programmable System Management* manual.

Table 35. Completion codes returned by MQI calls

Numeric	Literal
0	MQCC_OK
1	MQCC_WARNING
2	MQCC_FAILED

Table 36 (Page 1 of 5). Reason codes returned by MQI calls and MQSC commands

Numeric	Literal
2001	MQRC_ALIAS_BASE_Q_TYPE_ERROR
2002	MQRC_ALREADY_CONNECTED
2003	MQRC_BACKED_OUT
2004	MQRC_BUFFER_ERROR
2005	MQRC_BUFFER_LENGTH_ERROR
2006	MQRC_CHAR_ATTR_LENGTH_ERROR
2007	MQRC_CHAR_ATTRS_ERROR
2008	MQRC_CHAR_ATTRS_TOO_SHORT
2009	MQRC_CONNECTION_BROKEN
2010	MQRC_DATA_LENGTH_ERROR
2011	MQRC_DYNAMIC_Q_NAME_ERROR
2012	MQRC_ENVIRONMENT_ERROR
2013	MQRC_EXPIRY_ERROR
2014	MQRC_FEEDBACK_ERROR
2016	MQRC_GET_INHIBITED
2017	MQRC_HANDLE_NOT_AVAILABLE
2018	MQRC_HCONN_ERROR
2019	MQRC_HOBJ_ERROR
2020	MQRC_INHIBIT_VALUE_ERROR
2021	MQRC_INT_ATTR_COUNT_ERROR
2022	MQRC_INT_ATTR_COUNT_TOO_SMALL
2023	MQRC_INT_ATTRS_ARRAY_ERROR
2024	MQRC_SYNCPOINT_LIMIT_REACHED
2025	MQRC_MAX_CONNS_LIMIT_REACHED
2026	MQRC_MD_ERROR
2027	MQRC_MISSING_REPLY_TO_Q
2029	MQRC_MSG_TYPE_ERROR
2030	MQRC_MSG_TOO_BIG_FOR_Q
2031	MQRC_MSG_TOO_BIG_FOR_Q_MGR
2033	MQRC_NO_MSG_AVAILABLE
2034	MQRC_NO_MSG_UNDER_CURSOR
2035	MQRC_NOT_AUTHORIZED
2036	MQRC_NOT_OPEN_FOR_BROWSE
2037	MQRC_NOT_OPEN_FOR_INPUT
2038	MQRC_NOT_OPEN_FOR_INQUIRE
2039	MQRC_NOT_OPEN_FOR_OUTPUT
2040	MQRC_NOT_OPEN_FOR_SET
2041	MQRC_OBJECT_CHANGED
2042	MQRC_OBJECT_IN_USE
2043	MQRC_OBJECT_TYPE_ERROR

Reason codes

Table 36 (Page 2 of 5). Reason codes returned by MQI calls and MQSC commands

Numeric	Literal
2044	MQRC_OD_ERROR
2045	MQRC_OPTION_NOT_VALID_FOR_TYPE
2046	MQRC_OPTIONS_ERROR
2047	MQRC_PERSISTENCE_ERROR
2048	MQRC_PERSISTENT_NOT_ALLOWED
2049	MQRC_PRIORITY_EXCEEDS_MAXIMUM
2050	MQRC_PRIORITY_ERROR
2051	MQRC_PUT_INHIBITED
2052	MQRC_Q_DELETED
2053	MQRC_Q_FULL
2055	MQRC_Q_NOT_EMPTY
2056	MQRC_Q_SPACE_NOT_AVAILABLE
2057	MQRC_Q_TYPE_ERROR
2058	MQRC_Q_MGR_NAME_ERROR
2059	MQRC_Q_MGR_NOT_AVAILABLE
2061	MQRC_REPORT_OPTIONS_ERROR
2062	MQRC_SECOND_MARK_NOT_ALLOWED
2063	MQRC_SECURITY_ERROR
2065	MQRC_SELECTOR_COUNT_ERROR
2066	MQRC_SELECTOR_LIMIT_EXCEEDED
2067	MQRC_SELECTOR_ERROR
2068	MQRC_SELECTOR_NOT_FOR_TYPE
2069	MQRC_SIGNAL_OUTSTANDING
2070	MQRC_SIGNAL_REQUEST_ACCEPTED
2071	MQRC_STORAGE_NOT_AVAILABLE
2072	MQRC_SYNCPOINT_NOT_AVAILABLE
2075	MQRC_TRIGGER_CONTROL_ERROR
2076	MQRC_TRIGGER_DEPTH_ERROR
2077	MQRC_TRIGGER_MSG_PRIORITY_ERR
2078	MQRC_TRIGGER_TYPE_ERROR
2079	MQRC_TRUNCATED_MSG_ACCEPTED
2080	MQRC_TRUNCATED_MSG_FAILED
2082	MQRC_UNKNOWN_ALIAS_BASE_Q
2085	MQRC_UNKNOWN_OBJECT_NAME
2086	MQRC_UNKNOWN_OBJECT_Q_MGR
2087	MQRC_UNKNOWN_REMOTE_Q_MGR
2090	MQRC_WAIT_INTERVAL_ERROR
2091	MQRC_XMIT_Q_TYPE_ERROR
2092	MQRC_XMIT_Q_USAGE_ERROR
2093	MQRC_NOT_OPEN_FOR_PASS_ALL
2094	MQRC_NOT_OPEN_FOR_PASS_IDENT
2095	MQRC_NOT_OPEN_FOR_SET_ALL
2096	MQRC_NOT_OPEN_FOR_SET_IDENT
2097	MQRC_CONTEXT_HANDLE_ERROR
2098	MQRC_CONTEXT_NOT_AVAILABLE
2099	MQRC_SIGNAL1_ERROR
2100	MQRC_OBJECT_ALREADY_EXISTS
2101	MQRC_OBJECT_DAMAGED
2102	MQRC_RESOURCE_PROBLEM
2103	MQRC_ANOTHER_Q_MGR_CONNECTED
2104	MQRC_UNKNOWN_REPORT_OPTION
2109	MQRC_SUPPRESSED_BY_EXIT
2110	MQRC_FORMAT_ERROR
2111	MQRC_SOURCE_CCSID_ERROR
2112	MQRC_SOURCE_INTEGER_ENC_ERROR
2113	MQRC_SOURCE_DECIMAL_ENC_ERROR
2114	MQRC_SOURCE_FLOAT_ENC_ERROR
2115	MQRC_TARGET_CCSID_ERROR
2116	MQRC_TARGET_INTEGER_ENC_ERROR
2117	MQRC_TARGET_DECIMAL_ENC_ERROR
2118	MQRC_TARGET_FLOAT_ENC_ERROR
2119	MQRC_NOT_CONVERTED
2120	MQRC_CONVERTED_MSG_TOO_BIG
2127	MQRC_ADAPTER_STORAGE_SHORTAGE
2129	MQRC_ADAPTER_CONN_LOAD_ERROR
2130	MQRC_ADAPTER_SERV_LOAD_ERROR
2131	MQRC_ADAPTER_DEFS_ERROR
2132	MQRC_ADAPTER_DEFS_LOAD_ERROR
2138	MQRC_ADAPTER_DISC_LOAD_ERROR
2140	MQRC_CICS_WAIT_FAILED
2143	MQRC_SOURCE_LENGTH_ERROR
2144	MQRC_TARGET_LENGTH_ERROR
2145	MQRC_SOURCE_BUFFER_ERROR
2146	MQRC_TARGET_BUFFER_ERROR
2150	MQRC_DBCS_ERROR
2151	MQRC_TRUNCATED
2157	MQRC_ASID_MISMATCH
2160	MQRC_CONN_ID_IN_USE

