MQSeries for Windows**

User's Guide

Version 2.0

GC33-1822-00

MQSeries for Windows**

User's Guide

Version 2.0

GC33-1822-00

Note

Before using this information and the product it supports, be sure to read the general information in Appendix G, “Notices” on page 187.

First Edition (September 1996)

This edition applies to IBM MQSeries for Windows, Version 2.0, part number 83H8535, program number 5622-960, 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

Contents

Figures and tables

Figures

Tables

Figures and tables

About this book

IBM MQSeries for Windows** Version 2.0 (known in this book as MQSeries for Windows) provides messaging and queuing services on Microsoft Windows.

This book tells you how to install, set up, administer, and operate MQSeries for Windows. However, this book is just to get you started, 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 does not contain descriptions of the commands you can use when you are running or administering MQSeries for Windows because descriptions of those commands are available in the online *MQSeries for Windows Command Reference* that you can install as part of the product.

This book introduces you to the utilities provided with MQSeries for Windows. For detailed information on how to use them, see the online help.

Who this book is for

This book is for:

- Users of MQSeries applications who want to install and configure MQSeries for Windows.
- System 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 to run on Windows. This book explains those features of the IBM Message Queue Interface (MQI) that are not supported on Windows.
- Experienced MQSeries application programmers who want to write a new application to run under MQSeries for Windows.

What you need to know to understand this book

The knowledge you need 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 users.

About this book

- 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, but for more detailed information you must read the *MQSeries Application Programming Guide* and the *MQSeries Application Programming Reference*.

How to use this book

This book is organized in a task-oriented way, so you should use the part of the book appropriate to the task you are performing. The installation and set-up tasks are described before the administration and programming tasks. The parts of the book are:

Part 1, Introduction
This part introduces MQSeries for Windows and the concepts of messaging and queuing. It tells you what you need to know and what you need to do before you start using MQSeries for Windows.

Part 2, Installing MQSeries for Windows
This part tells you how to install MQSeries for Windows on your workstation, and explains how to install MQSeries for Windows on another workstation remotely (that is, by issuing commands on your own workstation). It also tells you how to verify that the system you have created is working correctly.

Part 3, Setting up queue managers
This part tells you how to create MQSeries for Windows components and how to make one queue manager communicate with another.

Part 4, Supporting users of MQSeries for Windows
This part tells you how to support other users who are running applications on MQSeries for Windows. It tells you how to monitor and change the queue managers they are running, and how to solve problems they may have.

Part 5, Application programming
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 6, 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 7, 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"
qmgrname	In the syntax of a command, this is a placeholder for information you must provide
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 Version 3.1 or later, including Win-OS/2.

MQSeries publications

MQSeries for Windows publications

The following information is available for MQSeries for Windows:

- *MQSeries for Windows Version 2.0 User's Guide*, GC33-1822 (available as this printed book and as an online book)
- *MQSeries for Windows Version 2.0 Command Reference* (available as an online book only)
- An online Quick Tour that explains the MQSeries for Windows program group
- Online help for the utilities provided by MQSeries for Windows

To open the online books, click on their icons in the MQSeries for Windows program group.

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

- Read the READ.ME file by clicking on its icon in the MQSeries for Windows program group
- See the IBM MQSeries site on the Internet (see "Information about MQSeries on the Internet" on page xv)

MQSeries publications

Evaluating products

MQSeries Brochure, G511-1908

MQSeries: An Introduction to Messaging and Queuing, GC33-0805

MQSeries Message Queue Interface Technical Reference, SC33-0850

Planning

MQSeries Planning Guide, GC33-1349

MQSeries for MVS/ESA Version 1 Release 1.4 Licensed Program Specifications, GC33-1350

MQSeries for OS/400 Version 3 Release 1 (and later) Licensed Program Specifications, GC33-1360 (softcopy only)

MQSeries publications

Administration

MQSeries Clients, GC33-1632

MQSeries Command Reference, SC33-1369

MQSeries Programmable System Management, SC33-1482

MQSeries for AIX Version 2.2.1 System Management Guide, SC33-1373

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

MQSeries for HP-UX Version 2.2.1 System Management Guide, GC33-1633

MQSeries for OS/2 Version 2.0.1 System Management Guide, SC33-1371

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

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

MQSeries for Sun Solaris Version 2.2 System Management Guide, GC33-1800

MQSeries for Windows NT Version 2.0 System Management Guide, SC33-1643

MQSeries for MVS/ESA Version 1 Release 1.4 Program Directory

MQSeries for MVS/ESA Version 1 Release 1.4 System Management Guide, SC33-0806

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

MQSeries for OS/400 Version 3 Release 6 Administration Guide, SC33-1361

MQSeries for OS/400 Version 3 Release 6 Programmable Command Formats, SC33-1228

MQSeries Three Tier Administration Guide, SC33-1451

MQSeries Three Tier Reference Summary, SX33-6098

Application programming

MQSeries Application Programming Guide, SC33-0807

MQSeries Application Programming Reference, SC33-1673

MQSeries Application Programming Reference Summary, SX33-6095

MQSeries for OS/400 Version 3 Release 1 (and later) Application Programming Reference (RPG), SC33-1362

MQSeries for OS/400 Version 3 Release 6 Application Programming Reference (C and COBOL), SC33-1363

MQSeries Three Tier Application Design, SC33-1636

MQSeries Three Tier Application Programming, SC33-1452

MQSeries Three Tier Reference Summary, SX33-6098

MQSeries publications

Problem determination

MQSeries for AIX Version 2.2.1 System Management Guide, SC33-1373

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

MQSeries for HP-UX Version 2.2.1 System Management Guide, GC33-1633

MQSeries for OS/2 Version 2.0.1 System Management Guide, SC33-1371

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

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

MQSeries for Sun Solaris Version 2.2 System Management Guide, GC33-1800

MQSeries for Windows NT Version 2.0 System Management Guide, SC33-1643

MQSeries for MVS/ESA Version 1 Release 1.4 Messages and Codes, GC33-0819

MQSeries for MVS/ESA Version 1 Release 1.4 Problem Determination Guide, GC33-0808

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

MQSeries for OS/400 Version 3 Release 6 Administration Guide, SC33-1361

MQSeries Three Tier Administration Guide, SC33-1451

Special topics

MQSeries Distributed Queuing Guide, SC33-1139

Softcopy books

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

Other softcopy formats are available, depending on the platform being used.

Other MQSeries Version 1 publications

For information about other MQSeries platforms, 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 User's Guide, SC33-1144

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

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

MQSeries for UnixWare User's Guide, SC33-1379

MQSeries for VSE/ESA Version 1 Release 3.1 Licensed Program Specifications, GC33-1483

MQSeries for VSE/ESA Version 1 Release 3.1 User's Guide, SC33-1142

Information about MQSeries on the Internet

The MQSeries home page

The URL of the MQSeries product family home page is:

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

MQSeries publications

Part 1. Introduction

Chapter 1. Introduction to MQSeries for Windows

MQSeries for Windows is a lightweight messaging product that provides MQSeries functions on workstations that run Microsoft Windows** 3.1 or Windows 95. It uses significantly fewer resources than other MQSeries products, so it is a good choice to use as a single-user queue manager running on a small or medium-sized personal computer.

MQSeries for Windows is particularly well suited to users of messaging applications who want to use a standard configuration. MQSeries for Windows provides a Create and Go utility that automatically creates and starts the messaging components the users need, and it can automatically start components when the users start their workstations. These features reduce the need for users of applications to be aware of the messaging product and allow them to concentrate on the applications they want to use.

This chapter contains the following sections:

"Where to use MQSeries for Windows"
This explains the intended use of MQSeries for Windows.

"The features of MQSeries for Windows" on page 5
This introduces the features provided by MQSeries for Windows.

"Where to find the information you need" on page 8
This explains where to find the information you need to use MQSeries for Windows.

"Introduction to messaging" on page 9
For new users of MQSeries, this section provides an introduction to the concepts of messaging and queuing.

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 an MQSeries network of computers (see Figure 1 on page 4).

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.
- 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.

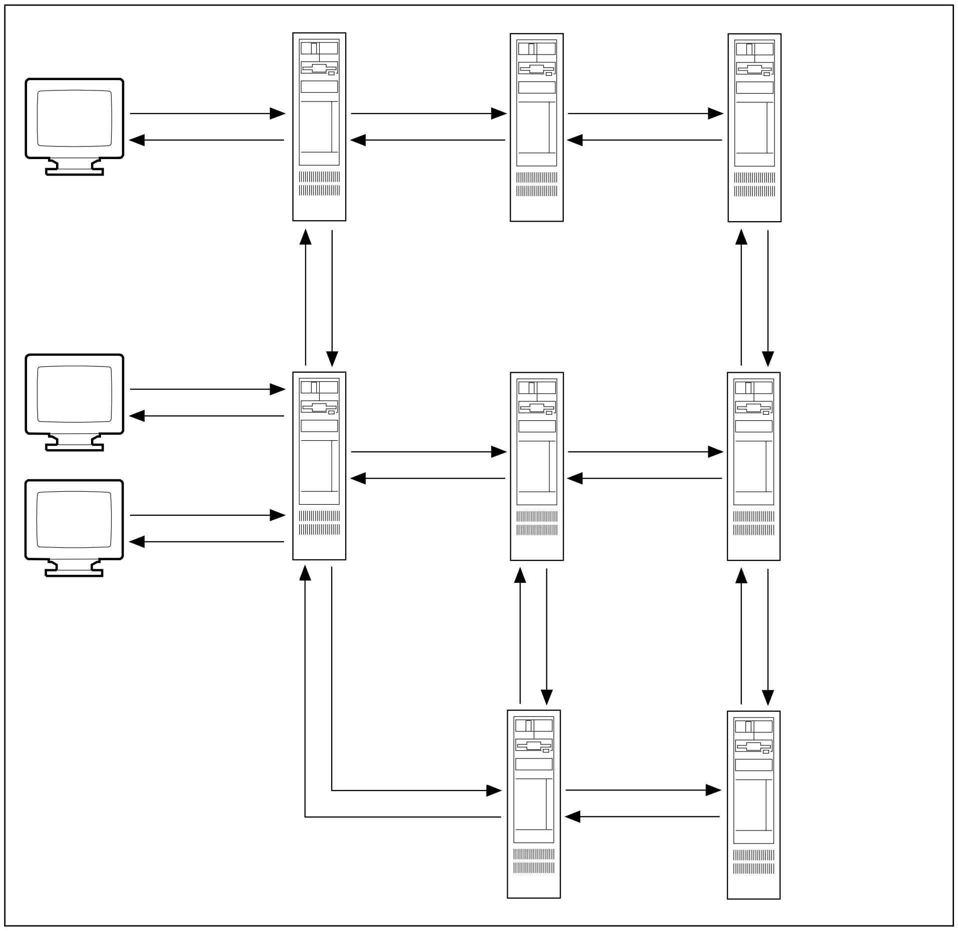

Figure 1. A network of server queue managers and three leaf nodes. The leaf nodes run on Windows; the server nodes run on any other MQSeries platform.

- A server-node queue manager is a product 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.

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, “How MQSeries for Windows differs from the MQSeries family” on page 163.

Introduction

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 utility programs are not provided by other MQSeries products.

The features of MQSeries for Windows

MQSeries for Windows provides existing MQSeries features, but on the Windows operating system:

- A queue manager that runs on Windows
- The MQSeries Message Queue Interface (MQI) for application development on Windows
- Application development support for the C and Visual Basic programming languages
- Communication between queue managers using TCP/IP
- Standard MQSeries message types and formats
- *Persistent messages* (which survive restarts of the workstation) and non-persistent messages
- Standard MQSC commands to create, alter, or delete MQSeries objects (but MQSeries for Windows does not support all the commands)
- Enablement for automatic installation using Configuration, Installation, and Distribution (CID)
- Report generation, including confirm on arrival (COA), confirm on delivery (COD), and message expiry

In addition, MQSeries for Windows provides these features:

- To help users of applications to get started quickly and easily the first time they use the product, MQSeries for Windows provides a Create and Go utility that automatically creates and starts the messaging components the users need.
- To help users of applications to get started when they start their workstations, MQSeries for Windows can automatically start its components.
- To help you to set up and work with your queue managers, it provides utilities you can access from the MQSeries for Windows program group in the Windows Program Manager. They are Windows programs and have extensive online help. For an introduction to these utilities, see “The MQSeries for Windows utilities” on page 6.
- To make it easier to work with the message channels that you must use to send messages between queue managers, MQSeries for Windows provides channel groups. A channel group is a collection of channels that you start and stop at the same time.
- To make it easier to work with dial-up devices (such as modems) when you connect two queue managers, it provides transport links, which can help you to control the duration (and hence the cost) of such a connection.

Introduction

The MQSeries for Windows utilities

This section introduces the utilities, and shows the icons you must select in the MQSeries for Windows program group to start them.

Verify Install utility

Use this utility to verify that you have installed MQSeries for Windows correctly.

For more information, see “Verifying your installation” on page 35.

Create and Go utility

Use this utility to create and start MQSeries components automatically, using definitions from an initialization file.

For more information, see “Creating components using the Create and Go utility” on page 44.

Create Components utility

Use this utility to create any of the following components:

- A queue manager
- A queue
- A channel
- A channel group
- A transport link

For information on how to use this utility, see “Creating a queue manager” on page 44.

Standard Controls utility

Use this utility to start a queue manager, a channel group, or a transport link, and to monitor their status.

This utility is provided for end users who may, for example, be required to start a queue manager and report diagnostic information about the status of their local MQSeries installation to a central administrator.

For more information, see “Monitoring queue managers and queues using the Standard Controls utility” on page 46.

Delete Components utility

Use this utility to delete any of the following components:

- A queue manager
- A queue
- A channel
- A channel group
- A transport link

For information on how to use this utility, see "Deleting queue managers and queues using the Delete Components utility" on page 51.

Advanced Controls utility

Use this utility to start a queue manager, a channel group, or a transport link, and to monitor their status.

This utility is intended for MQSeries administrators who support other users. Like the Standard Controls utility, it allows them to view the status of their MQSeries installation, but it also allows them to change it.

For more information, see "The Advanced Controls utility" on page 82.

MQSC Commands utility

Use this utility to issue MQSC commands either individually or from a command file. When you issue commands individually, you can recall and edit them as well.

For more information, see Chapter 10, "Using MQSC commands" on page 85.

Service Information utility

Use this utility to display service information about an MQSeries for Windows installation.

For more information, see "The Service Information utility" on page 119.

Service Trace utility

Use this utility to trace the operation of an MQSeries for Windows queue manager.

For more information, see "The Service Trace utility" on page 119.

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 want to run an MQSeries application
To run an MQSeries application without having to set up MQSeries for Windows, see:

- “Creating components using the Create and Go utility” on page 44

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

- Part 2, “Installing MQSeries for Windows” on page 17

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

- Part 1, “Introduction” on page 1
- Part 2, “Installing MQSeries for Windows” on page 17
- Part 3, “Setting up queue managers” on page 41
- The online Quick Tour

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

- Part 1, “Introduction” on page 1
- Part 3, “Setting up queue managers” on page 41
- Part 4, “Supporting users of MQSeries for Windows” on page 79
- The differences described in Part 6, “Appendixes” on page 161
- The online Quick Tour
- The online *MQSeries for Windows Command Reference*

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 1, “Introduction” on page 1
- Part 3, “Setting up queue managers” on page 41
- Part 4, “Supporting users of MQSeries for Windows” on page 79
- Part 6, “Appendixes” on page 161

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 5, “Application programming” on page 129
- The differences described in Part 6, “Appendixes” on page 161

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 5, “Application programming” on page 129
- The differences described in Part 6, “Appendixes” on page 161

Introduction to messaging

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 computers, 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 some utilities to help you do this.

Messaging

How applications identify themselves to queue managers

An application that issues MQI calls is referred to as an MQSeries application. When this book refers to an application, it means an MQSeries application.

Any MQSeries application must make a successful connection to a queue manager before it can make any other MQI calls. When the application successfully makes the connection, 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*), so only one connection handle is valid (for that particular application) at a time. 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.

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 on 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 2 on page 11 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 computer and connected to the same queue manager.

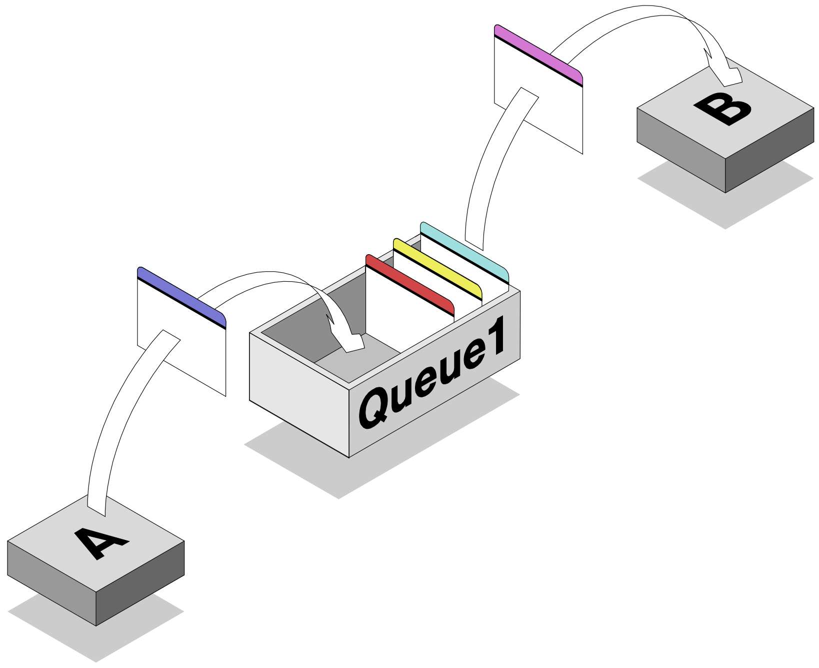

Figure 2. Programs connected to the same queue manager. Program A puts messages on the queue; program B gets messages from the queue. In this case, the programs and the queue manager are running on the same workstation.

Messaging using more than one queue manager

The arrangement shown in Figure 2 is not typical for a real messaging application because both programs are running on the same computer, and connected to the same queue manager. In a commercial application, the putting and getting programs would probably be on different computers, 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 Chapter 7, “Using more than one queue manager” on page 53.

Messaging

Chapter 2. Planning for MQSeries for Windows

This chapter describes:

- The hardware and software you need to run MQSeries for Windows
- The differences between an MQSeries for Windows queue manager, an MQSeries client, and an MQSeries server

Suggested hardware

MQSeries for Windows is a 16-bit product, so it runs on computers that run Windows 3.1, and it runs in 16-bit compatibility mode on Windows 95. Table 1 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 system at the same time.

Table 1. Suggested hardware configurations for MQSeries for Windows

Configuration	**Processor**	**RAM**	**Hard disk**
For running applications	386 16 MHz	4–8 MB	3.9 MB available
For developing applications	486 66 MHz or better.	8–16 MB	5 MB available
Note: The specification for developing applications does not include hardware requirements for other development tools (for example, compilers).			