Table 36 (Page 3 of 5). Reason codes returned by MQI calls and MQSC commands

Numeric	Literal
2161	MQRC_Q_MGR_QUIESCING
2162	MQRC_Q_MGR_STOPPING
2163	MQRC_DUPLICATE_RECOV_COORD
2173	MQRC_PMO_ERROR
2182	MQRC_API_EXIT_NOT_FOUND
2183	MQRC_API_EXIT_LOAD_ERROR
2184	MQRC_REMOTE_Q_NAME_ERROR
2186	MQRC_GMO_ERROR
2192	MQRC_PAGESET_FULL
2193	MQRC_PAGESET_ERROR
2194	MQRC_NAME_NOT_VALID_FOR_TYPE
2195	MQRC_UNEXPECTED_ERROR
2196	MQRC_UNKNOWN_XMIT_Q
2197	MQRC_UNKNOWN_DEF_XMIT_Q
2198	MQRC_DEF_XMIT_Q_TYPE_ERROR
2199	MQRC_DEF_XMIT_Q_USAGE_ERROR
2201	MQRC_NAME_IN_USE
2202	MQRC_CONNECTION_QUIESCING
2203	MQRC_CONNECTION_STOPPING
2204	MQRC_ADAPTER_NOT_AVAILABLE
2206	MQRC_MSG_ID_ERROR
2207	MQRC_CORREL_ID_ERROR
2208	MQRC_FILE_SYSTEM_ERROR
2209	MQRC_NO_MSG_LOCKED
2217	MQRC_CONNECTION_NOT_AUTHORIZED
2218	MQRC_MSG_TOO_BIG_FOR_CHANNEL
2219	MQRC_CALL_IN_PROGRESS
2222	MQRC_Q_MGR_ACTIVE
2223	MQRC_Q_MGR_NOT_ACTIVE
2224	MQRC_Q_DEPTH_HIGH
2225	MQRC_Q_DEPTH_LOW
2226	MQRC_Q_SERVICE_INTERVAL_HIGH
2227	MQRC_Q_SERVICE_INTERVAL_OK
2280	MQRC_HCONFIG_ERROR
2281	MQRC_FUNCTION_ERROR
2282	MQRC_CHANNEL_STARTED
2283	MQRC_CHANNEL_STOPPED
2284	MQRC_CHANNEL_CONV_ERROR
2285	MQRC_SERVICE_NOT_AVAILABLE

Table 36 (Page 3 of 5). Reason codes returned by MQI calls and MQSC commands

Numeric	Literal
2286	MQRC_INITIALIZATION_FAILED
2287	MQRC_TERMINATION_FAILED
2288	MQRC_UNKNOWN_Q_NAME
2289	MQRC_SERVICE_ERROR
2290	MQRC_Q_ALREADY_EXISTS
2291	MQRC_USER_ID_NOT_AVAILABLE
2292	MQRC_UNKNOWN_ENTITY
2293	MQRC_UNKNOWN_AUTH_ENTITY
2294	MQRC_UNKNOWN_REF_OBJECT
3001	MQRCCF_CFH_TYPE_ERROR
3002	MQRCCF_CFH_LENGTH_ERROR
3003	MQRCCF_CFH_VERSION_ERROR
3004	MQRCCF_CFH_MSG_SEQ_NUMBER_ERR
3005	MQRCCF_CFH_CONTROL_ERROR
3006	MQRCCF_CFH_PARM_COUNT_ERROR
3007	MQRCCF_CFH_COMMAND_ERROR
3008	MQRCCF_COMMAND_FAILED
3009	MQRCCF_CFIN_LENGTH_ERROR
3010	MQRCCF_CFST_LENGTH_ERROR
3011	MQRCCF_CFST_STRING_LENGTH_ERR
3012	MQRCCF_FORCE_VALUE_ERROR
3013	MQRCCF_STRUCTURE_TYPE_ERROR
3014	MQRCCF_CFIN_PARM_ID_ERROR
3015	MQRCCF_CFST_PARM_ID_ERROR
3016	MQRCCF_MSG_LENGTH_ERROR
3017	MQRCCF_CFIN_DUPLICATE_PARM
3018	MQRCCF_CFST_DUPLICATE_PARM
3019	MQRCCF_PARM_COUNT_TOO_SMALL
3020	MQRCCF_PARM_COUNT_TOO_BIG
3021	MQRCCF_Q_ALREADY_IN_CELL
3022	MQRCCF_Q_TYPE_ERROR
3023	MQRCCF_MD_FORMAT_ERROR
3025	MQRCCF_REPLACE_VALUE_ERROR
3026	MQRCCF_CFIL_DUPLICATE_VALUE
3027	MQRCCF_CFIL_COUNT_ERROR
3028	MQRCCF_CFIL_LENGTH_ERROR
3029	MQRCCF_QUIESCE_VALUE_ERROR
3030	MQRCCF_MSG_SEQ_NUMBER_ERROR
3031	MQRCCF_PING_DATA_COUNT_ERROR

Reason codes

Table 36 (Page 4 of 5). Reason codes returned by MQI calls and MQSC commands

Numeric	Literal
3032	MQRCCF_PING_DATA_COMPARE_ERROR
3034	MQRCCF_CHANNEL_TYPE_ERROR
3035	MQRCCF_PARM_SEQUENCE_ERROR
3036	MQRCCF_XMIT_PROTOCOL_TYPE_ERR
3037	MQRCCF_BATCH_SIZE_ERROR
3038	MQRCCF_DISC_INT_ERROR
3039	MQRCCF_SHORT_RETRY_ERROR
3040	MQRCCF_SHORT_TIMER_ERROR
3041	MQRCCF_LONG_RETRY_ERROR
3042	MQRCCF_LONG_TIMER_ERROR
3043	MQRCCF_SEQ_NUMBER_WRAP_ERROR
3044	MQRCCF_MAX_MSG_LENGTH_ERROR
3045	MQRCCF_PUT_AUTH_ERROR
3046	MQRCCF_PURGE_VALUE_ERROR
3047	MQRCCF_CFIL_PARM_ID_ERROR
3048	MQRCCF_MSG_TRUNCATED
3049	MQRCCF_CCSID_ERROR
3050	MQRCCF_ENCODING_ERROR
3052	MQRCCF_DATA_CONV_VALUE_ERROR
3053	MQRCCF_INDOUBT_VALUE_ERROR
3054	MQRCCF_ESCAPE_TYPE_ERROR
3062	MQRCCF_CHANNEL_TABLE_ERROR
3063	MQRCCF_MCA_TYPE_ERROR
3064	MQRCCF_CHL_INST_TYPE_ERROR
3065	MQRCCF_CHL_STATUS_NOT_FOUND
4001	MQRCCF_OBJECT_ALREADY_EXISTS
4002	MQRCCF_OBJECT_WRONG_TYPE
4003	MQRCCF_LIKE_OBJECT_WRONG_TYPE
4004	MQRCCF_OBJECT_OPEN
4005	MQRCCF_ATTR_VALUE_ERROR
4006	MQRCCF_UNKNOWN_Q_MGR
4007	MQRCCF_Q_WRONG_TYPE
4008	MQRCCF_OBJECT_NAME_ERROR
4009	MQRCCF_ALLOCATE_FAILED
4010	MQRCCF_HOST_NOT_AVAILABLE
4011	MQRCCF_CONFIGURATION_ERROR
4012	MQRCCF_CONNECTION_REFUSED
4013	MQRCCF_ENTRY_ERROR
4014	MQRCCF_SEND_FAILED
4015	MQRCCF_RECEIVED_DATA_ERROR
4016	MQRCCF_RECEIVE_FAILED
4017	MQRCCF_CONNECTION_CLOSED
4018	MQRCCF_NO_STORAGE
4019	MQRCCF_NO_COMMS_MANAGER
4020	MQRCCF_LISTENER_NOT_STARTED
4024	MQRCCF_BIND_FAILED
4025	MQRCCF_CHANNEL_INDOUBT
4026	MQRCCF_MQCONN_FAILED
4027	MQRCCF_MQOPEN_FAILED
4028	MQRCCF_MQGET_FAILED
4029	MQRCCF_MQPUT_FAILED
4030	MQRCCF_PING_ERROR
4031	MQRCCF_CHANNEL_IN_USE
4032	MQRCCF_CHANNEL_NOT_FOUND
4033	MQRCCF_UNKNOWN_REMOTE_CHANNEL
4034	MQRCCF_REMOTE_QM_UNAVAILABLE
4035	MQRCCF_REMOTE_QM_TERMINATING
4036	MQRCCF_MQINQ_FAILED
4037	MQRCCF_NOT_XMIT_Q
4038	MQRCCF_CHANNEL_DISABLED
4039	MQRCCF_USER_EXIT_NOT_AVAILABLE
4040	MQRCCF_COMMIT_FAILED
4042	MQRCCF_CHANNEL_ALREADY_EXISTS
4043	MQRCCF_DATA_TOO_LARGE
4044	MQRCCF_CHANNEL_NAME_ERROR
4045	MQRCCF_XMIT_Q_NAME_ERROR
4047	MQRCCF_MCA_NAME_ERROR
4048	MQRCCF_SEND_EXIT_NAME_ERROR
4049	MQRCCF_SEC_EXIT_NAME_ERROR
4050	MQRCCF_MSG_EXIT_NAME_ERROR
4051	MQRCCF_RCV_EXIT_NAME_ERROR
4052	MQRCCF_XMIT_Q_NAME_WRONG_TYPE
4053	MQRCCF_MCA_NAME_WRONG_TYPE
4054	MQRCCF_DISC_INT_WRONG_TYPE
4055	MQRCCF_SHORT_RETRY_WRONG_TYPE
4056	MQRCCF_SHORT_TIMER_WRONG_TYPE
4057	MQRCCF_LONG_RETRY_WRONG_TYPE
4058	MQRCCF_LONG_TIMER_WRONG_TYPE

Table 36 (Page 5 of 5). Reason codes returned by MQI calls and MQSC commands

Numeric	Literal
4059	MQRCCF_PUT_AUTH_WRONG_TYPE
4061	MQRCCF_MISSING_CONN_NAME
4062	MQRCCF_CONN_NAME_ERROR
4063	MQRCCF_MQSET_FAILED
4064	MQRCCF_CHANNEL_NOT_ACTIVE
4065	MQRCCF_TERMINATED_BY_SEC_EXIT
4067	MQRCCF_DYNAMIC_Q_SCOPE_ERROR
4068	MQRCCF_CELL_DIR_NOT_AVAILABLE

Reason codes

Appendix E. Error messages

MQSeries for Windows can issue the error messages shown in this appendix. When you run an MQSC command file, the error messages also appear in the file MQSC.LOG.

In some messages, the text includes strings or numbers that vary depending on the circumstances that cause the message. In this appendix, these message variables are shown in the text of the message like *this*.

For explanations of the syntax of the MQSC commands, see the online *MQSeries for Windows Command Reference*.

AMQ3500 *primary-keyword secondary-keyword name* **was successful.**

Explanation: The operation was successful.

Action: None required.

AMQ3501 *character* **- string expected.**

Explanation: The first quotation mark of a quoted string was expected, but the character *character* was found.

Action: Enclose the string in quotation marks and retry the command.

AMQ3502 *integer* **- value out of range.**

Explanation: The specified integer value for an attribute is outside the allowed range.

Action: Change the integer to a value that is within the allowed range and retry the command.

AMQ3503 *primary-keyword secondary-keyword name* **failed. Return code =** ***code*****.**

Explanation: The MQSC command is syntactically correct, but the queue manager could not perform the command.