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):

- MS-DOS or PC DOS Version 3.3
- Microsoft Windows 3.1,
  or Windows 95,
  or Windows for Workgroups 3.11,
  or Win-OS/2 on OS/2 Version 3.0 (Warp)

  MQSeries for Windows runs in 16-bit compatibility mode on Windows 95.
- TCP/IP for the operating system you are using:
  - For Microsoft Windows 3.1, you need IBM TCP/IP for DOS V2.1.1 with CSD 2.1.1.4.

- For Microsoft Windows 95, use the version of TCP/IP supplied with Windows 95.
- For Microsoft Windows for Workgroups 3.11, you need IBM TCP/IP for DOS V2.1.1 with CSD 2.1.1.4.
- For Win-OS/2, you need IBM TCP/IP for OS/2 Version 2.

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" on page 13***, you need only the compiler for the programming language you will use:

For 16-bit C
Microsoft Visual C++ Version 1.5

For 32-bit C
Microsoft Visual C++ Version 2.0

For 16-bit BASIC
Microsoft Visual Basic Version 3.0
or Microsoft Visual Basic Version 4.0

For 32-bit BASIC
Microsoft Visual Basic Version 4.0

MQSeries for Windows runs in 16-bit compatibility mode on Windows 95, but you can write 32-bit MQSeries for Windows applications.

Comparing queue managers, clients, and servers

If you already use MQSeries clients, see Table 2 for a summary of the differences between an MQSeries for Windows queue manager, an MQSeries client, and an MQSeries server.

Table 2. Comparison of supported features

Feature	MQSeries for Windows	MQSeries client on Windows	MQSeries for OS/2
Independent operation	Yes	No	Yes
Queue manager	Yes	No	Yes
Queues	Yes	No	Yes
Message channels	Yes	No	Yes
Run MQSC commands	Utility or command file	No	Command line or command file
Persistence of MQSeries objects	Yes	All objects are on the server	Yes
Logging and media recovery	No	All objects are on the server	Yes
Automatic installation	Yes	Yes	Yes
Automatic start up	Yes	No	No
Supports MQSeries clients	No	Not applicable	Yes

Comparison with clients

Part 2. Installing MQSeries for Windows

Chapter 3. Installing MQSeries for Windows from diskettes

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

- "Components you can install"
- "Installing the product" on page 20
- "Changing your installation" on page 23

If you want to install MQSeries for Windows automatically from a LAN, see Chapter 4, "Installing MQSeries for Windows automatically" on page 27.

Components you can install

When you install MQSeries for Windows, you can select which components you install. The components are:

Base component
This contains the MQSeries for Windows runtime code, which includes:

- The queue manager
- The channel support
- The administration utilities:
  - Verify Install utility
  - Create and Go utility
  - Create Components utility
  - Standard Controls utility
  - Delete Components utility
  - Advanced Controls utility
  - MQSC Commands utility
  - Service Trace utility
  - Service Information utility
- The MQSC command file AMQSCOMW.TST, which creates the default MQSeries objects
- The READ.ME file, which contains information that was not available when this book was published

You must always install the Base component.

Online information
This contains:

- The online *MQSeries for Windows Command Reference*
- An online version of the *MQSeries for Windows User's Guide*
- The Quick Tour

You should always install the Online information component.

Toolkit and samples

This provides the header files, libraries, and sample MQSC command files to enable you to compile an MQSeries for Windows application. It also provides a set of sample applications (in both source and executable forms) written in the C language.

If you want to use MQSeries for Windows to run applications, you should install at least the Base component and the Online Information component. If you want to write applications, you need the Toolkit and Samples component as well.

Note: This book describes procedures for testing that you have set up your queue managers correctly. These procedures use the supplied sample programs, so to follow them you must install the Toolkit and Samples component.

If you are in any doubt about which components to install, install them all.

Disk space requirements

Table 3 shows the approximate disk space requirements for each component of MQSeries for Windows.

Table 3. Disk space requirements for MQSeries components

Component	Disk space
Base component	3.9 MB
Online information	600 KB
Toolkit and samples	410 KB

Installing the product

To ensure that MQSeries for Windows can find the required file paths, the file AUTOEXEC.BAT must be updated on each workstation. When you use the Installation Utility, you can specify whether this is done automatically, or whether you do it yourself manually. If you specify automatic update, a copy of the existing AUTOEXEC.BAT is taken as a backup. The backups are named AUTOEXEC.*nnn*, where *nnn* is the next sequential number for backups.

Note: After installation, check the length of the PATH statement in AUTOEXEC.BAT using a text editor. DOS allows a maximum of 127 characters, if this limit is reached, you must consider reconfiguring your start-up procedures to accommodate this.

MQSeries for Windows is supplied on diskettes. However, it is enabled for Configuration, Installation, and Distribution (CID), so you can put the installation files on a LAN server for easier access. For more information on this, see Chapter 4, “Installing MQSeries for Windows automatically” on page 27.

Installing on Windows 3.1 or Win-OS/2

To install the product on Windows 3.1 or Win-OS/2:

1. Insert the first MQSeries for Windows diskette in your diskette drive.
2. In Program Manager, select **Run...** from the File menu.
3. In the resulting window, type:

   ```
   A:\INSTALL
   ```

   and press Enter. If you are using a different drive (for example, drive B) for your diskette drive, change this command accordingly.
4. When the MQSeries for Windows window appears, read the text in the Installation window, then select **Continue**.
5. Proceed to "Completing the installation."

Installing on Windows 95

To install the product on Windows 95:

1. From the Windows 95 desktop, select **MyComputer**.
2. Open the Control Panel.
3. Select **Add/Remove Program**.
4. Select **Install**.
5. Insert the first MQSeries for Windows diskette in your diskette drive.
6. Select **Next**.
7. Make sure that the INSTALL program is highlighted.
8. Select **Finish**.
9. When the MQSeries for Windows window appears, read the text in the Installation window, then select **Continue**.
10. Proceed to "Completing the installation."

Completing the installation

When the MQSeries for Windows window appears, proceed as follows:

1. In the Install window, select the option that updates the CONFIG.SYS and AUTOEXEC.BAT files, then select **OK**.
2. In the Install - Directories window, select the components you want to install.

   To see a description of a product component, select the **Descriptions...** push button.

   If you are in any doubt about which components to select, install all the components. To do this, choose **Select All**.

   If you do not want to install the product on the default drives or paths, you can alter the drive and directory paths in this window.

If you are not sure how much disk space you have on each drive on your workstation, select the **Disk space...** push button. The Disk Space window shows you the available space on each drive.

When you are ready to proceed, select the **Install** push button. An indicator shows the progress of the installation process. The process takes only a few minutes.

The number of files that are transferred depends on which components you select. The source and target file names are continually updated.

3. Follow the on-screen instructions, changing diskettes when prompted.

When the installation is complete, a confirmation message is displayed and Program Manager displays the MQSeries for Windows program group. You should then use the information in Chapter 5, “Verifying your installation and configuration” on page 35 to test that the product has installed correctly, to create a test queue manager, and to run the sample programs to test that queue manager.

Directories after installation

Figure 3 shows the directory structure after you have installed all the components of MQSeries for Windows. If you do not install all the components, the directories installed will be a subset of those shown.

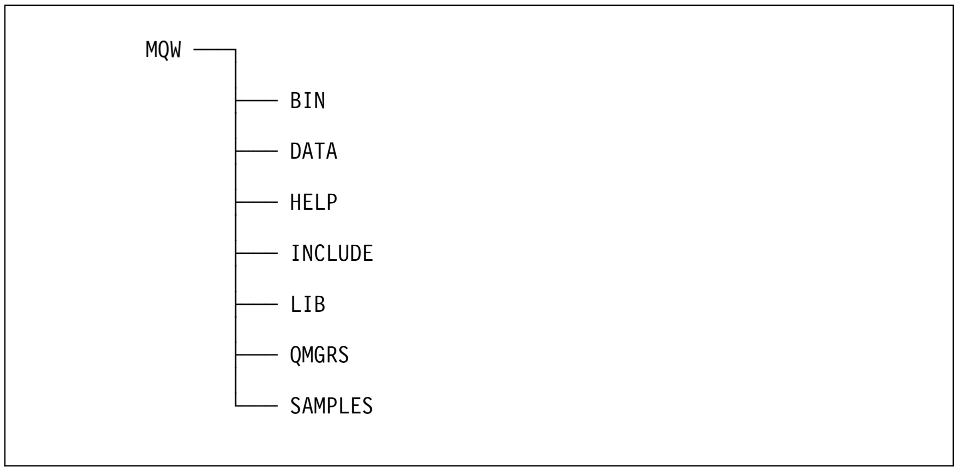

Figure 3. The default directory structure

In Figure 3 on page 22, the directories are:

Directory	***Function***
BIN	Executable files for the Base component.
DATA	Data files for the Base component, including the log files.
HELP	Online information.
INCLUDE	Include files for the Toolkit and samples.
LIB	Product libraries for the Toolkit and samples.
QMGRS	This directory will contain a subdirectory for each queue manager you subsequently create.
SAMPLES	The source code and executable files for the sample programs.

Moving the product to another directory

Although you can move the product files to a new directory after installation, you are recommended not to do this. The Installation Utility writes to the system initialization (WIN.INI) file and to the AUTOEXEC.BAT file. These files contain configuration parameters that include the location of the MQSeries product. To ensure the validity of these files, use the Installation Utility to delete the installed components, then reinstall the product at the new location. It is not sufficient simply to copy the files.