Action: Look up the return code in Appendix D, “Return codes” on page 207, correct the problem, then retry the command.

AMQ3504 *string* **- attribute keyword expected.**

Explanation: An attribute keyword was expected, but *string* was found.

Action: Ensure the attribute keyword is valid and spelled correctly, then retry the command.

AMQ3505 *string* **- left parenthesis expected.**

Explanation: A left parenthesis was expected, but *string* was found.

Action: Correct the command, adding a left parenthesis, then retry the command.

AMQ3506 *string* **- number expected.**

Explanation: An attribute requires an integer value, but *string* was found.

Action: Correct the attribute value, then retry the command.

AMQ3507 *string* **- right parenthesis expected.**

Explanation: A right parenthesis was expected, but *string* was found.

Action: Correct the command by adding a right parenthesis. If you have used a name that contains special characters, ensure that the name is enclosed in single quotation marks. Then retry the command.

AMQ3508 *string* **- string expected.**

Explanation: A quoted string value was expected, but *string* found.

Action: Correct the string value, then retry the command.

AMQ3509 *keyword* **not valid for this command.**

Explanation: The keyword or attribute is not valid for this command.

Action: Correct the keyword or attribute, then retry the command.

AMQ3510 ACTION keyword required.

Explanation: A RESOLVE CHANNEL command was issued without the ACTION keyword.

Action: Specify the ACTION keyword and an appropriate parameter, then retry the command.

AMQ3511 CHLTYPE must be specified immediately after the channel name.

Explanation: On the DEFINE CHANNEL and ALTER CHANNEL commands, you must specify the channel type immediately after the channel name. The validity of many of the following parameters are determined by the channel type.

Action: Specify the CHLTYPE immediately after the channel name, then retry the command.

AMQ3512 CONNAME keyword required.

Explanation: The channel being defined has a channel type of SDR or RQSTR. With these channel types, you must specify a connection name on the DEFINE CHANNEL command.

Action: Specify the CONNAME keyword, then retry the command.

AMQ3513 Channel already exists - specify REPLACE.

Explanation: The channel being defined already exists and the replace option has not been specified.

Action: If you want to replace the existing channel, specify the REPLACE keyword; otherwise choose a different channel name.

AMQ3514 Channel not found.

Explanation: The channel specified in the ALTER CHANNEL command does not exist.

Action: Correct the channel name, then retry the command.

AMQ3515 LIKE channel not found.

Explanation: The LIKE channel name specified on the DEFINE CHANNEL command does not exist.

Action: Correct the LIKE channel name, then retry the command.

AMQ3516 Primary keyword *string* not valid.

Explanation: A primary keyword was expected, but *string* was found.

Action: Correct the primary keyword, then retry the command.

AMQ3517 Error opening channel definition file.

Explanation: The operating system could not open the channel definition file.

Action: Try the command again. If the error persists, reinstall MQSeries for Windows, then try the command again. If this does not solve the problem, contact your MQSeries administrator.

AMQ3518 Error reading channel definition file.

Explanation: The operating system could not read the channel definition file.

Action: Try the command again. If the error persists, reinstall MQSeries for Windows, then try the command again. If this does not solve the problem, contact your MQSeries administrator.

AMQ3519 Secondary keyword *string* not valid.

Explanation: A secondary keyword was expected, but *string* was found.

Action: Correct the secondary keyword, then retry the command.

AMQ3520 TRPTYPE keyword required.

Explanation: When you define a channel of this type, you must specify the transport type.

Action: Specify the TRPTYPE keyword on the DEFINE CHANNEL command, then retry the command.

AMQ3521 Cannot read continuation line.

Explanation: The operating system cannot read the line following the plus (+) character.

Action: A plus character means that the command continues on the following line. Ensure the continuation character is required, correct the continuation line, then retry the command. If the error persists, reinstall MQSeries for Windows, then try the command again. If this does not solve the problem, contact your MQSeries administrator.

AMQ3522 Unexpected comma.

Explanation: A keyword was expected, but a comma (,) was found.

Action: Correct the keyword, then retry the command.

AMQ3523 Unexpected left parenthesis.

Explanation: A keyword was expected, but a left parenthesis was found.

Action: Correct the keyword, then retry the command.

AMQ3524 Unexpected number ***number*****.**

Explanation: A keyword was expected, but *number* was found.

Action: Correct the keyword, then retry the command.

AMQ3525 Unexpected right parenthesis.

Explanation: A keyword was expected, but a right parenthesis was found.

Action: Correct the keyword, then retry the command.

AMQ3526 Value type ***type*** **not supported.**

Explanation: There may be a fault in the MQSC command processor.

Action: Restart the MQ connection. If the error persists, reinstall MQSeries for Windows, then restart the MQ connection. If this does not solve the problem, contact your MQSeries administrator.

AMQ3527 Keyword ***keyword*** **is not valid for given channel type.**

Explanation: The keyword *keyword* is not valid for use with the channel type specified by the CHLTYPE keyword.

Action: Correct the keyword, then retry the command.

AMQ3528 Wrong CHLTYPE for LIKE channel.

Explanation: The channel type for the channel defined by the LIKE keyword is different from that specified by the CHLTYPE keyword. In the DEFINE CHANNEL command, the channel types must match.

Action: Either correct the LIKE keyword to specify a channel of the required type, or correct the CHLTYPE keyword to match the channel type of the LIKE channel. Then retry the command.

AMQ3529 Wrong CHLTYPE for given definition.

Explanation: The channel type of the named channel is different from the CHLTYPE specified. In the ALTER CHANNEL command, the channel types must match.

Action: Correct the CHLTYPE keyword to match the channel type of the named channel, then retry the command.

AMQ3530 XMITQ keyword required.

Explanation: The channel being defined has a channel type of SDR or SVR. With these channel types, you must specify a transmission queue name on the DEFINE CHANNEL command.

Action: Specify the XMITQ keyword, then retry the command.

AMQ3531 Character attribute buffer exceeded.

Explanation: A character attribute is greater than 2000 characters in length.

Action: Correct the attribute, then retry the command. If the error persists, reinstall MQSeries for Windows, then try the command again. If this does not solve the problem, contact your MQSeries administrator.

AMQ3532 File ended unexpectedly.

Explanation: The file ended unexpectedly while reading a continuation line.

Action: Ensure that the command correctly uses the continuation character, then retry the command.

AMQ3533 Character *character* not valid.

Explanation: While parsing the keywords and attributes, *character* was found. It is not valid in this context.

Action: Correct the character. If you have used a name that contains special characters, ensure that the name is enclosed in single quotation marks. Then retry the command.

AMQ3534 Keyword *keyword* not valid.

Explanation: You cannot use this keyword in this situation.

Action: Correct the syntax of the command, then retry it.

AMQ3535 LIKE channel name too long.

Explanation: The length of the channel name specified with the LIKE keyword is too long. Channel names can be a maximum of 20 characters.

Action: Correct the channel name, then retry the command.

AMQ3536 Terminator of number *number* not valid.

Explanation: The characters following the number *number* are not valid.

Action: Correct the number, then retry the command.

AMQ3537 Keyword beginning *keyword* too long.

Explanation: The length of the keyword *keyword* is more than 10 characters.

Action: Correct the keyword, then retry the command.

AMQ3538 Name type is not a string.

Explanation: There may be a fault in the MQSC command processor.

Action: Restart the MQ connection. If the error persists, reinstall MQSeries for Windows, then restart the MQ connection. If this does not solve the problem, contact your MQSeries administrator.

AMQ3539 Number beginning *number* is too long.

Explanation: The number beginning with *number* is longer than the maximum of 9 digits.

Action: Correct the number, then retry the command.

AMQ3540 Keyword DEADQ *(string)* must be blank.

Explanation: MQSeries for Windows does not support dead-letter queues. You can set this keyword to blanks only.

Action: Remove the DEADQ keyword, then retry the command.

AMQ3541 String *string* ended unexpectedly.

Explanation: The string *string* ended with a new-line character instead of a quotation mark.

Action: Correct the string, then retry the command.

AMQ3542 String beginning *string* is too long.

Explanation: The string *string* is too long for its associated keyword.

Action: Correct the string, then retry the command.

AMQ3543 Too many attributes, ignoring this one.

Explanation: You can specify a maximum of 256 attributes on one command; any more are ignored.

Action: Correct the number of attributes associated with the command, then retry it. If the error persists, reinstall MQSeries for Windows, then try the command again. If this does not solve the problem, contact your MQSeries administrator.

AMQ3544 Keyword MCANAME *(name)* must be blank.

Explanation: MQSeries for Windows does not support the MCANAME keyword. You can set it to blanks only.

Action: Remove the MCANAME keyword, then retry the command.

AMQ3545 Internal error number = *return code*

Explanation: There may be a fault in the MQSC command processor.

Action: Restart the MQ connection. If the error persists, reinstall MQSeries for Windows, then restart the MQ connection. If this does not solve the problem, contact your MQSeries administrator.

AMQ3546 Secondary keyword not specified.

Explanation: The command must have a secondary keyword, but none has been specified.

Action: Specify the secondary keyword, then retry the command.

AMQ3547 String length of string *string* is not valid.

Explanation: Each string associated with a keyword can be of a certain length only. The length of string *string* is not valid for its associated keyword.

Action: Correct the string, then retry the command.

AMQ3548 Line too long - MQSC file error

Explanation: The MQSC file did not end with the expected new-line character.

Action: Edit the file and ensure that the final command ends correctly. If necessary, add a blank line to the end of the file. Then run the MQSC command file again.

AMQ3549 MQSC file *filename* ran successfully.

Explanation: The MQSC command file ran successfully.

Action: None required.

AMQ3550 MQSC file *filename* did not run successfully. See MQSC.LOG for more details.

Explanation: Some of the commands in the MQSC command file *filename* contained errors.

Action: Correct the commands in the file, then run the file again.

AMQ3551 *string* contains characters not valid for MQSeries objects.

Explanation: The name *string* contains characters that are not valid for MQSeries objects. When you name MQSeries objects, you can use only the following characters:

- Uppercase A-Z
- Lowercase a-z
- Integers 0-9
- Period (.)
- Forward slash (/)
- Underscore (_)
- Percent sign(%)

Action: Replace the nonvalid characters in the string, then retry the command.

AMQ3552 Internal error in the DISPLAY command. Return code = *code*.

Explanation: The MQSC command is syntactically correct, but the queue manager could not perform the command.

Action: Look up the return code in Appendix D, “Return codes” on page 207 and try to correct the problem. Then stop and restart the MQ connection. If the error persists, contact your MQSeries administrator.

AMQ3553 Insufficient memory for the DISPLAY command.

Explanation: MQ cannot allocate sufficient memory to display all the information generated by the DISPLAY command.

Action: If you used an asterisk (*) to specify a generic name, retry the command using a more restrictive name so MQ has to display fewer objects. If you cannot do this, free some memory by stopping some applications you are no longer using, then retry the command.

AMQ3554 The TYPE value specified is not supported for this DISPLAY command.

Explanation: Either:

- You have specified a queue type on the DISPLAY CHANNEL command.
- You have specified a channel type on the DISPLAY QUEUE command.

Action: Retry the command using the correct syntax. For explanations of the syntax of the MQSC commands, see the online *MQSeries for Windows Command Reference*.

AMQ3555 MQSeries object *name* not found.

Explanation: Either:

- There are no objects matching the name you specified in the command
- If you specified an object type in the command, there are no objects of that type matching the name you specified

Action: If you expected MQ to return information, retry the command, ensuring you type the correct object name.

AMQ3601 MQSeries for Windows has failed to verify correctly. Please see the *MQSeries for Windows User's Guide* for more information. Reason: *reason*.

Explanation: The *reason* describes the task that the Verify program was attempting when it found an error.

Action: For information on the possible causes of this message, see "Running the Verify function" on page 20. If you cannot rectify the problem, uninstall the product, reinstall it, then retry the Verify program. If the error persists, contact your MQ administrator.

AMQ3700 Command server started for queue manager *name*.

Explanation: The operation was successful.

Action: None required.

AMQ3701 Command server stopped.

Explanation: The operation was successful.

Action: None required.

AMQ3702 Cannot stop command server. Return code = *code*.

Explanation: There is an internal error in the command server.

Action: Look up the reason code in Appendix D, "Return codes" on page 207 and try to correct the problem. Then restart the command server. If the error persists, stop the connection, then restart it and the command server. If this does not solve the problem, contact your MQSeries administrator.

AMQ3703 Cannot create the command queue. Return code = *code*.

Explanation: The command queue (SYSTEM.ADMIN.COMMAND.QUEUE) does not exist and MQ cannot create it.

Action: Look up the reason code in Appendix D, "Return codes" on page 207 and try to correct the problem. If the error persists, contact your MQSeries administrator.