Changing your installation

After installing the product, you can use the Installation Utility to change your installation. Its icon is located in the MQSeries for Windows program group. The Installation Utility allows you to:

- Add any components of MQSeries for Windows you did not install initially
- Delete installed components that you no longer need
- Apply maintenance upgrades

To do any of these, you must have access to the MQSeries for Windows catalog file (named AMQICATW.ICF) that the installation utility uses to locate the installation or maintenance files. You do not need to understand the contents of the catalog file, but you do need to select the correct file.

Selecting a catalog

Many IBM products use the same mechanism for installation, so each one has its own catalog file. Therefore before you go any further, you must ensure that the current catalog (this is the last one you used) is the one you want to work with.

To select a catalog, first start the Installation Utility by double-clicking on its icon in the MQSeries for Windows program group.

Installing the product

From the Installation Utility window:

1. If your catalog and its associated files are on a diskette, insert the first diskette in the drive.
2. Select **Open catalog** from the File menu.
3. In the Open Drive Catalog window, select the required drive and type the name of the catalog you need. The name of the catalog for MQSeries for Windows is AMQICATW.ICF.
4. When the name of the correct catalog file is in the **Filename** field, click on **Open** to open the catalog file. The name of the new catalog is displayed in the Installation Utility window.

This makes the file you selected in the last step the current catalog. Any further actions, such as install, update, or delete, are performed using the files in this catalog.

Adding components after installation

If you have not installed all the components you need, you can add additional components later by using the Installation Utility.

To do this:

1. Double-click on the **Installation Utility** icon.
2. Click on the **Action** choice on the menu bar.
3. Click on **Install...**, and you see the Install window.

In the Install window, select the components you want to add.

Deleting components

To delete the product, or specific components:

1. Ensure that you have selected the correct catalog.
2. Double-click on the **Installation Utility** icon.
3. Select **Action** from the menu bar.
4. Select **Delete...**, and you see the Delete window.
5. Select the component or components you want to delete.
6. If you proceed with the delete, the Delete progress window is displayed.
7. When the product files have been deleted, exit from Windows and open a DOS window.
8. From the root directory of the boot drive, type:

   ```
   EPFIDEFI
   ```

   This runs the batch file `EPFIDEFI.BAT` to complete the delete process. This file is automatically put into the root directory of the boot drive when you begin the delete process, so do not expect to see it there normally.

9. If you are using Win-OS/2, delete the QMGRS directory. To do this, you could use the Win-OS/2 File Manager.

Applying maintenance updates

Maintenance updates are usually supplied on diskettes. You may choose to copy the update disks to a LAN server and perform the updates from there. Whatever the source of the updates, you must make the catalog associated with the updates the current catalog; see “Selecting a catalog” on page 23.

Read the READ.ME file supplied with the maintenance update to find out whether the update applies to your system and for instructions on how to apply it.

Reinstalling the product

You can reinstall the product if, for example, you are unsure of the status of an existing installation. To do this:

1. Insert the first MQSeries for Windows diskette in your diskette drive.
2. In Program Manager, select **Run...** from the File menu.
3. In the displayed window, type:

   ```
   A:\INSTALL
   ```

   and press Enter.

   If you are using a different drive (for example, drive B) for your diskette drive, change this command accordingly.
4. When the MQSeries for Windows window appears, select **Continue**.
5. Select **Delete the installed components and re-install** and click on the **Continue** push button.
6. The Delete window is displayed; see “Deleting components” on page 24 for information on how to proceed.

After the delete operation, the system continues automatically to the install operation.

Installing the product

Chapter 4. Installing MQSeries for Windows automatically

This chapter tells you how to install MQSeries for Windows components with little or no user intervention. You will find this information 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.

This chapter contains these sections:

What is automatic installation?

MQSeries for Windows is enabled for IBM Configuration, Installation, and Distribution, that is, it is CID-enabled. This means you can install MQSeries for Windows over a local area network (LAN) with little or no user intervention.

For automatic installation, you need a response file. This is an ASCII text file that contains a set of parameters (the responses) your installation requires. A response file provides the same information that you would have to specify if you were carrying out a normal (attended) installation.

You specify the parameters as keywords in the response file. These keywords determine how the installation will proceed. For example, you can use a response file to specify which components you are going to install on each workstation.

For a completely automatic installation, with no user intervention, you must use a software distribution manager, for example, IBM’s NetView Distribution Manager. You can use NetView Distribution Manager with any CID-enabled product. With this type of installation you must use an installation profile file. This contains the INSTALL command parameters and it refers to the relevant response files. Profile files are described in “Creating an installation profile” on page 30 and the automatic installation process is described in “The automatic installation process” on page 32.

Response files

Preparing for automatic installation

Before you can perform an automatic installation, you must copy the contents of the MQSeries for Windows installation diskettes to a local or LAN drive. For example, use the following command for each diskette:

```
XCOPY A:\*.* C:\IBMNVDM2\SHARE_A\IMG\MQW
```

The directory to which you copy the files must be accessible to all the workstations on which you want to install MQSeries for Windows.

Creating response files

An installation response file is an ASCII text file containing the options that control the installation of an MQSeries for Windows system.

In an installation response file you can specify:

- Whether CONFIG.SYS and AUTOEXEC.BAT should be updated automatically
- The names of the components to be installed
- Whether existing files should be overwritten
- The path for installation or maintenance
- Whether only backup versions of the product should be deleted

Typically, you have only one response file. However, if you need to install or update the product on a workstation with different options, you can use two files:

- A specific response file that contains options specific to a particular workstation
- A general response file that contains options common to all workstations

For example, you might use a general and a specific response file to install a particular component only on some workstations. See “Creating an installation profile” on page 30 for information about the INSTALL command parameters used to define response files.

Structure of a response file

There are two types of line in a response file:

Comment lines
Comment lines are either blank, start with an asterisk (*), or start with a semicolon (;).

Response lines
Response lines determine the options and configurations to install on the target system. They have syntax of the form:

```
keyword = value
```

Keyword-value pairs can be in any order, but there can be only one pair on each line. You can enter keywords in uppercase or lowercase letters; you cannot include spaces within keywords.

The maximum line length in a response file is 255 characters.

Keywords for response files

The following keywords are allowed in MQSeries for Windows response files:

CFGUPDATE
Specifies whether CONFIG.SYS and AUTOEXEC.BAT are updated automatically. Valid values for this keyword are:

AUTO	Automatically updates CONFIG.SYS and AUTOEXEC.BAT
MANUAL	Does not update CONFIG.SYS and AUTOEXEC.BAT

COMP
Specifies the name of a component to be installed. Valid values are:

- `COMP = Base component`
- `COMP = Online information`
- `COMP = Toolkit and samples`

You must not enclose the component name within quotation marks. The values you type are case sensitive, so type them exactly as they are shown.

You can specify multiple COMP keywords, one for each required component.

DELETEBACKUP
Specifies whether to delete the backup version of MQSeries for Windows. Valid values are YES and NO.

FILE Specifies the drive and directory for the product code; for example, the C:\MQW directory.

WORK
Specifies the drive and directory for the product data files. This keyword applies to the installation action only.

INCLUDE
Specifies which general response files to include with a specific response file. The format of this keyword is:

```
INCLUDE = filespec
```

where *filespec* specifies the name and path of the general response file to be included. If the file specification contains any global characters (* or ?), the first file found that matches the specification is included. If the specification is not valid, no general response file is included.

Note: You should not have more than five levels of included response files.

OVERWRITE
Specifies whether to automatically overwrite files during installation. Valid values for this keyword are YES and NO.

SAVEBACKUP
Specifies whether to save a backup version when the product is updated. Valid values for this keyword are YES and NO.

Installation profile

Example response file

MQSeries for Windows provides an example response file, named AMQICATW.RSP. It is supplied in the \MQW directory and it is shown in Figure 4.

```
     FILE         = C:\MQW
     CFGUPDATE    = AUTO
     OVERWRITE    = YES
     COMP         = Base component
     COMP         = Online information
     COMP         = Toolkit and samples
     DELETEBACKUP = NO
     SAVEBACKUP   = NO
```

Figure 4. The supplied sample response file AMQICATW.RSP

Creating an installation profile

The installation profile is an ASCII file that contains the parameters of the INSTALL command. These parameters provide the NetView installation instructions and point to the required response files.

The installation and maintenance parameters are:

```
INSTALL /A:action
        /C:catalog
        /G:include path
        /L1:error log
        /L2:history log
        /R:response file
        /S:source location
        /T:install directory
        /X
```

Note: You can enter the parameters in any order. You can use equals signs (=) instead of colons (:) in the parameters. Values can be uppercase or lowercase. The parameters must all be on one line.

INSTALL parameters

/A:*action*
Specifies the action to be performed by the installation program. Valid values for *action* are:

D	Delete an installed MQSeries for Windows system
I	Install a new MQSeries for Windows system
R	Restore a backed up MQSeries for Windows system
U	Update an installed MQSeries for Windows system

/C:*catalog*
Specifies the name of the catalog to be used. This is AMQICATW.ICF.

/G:*include path*
Specifies the drive and path of any general response file to be included by the specific response file. For more information about response files, see "Creating response files" on page 28.

/L1:*error log*
Specifies the drive, path, and file name of the error log file. The error log contains messages associated with installation, including confirmations and error messages. Messages are written to the error log only if you specify the /X parameter.

Specify a full drive and path; otherwise the error log is written to a temporary directory and could be lost. If you do not specify the /L1 parameter, no error log is maintained. If the file you specify already exists, log messages are added to it.

For example:

```
/L1:D:\LOG\LOGMQW.OUT
```

/L2:*history log*
Specifies the drive, path, and file name of the history log file. The history log contains entries for each file transferred, each object created, and each installation.