AMQ3704 Cannot open the command queue. Return code = *code*.

Explanation: MQ cannot open the command queue (SYSTEM.ADMIN.COMMAND.QUEUE).

Action: Look up the reason code in Appendix D, "Return codes" on page 207 and try to correct the problem. If the error persists, contact your MQSeries administrator.

AMQ3705 Cannot allocate memory to service the command queue.

Explanation: The command server must allocate sufficient memory to service the longest message on the command queue. This memory is not available.

Action: Free some memory by stopping any applications you no longer need to run, then restart the command server.

AMQ3706 Cannot query the attributes of the command queue. Return code = ***code*****.**

Explanation: The command server uses the MQINQ call to find the length of the longest message on the command queue. This call has failed.

Action: Look up the reason code in Appendix D, “Return codes” on page 207 and try to correct the problem. If the error persists, contact your MQSeries administrator.

AMQ3707 Internal error in the command server. Return code = ***code*****.**

Explanation: There is an internal error in the command server.

Action: Look up the reason code in Appendix D, “Return codes” on page 207 and try to correct the problem. Then restart the command server. If the error persists, stop the connection, then restart it and the command server. If this does not solve the problem, contact your MQSeries administrator.

AMQ3708 The reply-to queue does not exist.

Explanation: The reply-to queue specified in the PCF command does not exist. The command server cannot process the command.

Action: Before you retry the command, ensure that all the queues required by the PCF application exist on the queue manager.

AMQ3709 Cannot open or put a message on the reply-to queue. Return code = ***code*****.**

Explanation: MQ cannot open the reply-to queue or it cannot put a message on the queue.

Action: Look up the reason code in Appendix D, “Return codes” on page 207 and try to correct the problem. If the error persists, contact your MQSeries administrator.

AMQ3710 Cannot close the reply-to queue. Return code = ***code*****.**

Explanation: MQ cannot close the reply-to queue.

Action: Look up the reason code in Appendix D, “Return codes” on page 207 and try to correct the problem. If the error persists, contact your MQSeries administrator.

AMQ3711 Internal error in the command server. Return code = ***code*****.**

Explanation: There is an internal error in the command server.

Action: Look up the reason code in Appendix D, “Return codes” on page 207 and try to correct the problem. Then restart the command server. If the error persists, stop the connection, then restart it and the command server. If this does not solve the problem, contact your MQSeries administrator.

AMQ3712 Cannot get a message from the administration command queue. Return code = *code*.

Explanation: MQ cannot get a message from the administration command queue (SYSTEM.ADMIN.COMMAND.QUEUE). The command server has stopped.

Action: Look up the reason code in Appendix D, "Return codes" on page 207 and try to correct the problem. Then restart the command server. If the error persists, contact your MQSeries administrator.

AMQ3713 Internal error in the command server. Return code = *code*.

Explanation: There is an internal error in the command server.

Action: Look up the reason code in Appendix D, "Return codes" on page 207 and try to correct the problem. Then restart the command server. If the error persists, stop the connection, then restart it and the command server. If this does not solve the problem, contact your MQSeries administrator.

AMQ3714 Cannot get messages from the administration command queue because it is inhibited.

Explanation: The InhibitGet attribute of the administration command queue (SYSTEM.ADMIN.COMMAND.QUEUE) is set so that MQGET calls are not allowed. This means the command server cannot get messages from the queue.

Action: If you do not want to prevent MQGET calls, change the InhibitGet attribute of the queue to allow MQGET calls. To do this on the Complete version of the product, use the Components page of the MQSeries Properties dialog box. To do this on the Compact version of the product, use an MQSC command.

AMQ3715 The administration command queue is no longer inhibited.

Explanation: The InhibitGet attribute of the administration command queue (SYSTEM.ADMIN.COMMAND.QUEUE) has been changed so that MQGET calls are now allowed. This means the command server can get messages from the queue.

Action: None required.

AMQ3716 Cannot stop command server. Return code = *code*.

Explanation: MQ cannot stop the command server. If you were trying to stop a connection that uses the command server, that connection has stopped, but the command server is still running.

Action: Use the Command Server page of the MQSeries Properties dialog box to start the command server, then stop it again.

If the error persists, look up the reason code in Appendix D, "Return codes" on page 207 and try to correct the problem. If you cannot correct the problem, contact your MQSeries administrator.

AMQ3801 Connection broken; there is a problem with the queue manager.

Explanation: The connection is stopped because there is a problem with its queue manager.

Action: For more information, use the MQSeries Properties dialog to view the status of the queue manager. Then restart the connection.

AMQ3802 Connection broken; there is a problem with the channel group.

Explanation: The connection is stopped because there is a problem with its channel group.

Action: For more information, use the MQSeries Properties dialog to view the status of the channel group. Then restart the connection.

AMQ3803 Connection broken; there is a problem with the phonebook entry.

Explanation: The connection is stopped because there is a problem with its phonebook entry.

Action: For more information, use the MQSeries Properties dialog to view the status of the phonebook entry. Then restart the connection.

AMQ3804 Channel group broken; there is a problem with the listener.

Explanation: The connection is stopped because there is a problem with the listener in its channel group.

Action: For more information, use the MQSeries Properties dialog to view the status of the listener. Then restart the connection.

AMQ3805 Channel group broken; there is a problem with one or more channels.

Explanation: The connection is stopped because there is a problem with one or more of the channels in its channel group.

Action: For more information, use the MQSeries Properties dialog to view the status of the channels. Then restart the connection.

Appendix F. Notices

Trademarks

Trademarks

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

AIX	MQ	OS/2
BookManager	MQSeries	OS/400
CICS	MQSeries Three Tier	VSE/ESA
FFST	MVS	WIN-OS2
IBM	MVS/ESA	

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, and the Windows 95 Logo are trademarks or 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 5. Glossary and index

Glossary of terms and abbreviations

This glossary describes terms used in this book and words used with other than their everyday meaning. In some cases, a definition may not be the only one applicable to a term, but it gives the particular sense in which the word is 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.

A

alias queue. An MQSeries object that enables MQI applications to specify aliases for queue names. At run time, the alias is resolved and the requested operation is performed on the queue with the resolved name.

APAR. Authorized program analysis report.

application queue. A local queue used by an application, as opposed to special-purpose queues (for example, transmission queues).

attribute. One of a set of properties that defines the characteristics of an MQSeries queue manager, queue, or channel.

authorized program analysis report (APAR). A report of a problem caused by a suspected defect in a current, unaltered, release of a product.

B

back out. To reverse all the changes made during the current unit of recovery or unit of work.

browse. In message queuing, to copy a message without removing it from the queue. See also *get*.

browse cursor. An identifier that specifies the next message on a queue to be browsed when an application issues a ‘get with browse’ call.

C

call back. In MQSeries, a requester message channel initiates a transfer from a sender channel by first calling the sender, then closing down and awaiting a call back.

caller. The message channel agent (MCA) that starts the channel. It does this by sending a connection request message which the channel listener is listening for.

CCSID. Coded character-set identifier.

channel. See *message channel*.

channel event. An event that indicates a channel instance has become available or unavailable. Channel events are generated on the queue managers at both ends of the channel. On MQSeries for Windows, channel event messages are generated only if you create a channel event queue to hold them.

channel group. In MQSeries for Windows, a named collection of channels, owned by one queue manager, that MQ starts and stops as a group.

channel initiator. In MQSeries, a program that starts one end of a message channel. The other end of the channel must be listening for incoming connection requests. MQSeries for Windows does not use the MQSeries channel initiator program.

channel listener. A program that monitors connection requests from queue managers on other workstations.

COA. Confirm on arrival. See *report message*.

COD. Confirm on delivery. See *report message*.

coded character-set identifier (CCSID). The name of a coded set of characters and their code-point assignments.

command processor. The part of the queue manager that processes commands.

command server. The part of MQSeries that reads commands from the administration command queue, verifies them, and passes on the valid ones for processing by the command processor.

Glossary

commercial messaging. A messaging strategy that allows distributed applications, particularly commercial applications, to communicate using messages. MQSeries for Windows is an example of a commercial messaging product.

commit. See *single-phase commit* and *two-phase commit*.

completion code. A return code indicating whether an MQI call was successful. If the call failed, or partially succeeded, a reason code provides more information about the cause. See also *reason code*.

component. In MQSeries for Windows, a component is an MQ connection, a queue manager, or a channel group. See also *object*.

connection. In MQSeries for Windows, a connection contains all the MQ objects an application user needs to run an MQ application. It comprises a queue manager and the objects MQ needs to communicate from that queue manager to another. These objects might be a channel group and a phonebook entry. A typical user chooses from a list of connections when MQ starts.

connection handle. The identifier, or token, by which a program accesses the queue manager to which it is connected.

context. In MQSeries, context information is included in the message header to show the origin of the message. MQSeries for Windows does not copy context information from messages it receives from other queue managers.

D

dead-letter queue. A queue to which a queue manager or application sends messages it cannot deliver to their correct destination. MQSeries for Windows does not allow dead-letter queues.

default object. A definition of an object (for example, a queue) with all its attributes defined. If you define an object, but do not specify all the possible attributes that object could have, the queue manager uses these default attributes for the missing ones.

definition file. See *MQD file*.

dial-up connection. An MQ connection that comprises a queue manager, a channel group, and a phonebook entry for communicating with another queue manager. Application users use a dial-up connection when they want to transmit data to a computer that is connected to theirs through a dial-up telephone link.

dial-up networking connection. See *phonebook entry*.

distributed queue management. In message queuing, the setup and control of message channels to queue managers on other systems.

dynamic queue. A local queue that is created when a program opens a model queue object.

E

event. See *channel event*, *performance event*, and *queue manager event*.

event message. A message that contains information (such as the category of the event, the name of the application that caused the event, and queue manager statistics) relating to the origin of an event in a network of MQSeries systems.

event queue. The queue on to which the queue manager puts an event message after it detects an event. Each category of event (channel, performance, or queue manager) has its own event queue.

F

FFST. First Failure Support Technology. A program used by MQSeries to indicate possible software problems.

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.

G

get. In message queuing, to retrieve a message by removing the message from a queue or by browsing the message. See also *browse*.

H

handle. The identifier, or token, by which a program accesses an MQSeries object. See *connection handle* and *object handle*.

I

initiator. See *channel initiator*.

input parameter. A parameter of an MQI call in which you supply information when you make the call.

input/output parameter. A parameter of an MQI call in which you supply information when you make the call, and in which the queue manager changes the information when the call completes or fails.

L

LAN connection. An MQ connection that comprises a queue manager and a channel group for communicating with another queue manager. Application users use a LAN connection when they want to transmit data to a computer that is connected to theirs through a local area network.

leaf node. In a network of nodes, a leaf node is connected to only one other node, so it is on the outer edge of the network. A leaf-node queue manager is intended for use by a single user, and not as an intermediate queue manager that passes messages between other queue managers and serves many users. MQSeries for Windows queue managers are leaf-node queue managers.

listener. See *channel listener*.

local definition. An MQSeries object that belongs to a local queue manager.

local definition of a remote queue. An MQSeries object that belongs to the 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. To a program, the queue manager to which the program is connected. This is the queue manager that provides message queuing services to that 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.

logical unit of work (LUW). See *unit of work*.

M

MCA. Message channel agent.

message. In message queuing applications, a communication sent from a program to another program.

message channel. A named unidirectional network transport mechanism along which MQSeries messages are sent between two queue managers.

message channel agent (MCA). In MQSeries, a program that either transmits prepared messages from a transmission queue to a network, or takes messages from the network and puts them on a destination queue.

message descriptor. Control information 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 are retrieved from a queue.

message queue. Synonym for queue.

Message Queue Interface (MQI). The application programming interface provided by the MQSeries queue managers. This 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.

message sequence numbering. A programming technique in which messages are given unique numbers during transmission over a communication link. This enables the receiving process to check whether all messages are received, to place them on a queue in the original order, and to discard duplicate messages.

messaging. A method for communication between programs. Messaging can be synchronous or independent of time.

model queue. An MQSeries object that contains a set of queue attributes that act as a template when a program creates a dynamic queue.

MQD file. In MQSeries for Windows, a file that defines the properties of an MQ component. MQ runs the MQD file at the end of the installation process. If the file has changed, MQ also runs the file each time it starts.

MQI. Message queue interface.

MQI channel. A special channel that connects an MQSeries client to an MQSeries server (queue manager) and transfers only MQI calls and responses. MQSeries for Windows does not support MQI channels.

MQSC commands. Human readable commands in a specific format that change the attributes of MQSeries objects.

Contrast with *programmable command format*.

MQSeries client. A runtime component of MQSeries for OS/2, AIX, and UNIX systems. MQSeries for Windows does not support MQSeries clients.