Specify a full drive and path; otherwise the history log is written to a temporary directory and could be lost. If you do not specify the /L2 parameter, no history log is maintained. If the file you specify already exists, log messages are added to it.

For example:

```
/L2:D:\LOG\LOGMQW.HIS
```

/R:*response file*
Specifies the drive, path and file name of a response file; see "Creating response files" on page 28.

For example:

```
/R:L:\USER\TEST.RSP
```

/S:*source location*
Specifies the drive and path containing the source files to be installed or updated.

/T:*install directory*
Specifies the drive and path on which the files are to be installed. If you specify this parameter, it overrides the FILE path specified in the response files.

/X Specifies that the installation is fully automatic.

When you specify this parameter, no progress indicator panel is shown and error messages are logged in the error log file. (You specify the path name of the error log file using the /L1 parameter.) If you do not specify all the information required for the action to complete, an error occurs.

If you do not specify the /X parameter, you are prompted for any information that the install program needs to complete the action. In this interactive mode of operation, progress indication is shown and error messages are displayed in secondary windows.

The automatic installation process

To start the automatic installation process:

1. Create a profile (.PRO file) and the necessary response (.RSP) files.
2. In NetView Distribution Manager, click on the **CDM Dialogs** icon.
3. From the **CDM Catalog** box, select **Build from profile** from the File menu.
4. In the **Build change file** box, enter the profile name in the **Change profile name** field. In the **Target file** field, type the required name of the change file that will be created by NetView as part of the installation process. Click on the **Build** push button.
5. When the change file has been built, select the change file from the **CDM Catalog** box and click on **Selected**.
6. Select **Install...** from the menu.
7. Select either **Force** or **No force**.
8. Select your workstation and click on the **Install** push button. MQSeries for Windows will be installed.

Return codes from the installation process

The installation process returns the following codes to a software distribution manager, such as NetView Distribution Manager/2:

- Successful termination.

00 00	No messages were logged.

- Unsuccessful termination.

08 00	A required data resource was not found.
08 04	Access to a required data resource was denied because the resource was already in use.
08 08	Access to a required data resource was denied because the appropriate authorization was missing.
08 12	A required data path was not found.
08 16	The product was not configured.

- Unsuccessful termination.

12 00	An I/O error (exception) occurred.
12 04	A device-not-ready exception occurred.
12 08	A not-enough-disk-space exception occurred.

- Unsuccessful termination.

16 00	An incorrect program invocation error was encountered. A required command line parameter was missing or was incorrect.
16 04	An unexpected condition was encountered, or an exit routine produced a non-CID defined return code.

- Successful termination. For these return codes, restart the workstation operating system before calling installation process again.

FE 00	No messages.
FE 04	Warning messages were logged.
FE 08	Error messages were logged.
FE 12	Severe error messages were logged.
FF *yy*	The install operation is not complete. *yy* can vary from 00 to FF.

All return codes are shown as two-byte hexadecimal values.

Automatic installation

Chapter 5. Verifying your installation and configuration

After you have installed MQSeries for Windows, use the procedures described in this chapter to test the installation, to create a sample queue manager and some queues, and to run the supplied sample programs to test that queue manager.

This chapter describes:

- "Verifying your installation"
- "Creating your test configuration" on page 37
- "Running the sample programs" on page 37

Note that to use the sample programs, you must have installed the Toolkit and Samples component of MQSeries for Windows.

If, after following the procedures in this chapter, you need to reinstall any of the components of MQSeries for Windows, first stop the active queue manager, then see "Changing your installation" on page 23.

If you want to connect two queue managers together, see Chapter 8, "Setting up and verifying two queue managers" on page 67.

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. If you have installed at least the base component, you can test the installation by using the Verify Install utility in the MQSeries for Windows program group.

The Verify Install utility performs the following tasks:

- Creating a queue manager
- Starting a queue manager
- Connecting to a queue manager
- Opening a queue
- Putting messages on a queue
- Getting messages from a queue
- Closing a queue
- Disconnecting from a queue manager
- Stopping a queue manager
- Deleting a queue manager

To start the Verify Install utility, double click on its icon in the MQSeries for Windows program group. The utility automatically starts the verification process; no further user interaction is required. The test takes only a few minutes.

Verifying your installation

If the verification is successful, the Verify Install utility displays the message:

```
MQSeries for Windows has successfully installed.
The product is now ready to use. (AMQ3000)
```

If this message is displayed, close the utility by selecting the **Exit** push button.

If any of the verification tasks fail, the utility stops and displays the message:

```
MQSeries for Windows has failed to install correctly.
Please see the MQSeries for Windows User's Guide for more information.
Reason: task failed.  AMQ3001
```

The message shows the verification *task* that failed. The most likely reasons for failure are:

- There is not enough space available to run the Verify Install utility on the disk on which MQSeries for Windows is installed.

  Make some space on the disk on which you have installed MQSeries for Windows, then retry the Verify Install utility.

  Note: The Service Information utility tells you which drive you have installed the product on and how much disk space remains.

- The MQSC command file used to create the default queues is missing.

  The file AMQSCOMW.TST must be in the \MQW\QMGRS directory. If you have moved this file, move it back to this directory; otherwise reinstall MQSeries for Windows.

- There are insufficient system resources available to run the Verify Install utility.

  Close some of the programs you are running, the retry the Verify Install utility.

- A queue manager named SYSTEM.INSTALL.QUEUE.MANAGER already exists.

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

If you can rectify the problem, do so, then restart the verification by selecting **Start Verify** from the File menu. Otherwise, you must uninstall the product and try again. If the error persists, contact your service administrator.

If you need to change your installation, see “Changing your installation” on page 23.

Note: You can use the Verify Install utility at any time to test that your MQSeries for Windows queue manager is working correctly.

Creating your test configuration

MQSeries for Windows supplies files you can use to create a queue manager for test purposes. To create this queue manager, use the Create and Go utility with the supplied initialization file, CREATEMQ.INI.

Start the Create and Go utility by double-clicking on its icon in the MQSeries for Windows program group. The first time you run the utility, you are prompted to provide registration information for MQSeries for Windows.

The utility uses the supplied initialization file to:

- Create a queue manager named SAMPLE_QM
- Load the MQSC command file named AMQSCOSW.TST, which creates some sample queues
- Start the queue manager
- Start the Standard Controls utility

You can then use the procedures described in the remainder of this chapter to put some messages on a queue, and, with the queue manager still running, get the messages from the queue.

Running the sample programs

To verify that the queue manager you have created is working correctly, run the sample programs that are supplied with MQSeries for Windows. The sample programs you will use are:

Putting Messages
The Putting Messages sample puts a message on a specified queue.

Getting Messages
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 the queue.

To run the sample programs, a queue manager must be active. If queue manager SAMPLE_QM (that you started using the Create and Go utility) is not still running, restart it by double clicking on the icon of the Standard Controls utility in the MQSeries for Windows program group.

Running the samples

Putting messages on a queue

To put messages on a queue:

1. Start the Putting Messages sample by double-clicking on its icon in the MQSeries for Windows program group.

   This starts the program and connects it to the active queue manager.

2. In the **Queue** field of the Putting Messages Sample window, type the following name in uppercase letters:

   ```
   SYSTEM.SAMPLE.LOCAL
   ```

3. Select the **Open** push button to open the queue. Check that the reason code displayed in the **API Return Code** field is zero; if it is not, the computer beeps.

   If the reason code is not zero, the program cannot open the queue. The reason codes are listed in Appendix E, "Reason codes" on page 175. You are most likely to see the following reason codes:

2059	This means the queue manager is not running. If this happens, close the sample by selecting **Exit** from the File menu. Start a queue manager using the Standard Controls utility before you retry 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. Select the **Put** push button. This puts the message on the queue and the message text also appears in the **Log list** box.
6. Check that the reason code displayed in the **API Return Code** field is zero.
7. Repeat steps 4 and 5 to put other messages on the queue.

You can leave the sample running so you can continue putting messages after you have used the Getting Messages sample program to remove some from the queue.

When you have finished using the Putting Messages sample program:

1. Select the **Close** push button. This closes the queue and disconnects the sample from the queue manager.
2. Close the window of the sample program.

Getting messages from the queue

To get messages from a queue:

1. Start the Getting Messages sample by double-clicking on its icon in the MQSeries for Windows program group.

   This starts the program and connects it to the active queue manager.

2. In the **Queue** field of the Getting Messages Sample window, type the following name in uppercase letters:

   ```
   SYSTEM.SAMPLE.LOCAL
   ```

3. Select the **Open** push button to open the queue. Check that the reason code displayed in the **API Return Code** field is zero; if it is not, the computer beeps.

   If the reason code is not zero, the program cannot open the queue. This is likely to be for one of the reasons explained in “Putting messages on a queue” on page 38.

4. Select the **Get** push button. This retrieves the oldest message from the queue and displays it in the **Data list** box. The **Length** field shows the length of the message data.

5. Check that the reason code displayed in the **API Return Code** field is zero.

   You are most likely to see the following reason codes:

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.
2219	This means the sample is waiting for a message to arrive. If there are no messages on the queue, the sample waits for 15 seconds, then it returns reason code 2033. If you select the **Get** push button within this waiting period, code 2219 tells you that the program is busy.

   The reason codes are listed in Appendix E, “Reason codes” on page 175.

6. You can keep repeating step 4 to get any other messages from the queue.

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. Select the **Close** push button. This closes the queue and disconnects the sample from the queue manager.
2. Close the window of the sample program.