N

name transformation. In MQSeries, a process that creates file names for MQSeries objects so that they are unique and valid for the system being used.

O

object. In MQSeries, an object is a queue manager, a queue, or a channel. See also *component*.

object descriptor. A data structure that identifies a particular MQSeries object. It includes the object name and its type.

object handle. The identifier, or token, by which a program accesses the MQSeries object with which it is working.

output parameter. A parameter of an MQI call in which the queue manager returns information when the call completes or fails.

P

PCF. See *programmable command format*.

performance event. An event that indicates a limit condition has occurred.

permanent queue. A queue that is not erased when the queue manager stops. Contrast with *temporary queue*.

persistent message. A message that survives a restart of the queue manager.

phonebook entry. Known in the Windows operating system as a dial-up networking connection. It relates a name to a telephone number.

You must install the dial-up networking component of Windows before you can define a phonebook entry.

ping. In distributed queue management, a diagnostic aid that uses the exchange of a test message to confirm that a message channel is functioning.

platform. In MQSeries, the operating system on which a queue manager is running.

programmable command format. A type of MQSeries message used by:

- User administration applications that put PCF commands on the administration command 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 commands*.

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. Queues can be of type local, alias, model, or remote. Local queues can contain a list of messages waiting to be processed. Queues of other types cannot contain messages; they point to other queues.

queue manager. A program that provides messaging services to applications. It provides an application programming interface so that programs can access messages on the queues that the queue manager owns.

queue manager event. An event that indicates:

- An error condition has occurred in relation to the resources used by the queue manager. For example, an error condition caused by a queue being unavailable.
- A significant change has occurred in the queue manager. For example, a queue manager has started or stopped.

queuing. See *message queuing*.

R

reason code. A return code that describes the reason for the failure or partial success of an MQI call.

receiver channel. In message queuing, a channel that responds to a sender channel, takes messages from a communication link, and puts them on the specified local queue.

remote queue. A queue that belongs to a remote queue manager. Programs can put messages on remote queues, but they cannot get messages from remote queues. Contrast with *local queue*.

remote queue manager. To an MQI application, a queue manager is remote if it is not the queue manager to which the program is connected.

remote queue object. See *local definition of a remote queue*.

remote queuing. In message queuing, the provision of services to enable applications to put messages on queues that belong to other queue managers.

reply message. A type of message used for replies to request messages.

reply-to queue. The name of a queue to which the program that issued an MQPUT call wants a reply message or report message sent.

report message. A message that provides information about the delivery (or nondelivery) of an MQSeries message that was put on a queue by an application issuing an MQPUT call. A report message can indicate the original message:

- Has arrived on the target queue; this is a Confirm on Arrival (COA) report.
- Was retrieved by an application and deleted from the queue; this is a Confirm on Delivery (COD) report.
- Could not be delivered because, for example, a channel is not available; this is an Exception report.
- Has been deleted from the queue because its expiry date has elapsed; this is an Expiry report.

requester channel. In MQSeries, a channel that may be started remotely by a sender channel. The requester channel accepts messages from the sender channel over a communication link and puts the messages on the local queue designated in the message.

request message. A type of message used for requesting a reply from another program.

responder. The message channel agent (MCA) that the channel listener program starts when it receives a connection request message from a caller MCA.

return codes. The collective name for completion codes and reason codes.

rollback. Synonym for back out.

S

sender channel. In MQSeries, a channel that initiates transfers of messages, removes messages from a transmission queue, and moves them over a communication link to a receiver or requester channel.

sequential delivery. In MQSeries, a method of transmitting messages with a sequence number so that the receiving channel can reestablish the message sequence when storing the messages. This is required when messages must be delivered only once, and in the correct order.

sequential number wrap value. In MQSeries, a method of ensuring that both ends of a communication link reset their current message sequence numbers at the same time. Transmitting messages with a sequence number ensures that the receiving channel can reestablish the message sequence when storing the messages.

Glossary

server. The program that responds to requests for information in the particular two-program information-flow model of client/server.

server channel. In MQSeries, a channel that responds to a requester channel, removes messages from a transmission queue, and moves them over the network to the requester channel.

Service Trace. In MQSeries for Windows, a utility that traces the operation of a queue manager. Use it to help you debug an MQSeries application.

signaling. A feature that allows the operating system to notify a program when an expected message arrives on a queue. Signaling is available only on MQSeries for MVS/ESA and MQSeries for Windows Version 2.1.

single-phase backout. A method in which an action that is in progress must not be allowed to finish, and all changes that are part of that action must be undone.

single-phase commit. A method in which a program can commit updates to a queue without coordinating those updates with updates the program has made to resources controlled by another resource manager.

standalone connection. An MQ connection that comprises no components for communicating with other queue managers. Application users use a standalone connection when they want to work on a computer that does not have an external communication facility such as a LAN cable or a modem.

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.

T

temporary queue. A queue that is deleted when the queue manager is stopped. It can contain only nonpersistent messages.

thread. In MQSeries, the lowest level of parallel execution available on an operating system.

time-independent messaging. A method for communication between programs in which the requesting program proceeds with its own processing without waiting for a reply to its request.

trace. A facility for recording MQSeries activity.

transmission program. See *message channel agent*.

transmission queue. A local queue on which prepared messages destined for a remote queue manager are stored temporarily.

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. MQSeries for Windows does not support two-phase commit.

U

unit of recovery. A recoverable sequence of operations within a single resource manager. Compare 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 at a user-requested syncpoint. It ends either at a user-requested syncpoint or at the end of a transaction. Compare with *unit of recovery*.

Index

Index

Index

Index

Index

Index

Index

Index

Sending your comments to IBM

MQSeries for Windows**

User’s Guide

GC33-1965-00

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 (RCF)
- 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/address/telephone number/fax number/network ID.

Readers' Comments

MQSeries for Windows**

User's Guide

GC33-1965-00

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

MQSeries for Windows**
MQSeries for Windows User's Guide GC33-1965-00**

IBM

You can send your comments POST FREE on this form from any one of these countries:

Australia	Finland	Iceland	Netherlands	Singapore	United States of America
Belgium	France	Israel	New Zealand	Spain	
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.).

1 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

IBM

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 ____________

1 Cut along this line

4 Fasten here with adhesive tape

Program Number: 5639-B69

Printed in U.S.A.

GC33-1965-00

Spine information:

MQSeries for Windows **User’s Guide** Version 2.1 GC33-1965-00