Running the samples

When you are using the Getting Messages sample program, remember:

- Getting the messages from the queue removes them from the queue.
- The messages are retrieved in the same order in which they were put on the queue.
- The put and get operations are independent, so the put and get programs can operate at the same time, or one at a time.

When you want to stop the queue manager:

- Go to the Standard Controls utility
- Select (highlight) the SAMPLE_QM queue manager
- Select **Stop** from the Selected menu

Part 3. Setting up queue managers

Chapter 6. Working with a queue manager

After you have set up a test queue manager, and verified that it runs the supplied sample programs correctly, you are ready to learn how to create your own queue managers. This chapter tells you how to use the administration utilities supplied with MQSeries for Windows to create your own queue managers and queues, and to monitor your queue manager. This chapter covers:

- “Creating components using the Create and Go utility” on page 44
- “Creating queue managers and queues using the Create Components utility” on page 44
- “Monitoring queue managers and queues using the Standard Controls utility” on page 46
- “Deleting queue managers and queues using the Delete Components utility” on page 51
- “Other MQSeries for Windows utilities” on page 52

There are two ways of creating queue managers and queues:

- You can use the Create and Go utility to create and start components automatically using definitions supplied in an initialization file named CREATEMQ.INI. You used this utility in “Creating your test configuration” on page 37 to create and test a queue manager using the supplied definitions.

  To use the Create and Go utility to create your own components, see “Creating components using the Create and Go utility” on page 44.
- You can use the Create Components utility to create queue managers and queues individually. This is described in “Creating queue managers and queues using the Create Components utility” on page 44.

You can also use the Create and Go utility and the Create Components utility to create the channels, channel groups, and transport links that you will need if you are going to work with more than one queue manager. For further information, see:

- Chapter 7, “Using more than one queue manager” on page 53
- Chapter 12, “Working with transport links” on page 113

Creating components using the Create and Go utility

To use the Create and Go utility, you need a new initialization (INI) file that contains definitions of the components you want to create. The file must be named CREATEMQ.INI.

If someone like your systems administrator gives you an INI file for this purpose, copy the file to replace the one named CREATEMQ.INI that MQSeries for Windows supplies in the \MQW directory. Then start the Create and Go utility by double clicking on its icon in the MQSeries for Windows program group. The first time you run the utility, you are prompted to provide registration information for MQSeries for Windows.

The utility uses the definitions in the INI file to create the components, then start them. If you need to know how to create an INI file, see Chapter 11, “Creating an INI file for the Create and Go utility” on page 95.

Creating queue managers and queues using the Create Components utility

To create queue managers and queues (and other MQSeries components) individually, use the Create Components utility. To start the utility, click on its icon in the MQSeries for Windows program group.

Creating a queue manager

When you have started the Create Components utility, to create a queue manager:

1. Select the **Queue Manager** push button.

   This displays the Create Queue Manager window.

2. Complete the fields in this 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.

   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: (`no description`). Remember that, at a later date, a description could help you to identify the queue manager you want to work with.

You can type any text you like in this field.

Load MQSC file for the sample programs
If you want to run the supplied sample programs after you have created your queue manager, make sure this checkbox is marked.

If you do this, the Create Components utility, when it creates the queue manager, runs the MQSC command file that generates the queues and channels used by the sample programs.

The sample file is named AMQSCOSW.TST; for more information on it, see “Objects for the sample programs” on page 172.

You can load this file only if you have installed the Toolkit and samples component of MQSeries for Windows.

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 MQSeries for Windows for the first time, or just running the samples, you can leave this field blank.

If your administrator gives you an MQSC command file to run when you create your queue manager, specify its name in this field.

To find out more about MQSC commands, see Chapter 10, “Using MQSC commands” on page 85.

3. When you have completed the input fields in this window, click on **OK**.

   An indicator in the window shows the progress of the create process. When the queue manager has been created, a confirmation message is displayed.

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 171.

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. The Create Components utility creates the log file in a separate directory for each queue manager. For example, for a queue manager named TEST, the log file is C:\MQW\QMGRS\TEST\MQSC.LOG (if you installed MQSeries for Windows in directory C:\MQW). If the Create Components utility cannot create a directory using the first eight characters of the queue manager name (for example, if a directory of that name already exists), it creates a directory whose name is based on those eight characters. The name of the queue manager appears near the top of the log file. For information about these error messages, see Appendix F, “Error messages” on page 179.

If you create only one queue manager, that queue manager is started automatically when you use the Standard Controls utility or the Advanced Controls utility. If you create more than one queue manager, you can use the Standard Controls utility or the Advanced Controls utility to select which queue manager you want to start automatically (see “The Queue Managers view” on page 49).

When you have created your queue manager, remember to close the Create Components utility.

Creating a queue

To create a queue, use the Create Components utility. After you have started the utility, select the **Queue** push button in the Create Components window, then complete the fields in the Create Queue window. You must specify:

- The name of the queue manager for which you want to create a queue
- The name of the queue you want to create
- The type of queue you want to create

You can copy the queue definition from that of any of the existing queues (of the type you have chosen) owned by the queue manager. If you do this, you can change the attributes of the queue using the Create Queue window.

When you have created your queue, close the Create Components utility.

Monitoring queue managers and queues using the Standard Controls utility

The Standard Controls utility allows you to perform standard operations on your queue manager and queues:

- Starting a queue manager
- Stopping a queue manager
- Specifying the name of the queue manager (if any) that is started automatically when the utility is restarted
- Viewing the status of the local queue managers
- Viewing the attributes of the active queue manager
- Viewing the names and attributes of the queues owned by the active queue manager

You can also use the Standard Controls utility to work with channels and channel groups (see Chapter 7, “Using more than one queue manager” on page 53) and transport links (see Chapter 12, “Working with transport links” on page 113).

Starting the Standard Controls utility

To start the Standard Controls utility, click on its icon in the MQSeries for Windows program group. The first time you run the utility, you are prompted to provide registration information for MQSeries for Windows.

The Standard Controls utility window tells you the status of the components you choose to view. The window has three parts:

- A menu bar that provides access to the functions provided by the Standard Controls utility
- A tool bar containing icons that let you switch between the four views of the window
- The main area of the window, which shows the status of the components you are viewing

There are four views of the Standard Controls utility. They each allow you to display the status of different components. The four views are:

The **Connection Monitor** view shows the status of the connections to the queue manager you select in the Queue Managers view. This is described in “The Connection Monitor view” on page 48.

The **Queue Managers** view shows the status of all the queue managers on the workstation. This is described in “The Queue Managers view” on page 49.

The **Channel Groups** view shows the status of the channel groups of the active queue manager. You use channel groups when you create channels to communicate with another queue manager. For more information on this view, see “The Channel Groups view” on page 61.

The **Transport Links** view shows the status of the transport links of the active queue manager. Administrators use transport links to control the costs of connections. For more information on this view, see “The Transport Links view” on page 115.

The first time you use the Standard Controls utility, it shows the Connection Monitor view. If you change the view (using the icons on the tool bar or using the View menu), it remembers the view you used last and shows this next time you start the utility.

Standard Controls utility

The Connection Monitor view

The Connection Monitor view shows the current status of all the components associated with the selected queue manager. This is what the icons in the status window mean:

The **Leaf Node** icon represents your workstation. This icon is always present and its appearance does not change.

The **Queue Manager** icon represents the queue manager you are using. If the icon contains a check mark, the queue manager is active (that is, it is running). If there is no check mark, the queue manager is stopped.

The name to the right of this icon is the name of the queue manager.

This icon also represents your workstation. It is always present (and its appearance does not change) when you are not communicating with other queue managers. This is because you can send messages only to applications that use the local queue manager.

When you want to communicate with other queue managers, you must define channels to that queue manager. When you do this, the channels are also shown in this view, together with the destination queue manager. This is described in “The Connection Monitor view” on page 62.

The Queue Managers view

To move to the Queue Managers view, select **Queue Managers** from the View menu, or select the **Queue Managers** icon on the tool bar.

The Queue Managers view displays a list of the queue managers you have created. This is what the icons in the status window mean:

There is one **Queue Manager** icon for each queue manager you have created. If the icon contains a check mark, the queue manager is active (that is, it is running). If there is no check mark, the queue manager is stopped.

The name to the right of this icon is the name of the queue manager.

The **Monitor** icon identifies the queue manager whose details are displayed when you move to the Connection Monitor view.

To monitor a different queue manager, select that queue manager in the Queue Managers view, then select **Monitor** from the Selected menu.

The **Autostart** icon identifies the queue manager that is started automatically when you restart the Standard Controls utility.

To start a different queue manager automatically, select that queue manager in the Queue Managers view, then select **Autostart** from the Selected menu.

If you create only one queue manager, that queue manager is set to start automatically. You can remove this setting if you want.

Starting a queue manager

In MQSeries for Windows, you can run only one queue manager at a time. If you attempt to start a queue manager when another one is active, you are prompted to confirm that you want to stop the active one.

To start a queue manager, select its name in the Queue Manager view of the Standard Controls utility. Double click on the name, or select **Start** from the Selected menu.

Note: Alternatively, with the mouse pointer over the name of the queue manager you want to use, click on mouse button 2. This shows a pop-up menu that contains the same choices as the Selected menu.

You can also start a queue manager using the same methods in the Connection Monitor view.

When the queue manager starts, its icon shows a check mark.

Standard Controls utility

Stopping a queue manager

To stop a queue manager, select its name in the Queue Manager view of the Standard Controls utility. Double click on the name, or select **Stop** from the Selected menu.

Note: Alternatively, with the mouse pointer over the name of the queue manager you want to use, click on mouse button 2. This shows a pop-up menu that contains the same choices as the Selected menu.

You do not have to stop a queue manager before you start another one; when you start another queue manager, you are prompted to confirm that you want to stop the active one.

When you stop a queue manager, you can no longer put messages on, or get them from, any queues owned by that queue manager. Also, any channels that the queue manager owns are stopped.

Note: The queue manager stops when you close the Standard Controls utility.

Specifying an autostart queue manager

Use the **Autostart** icon of the Queue Managers view to select the queue manager you want to be started automatically when the Standard Controls utility starts. See "The Queue Managers view" on page 49.

Viewing the status of the local queue manager

Use the Connection Monitor view to display the status of the local queue manager. This view shows whether or not the queue manager is running. See "The Connection Monitor view" on page 48.

To display the status of all your queue managers, use the Queue Managers view (see "The Queue Managers view" on page 49).

Viewing the attributes of the active queue manager

You can view the attributes of the active queue manager. To do this, go to the Queue Managers view and select the active queue manager. Then select **Attributes...** from the Selected menu. In the resulting window, you can select each attribute to see its value.

If you want to change the attributes, you must use the Advanced Controls utility. For more information on these attributes, see the online help.

Viewing the attributes of a queue

You can list the queues of the active queue manager and view their attributes. To do this, go to the Queue Managers view and select the active queue manager. Then select **Queues** from the Selected menu. The resulting Queues for Queue Manager window shows all the queues that the active queue manager owns.

To view the attributes of a queue, select that queue, then select **Attributes** from the Selected menu. In the resulting window, you can select each attribute to see its value.

If you want to change the attributes, you must use the Advanced Controls utility. For more information on these attributes, see the online help.

Deleting queue managers and queues using the Delete Components utility

Each queue manager you create uses system resources, so you should delete a queue manager you no longer use. Similarly, you should delete any queues that a queue manager no longer uses. You can do both of these operations using the Delete Components utility.

Deleting a queue manager

To delete a queue manager, use the following procedure:

1. Close any MQSeries for Windows utilities that are running.
2. Click on the icon of the Delete Components utility in the MQSeries for Windows program group and select the **Queue Manager** push button.

   This displays the Delete Queue Manager window.
3. Select from the list the queue manager you want to delete.

   When you select a queue manager, its description is displayed if one was provided when the queue manager was created. The description can be useful in helping you to select the correct queue manager to delete.

   Note: You can delete only one queue manager at a time.
4. Click on the **OK** push button to start the delete process.

   You are prompted to confirm that you want to delete the queue manager, because when the queue manager is deleted, any queues (and the messages on them) and channels that it owns are also deleted.

When the queue manager has been successfully deleted, you can reuse its name when you create another queue manager.

If you uninstall the product, all queue managers are automatically deleted.

Utilities

Deleting a queue

To delete a queue, you use the Delete Components utility:

1. Select the **Queue** push button in the Delete Components window.
2. Complete the fields in the Delete Queue window.
3. Select the name of the queue manager that owns the queue you want to delete, and the name of the queue.

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

Deleting other components

You can also use the Delete Components utility to delete channels, channel groups, and transport links.

Other MQSeries for Windows utilities

In addition to the utilities described in this chapter, MQSeries for Windows provides the following to help you work with its components:

The Advanced Controls utility
Use this utility to start, stop, and view the status of MQSeries for Windows components, just like the Standard Controls utility. In addition, the Advanced Controls utility lets you change the attributes of the components. It is described in “The Advanced Controls utility” on page 82.

The MQSC Commands utility
You can also use *MQSC commands* to create, alter, or delete MQSeries objects when you have started a queue manager. MQSeries for Windows provides the MQSC Commands utility to help you do this. To find out more about this utility, see Chapter 10, “Using MQSC commands” on page 85.

Note: You can run only one utility at time.

Chapter 7. Using more than one queue manager

When you understand how you can work with a single queue manager, you are ready to learn what is involved when you want to create or use a network of queue managers, each one running on a different computer.

This chapter tells you how to set up those queue managers so that you can send MQSeries messages between them. It assumes that the computers on which the queue managers are running are already connected together using a local area network (LAN) or other network that uses TCP/IP for its communication.

This chapter covers:

Connecting two queue managers

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 computer or on any other, no matter where that computer 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*.

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.

Many queue managers

For example, the sample file MARS.TST that is used in Chapter 8, “Setting up and verifying two queue managers” on page 67 uses the queue definitions shown in Figure 5.

```
* 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

The message is transferred 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 55. 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

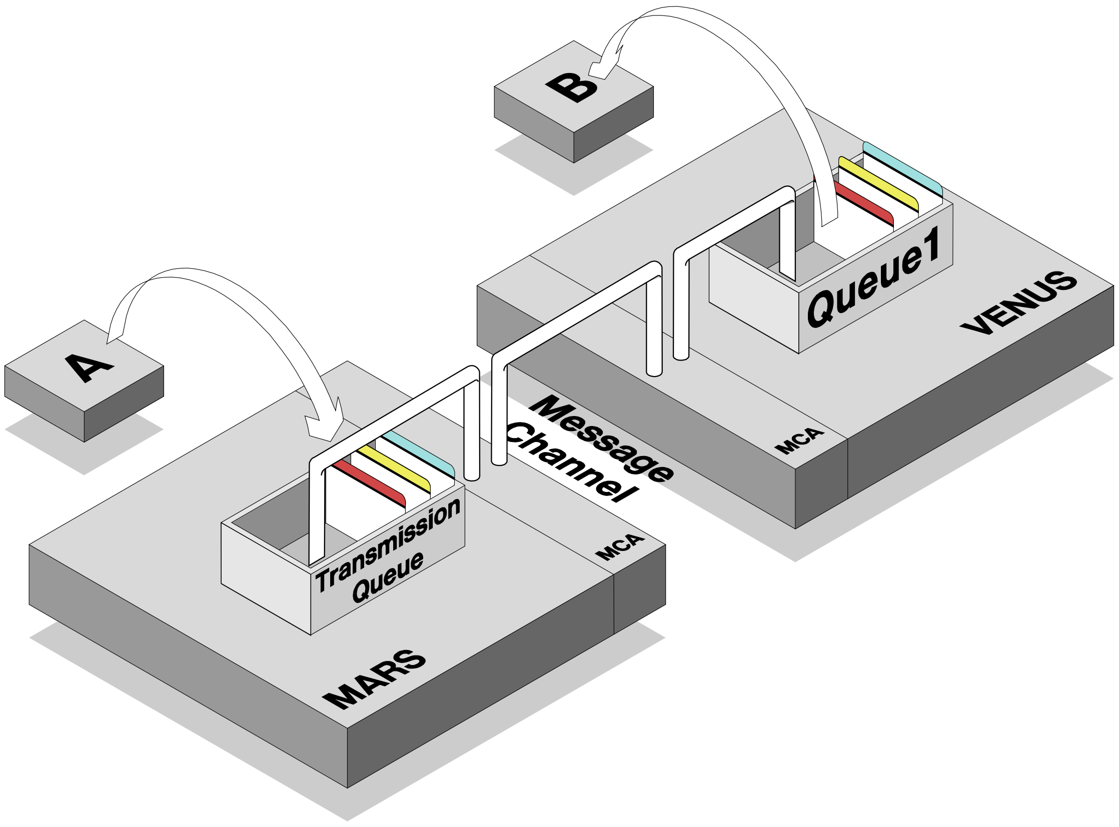

Figure 6. Communication between two queue managers. Application A puts a message on a local definition of Queue1. Application B gets the message from Queue1.

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.

Starting a channel

An MCA can act either as a ***caller*** or as a ***responder***. A caller MCA starts a channel by sending a connection request to a responder MCA. MQSeries for Windows supplies a program known as the *channel listener*, which listens for connection requests from caller MCAs. The connection requests contain the address of the MCA for which the request is aimed, so the listener can 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.

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` and `VENUS.TO.MARS`.

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, so 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).

Channel definitions

You can use any of the following combinations when you define the two ends of a channel:

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 the same 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 server-receiver channel.

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.

Channel groups

Using channel groups

In MQSeries for Windows, you can start, monitor, and stop channels only as a group, so each channel must belong to a *channel group*. A channel group is an MQSeries for Windows component; it is simply a named collection of channels and it is owned by a queue manager. Each queue manager can own many channel groups, but only one group can be active at a time. If you want to start another group, you must confirm that you want to stop the one that is already running.

You must create a channel before you can add it to a channel group. A group can contain a maximum of 32 channels, 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 Create Components utility, but you must first stop the queue manager. To add a channel to an existing channel group, or to change the attributes of the group, use the Advanced Controls utility.

You can specify that one channel group starts automatically when either the Standard Controls utility or the Advanced Controls utility starts. All the channels in that group are started automatically at that time.

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:

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.

Creating channels and channel groups

This section tells you how to create either a channel or a channel group using the windows of the Create Components utility. You can also create channels using MQSC commands, either when you create the queue manager using the Create Components utility, or by using the MQSC Commands utility.

Creating a channel

The procedure for creating a channel is similar to that for creating a queue. When you have started the Create Components utility, select the **Channel** push button in the Create Components window, then complete the fields in the Create Channel window. You must specify:

- The name of the queue manager for which you want to create a channel
- The name of the channel you want to create
- The type of channel you want to create

You can copy the channel definition from that of any of the existing channels (of the type you have selected) owned by the queue manager. If you do this, you can change the attributes of the channel using the Create Channel window.

Creating channels

If the channel you are creating can be a caller, make sure you specify some retries. Retries allow the channel to recover from problems that occur during start up of the channel. For example, the listener at the responding end could be busy starting another responder MCA, so it could miss a connection request; retries give it more chances to catch the request.

Note: You cannot specify retries for a requester channel.

When you have created your channel, remember to close the Create Components utility.

Creating a channel group

To create a channel group, use the Create Components utility like you did when you created a channel. Remember that you must create the channels first.

When you have started the Create Components utility, select the Channel Group push button in the Create Components window, then complete the fields in the Create Channel Group window. You must specify:

- The name of the queue manager for which you want to create a channel group
- The name of the channel group you want to create

You are prompted to select which existing channels you want to add to the channel group. Initially, all the channels owned by the queue manager are selected. Deselect those channels you do not want in the channel group.

You can have a maximum of 32 channels (or 31 and the listener) in a channel group.

If you need to add the listener to this channel group, make sure the **Start Listener** checkbox is marked.

If you want to add more channels to the group after you have created it, you must use the Advanced Controls utility.

When you have created your channel group, remember to close the Create Components utility.

Monitoring channel groups

The Standard Controls utility allows you to:

- View the status of a channel group using the Channel Groups and Connection Monitor views
- Start a channel group
- Stop a channel group
- View the attributes of a channel group
- View the attributes of a channel
- View the status of each channel owned by the active channel group

If you want to change any attributes, you must use the Advanced Controls utility.

The Channel Groups view

The Channel Groups view of the Standard Controls utility shows the status of all the channel groups owned by the active queue manager. This is what the icons in the status window mean:

There is one **Channel Group** icon for each channel group you have created. If the icon contains a status indicator (a check mark or an exclamation mark), the channel group is active (that is, it is running). If there is no status indicator, the channel group is stopped.

The name to the right of this icon is the name of the channel group.

The **Monitor** icon identifies the channel group whose details are displayed when you move to the Connection Monitor view.

To monitor a different channel group, select that channel group in the Channel Groups view, then select **Monitor** from the Selected menu.

The **Autostart** icon identifies the channel group that is started automatically when you restart the Standard Controls utility.

To automatically start a different channel group, select that channel group in the Channel Groups view, then select **Autostart** from the Selected menu.

If you create only one channel group, that channel group is set to start automatically. You can remove this setting if you want.

Monitoring channels

The Connection Monitor view

The Connection Monitor view shows the current status of all the components associated with the selected queue manager. When we saw it in "The Connection Monitor view" on page 48, this view contained icons for the leaf node and the queue manager only. But now that you are using channels to communicate with another queue manager, this view shows the channel group and that queue manager as well. This is what the icons in the status window mean:

The **Leaf Node** icon represents your workstation—this is your end of the connection.

The **Queue Manager** icon represents the queue manager you are using. If the icon contains a check mark, the queue manager is active (that is, it is running). If there is no check mark, the queue manager is stopped.

The name to the right of this icon is the name of the queue manager.

The **Channel Group** icon represents the channel group you are using. The icon also shows the status of the channel group:

- If there is any status mark on the icon, the channel group is active. If there is no mark, the channel group is stopped.
- If the icon contains a check mark, all the channels in the group are running.
- If the icon contains an exclamation mark (!), one or more of the channels in the group is stopped.

The name to the right of this icon is the name of the channel group.

The **Server** icon represents the server end of the connection. This is the queue manager with which you are communicating.

For an MQSeries application (represented by the leaf node icon at the top of the status window) to be able to communicate with a remote queue manager (represented by the server icon at the bottom of the status window), all the components between them must be in the active state. So for a successful connection, all the icons between the leaf node and the server must contain check marks. This view can give a quick visual confirmation that communication is possible.

Starting a channel group

In MQSeries for Windows, only one channel group can be active (that is, running) at a time. If you try to start a channel group when another one is active, you are prompted to confirm that you want to stop the active one.

To start a channel group, select its name in the Channel Groups view of the Standard Controls utility. Double click on the name, or select **Start** from the Selected menu. Wait until the channel group starts. When it does, a check mark is added to the channel group icon. You can also start a channel group using the same method in the Connection Monitor view.

If there is a transport link defined on the active queue manager but that transport link is not active when you start the channel group, the group is put into a quiesced state. From this state, the channel group is started automatically when the transport link starts. For information on transport links, see Chapter 12, “Working with transport links” on page 113.

Stopping a channel group

To stop a channel group, select its name. Double-click on it, or select **Stop** from the Selected menu.

You do not have to stop a channel group before you start another one; when you start another channel group, you are prompted to confirm that you want to stop the active one.

When you stop a channel group, its channels stop. If the channels are carrying any messages, those messages are preserved on the transmission queue at the sending end of the channel.

Note: The channel group stops when you close the Standard Controls utility.

Viewing the attributes of a channel group

You can view the attributes of any channel group at any time:

1. Go to the Channel Groups view and select the channel group you want to view.
2. Select **Attributes...** from the Selected menu.

   In the resulting window, you can see which channels are included in the group, and whether or not it includes the listener.

If you need to change which channels are included in the group, you must use the Advanced Controls utility.

Monitoring channels

Viewing the attributes of a channel

You can list the channels in the active channel group and view their attributes:

1. Go to the Channel Groups view and select the active channel group.
2. Select **Channels** from the Selected menu. The resulting Channels for Channel Group window shows all the channels that belong to the group.

To view the attributes of a channel:

1. Select the required channel.
2. Select **Attributes** from the Selected menu.
3. In the resulting window, select each attribute to see its value.

For more information on these attributes, see the online help. If you need to change any of the attributes, you must use the Advanced Controls utility.

Viewing the status of a channel

In addition to the ordinary attributes of a channel (such as the maximum message length and the name of the transmission queue), channels also have status attributes that report transient information, such as the number of bytes sent or received.

You can list the channels in the active channel group and view their status:

1. Go to the Channel Groups view and select the active channel group.
2. Select **Channels** from the Selected menu.

   The resulting Channels for Channel Group window shows all the channels that belong to the group. The icon for each channel contains a status indicator that gives a visual indication of the state of that channel:

   - A check mark means that the channel is running.
   - An exclamation mark (!) means that the channel is in an intermediate state (such as retrying or stopping).
   - A cross means that the channel has stopped.
   - No status indicator means that the channel has not started.

To get more information about the status of a channel, select that channel, then select **Status** from the Selected menu. 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 channel.

Deleting channel groups and channels

When you no longer need to use a channel or channel group, delete it using the Delete Components utility.

Deleting a channel group

To delete a channel group:

1. Close any MQSeries for Windows utilities that are running.
2. Click on the icon of the Delete Components utility in the MQSeries for Windows program group and select the **Channel Group** button.

   This displays the Delete Channel Group window.
3. Select from the list the name of the queue manager that owns the channel group you want to delete.
4. Select from the list the channel group you want to delete.

   When you select a channel group, its description is displayed. The description may help you select the correct channel group to delete.

   Note: You can delete only one channel group at a time.
5. Click on the **OK** push button to start the delete process.

   You are prompted to confirm that you want to delete the channel group.

When the channel group has been successfully deleted, you can reuse its name when you create another channel group.

If you uninstall the product, all channel groups are automatically deleted.

Deleting a channel

To delete a channel:

1. Select the **Channel** push button in the Delete Components window.
2. Complete the fields in the Delete Channel window. You must select the name of the queue manager that owns the channel you want to delete, and the name of the channel.

Deleting channels

Chapter 8. Setting up and verifying two queue managers

This chapter describes how to set up two queue managers so they can send messages to each other. It then describes how to use the sample programs supplied with MQSeries for Windows to verify that the two queue managers are set up correctly. It includes:

- “Setting up the two queue managers on Windows”
- “Setting up the two queue managers yourself” on page 71
- “Verifying the configuration” on page 73
- “When one of your queue managers is on a different platform” on page 76
- “Other tests you may want to try” on page 77

Follow the procedure in this chapter to create a queue manager on each of two workstations. In the examples in this chapter, the two queue managers are named MARS and VENUS. MQSeries for Windows supplies files that contain definitions for these two queue managers, and for the components they use. 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 queue managers.

   If both of your queue managers are to run on MQSeries for Windows, see “Setting up the two queue managers on Windows.”

   If one of your queue managers is to run on another MQSeries product, see “When one of your queue managers is on a different platform” on page 76.

2. Running the sample programs to verify your configuration.

   This is described in “Verifying the configuration” on page 73.

Setting up the two queue managers on Windows

Before you can set up your two queue managers, install MQSeries for Windows on each of the two workstations you are going to use. Make sure you install the Toolkit and samples component on each one.

It is then advisable to use the TCP/IP **ping** command to test the TCP/IP connection between them.

Two queue managers

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, but if your TCP/IP uses a domain name service, you can create a text alias for your IP address (but note that this alias is case sensitive). If you are using IBM TCP/IP for DOS, you can use the CUSTOM program to find your IP address.

When you know the addresses, try pinging each workstation by typing the **ping** command at a DOS command prompt on the other one; 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 Create and Go utility to create a queue manager, and its queues and channels, on each workstation.

Note: If you do not want to use the Create and Go utility to set up your queue managers, see “Setting up the two queue managers yourself” on page 71.

Setting up VENUS

Set up VENUS as follows:

1. If you have installed MQSeries for Windows on a drive other than C:, edit the file VENUS.INI (supplied in the directory \MQW\SAMPLES) to change references to this drive.
2. Make a backup copy of the supplied file named CREATEMQ.INI.

   For example:

   ```
   copy C:\MQW\CREATEMQ.INI C:\MQW\CREATEMQ.OLD
   ```

3. Copy the file C:\MQW\SAMPLES\VENUS.INI to replace the supplied file C:\MQW\CREATEMQ.INI.

   This is because the Create and Go utility uses only a file named CREATEMQ.INI.

4. Select the Create and Go utility. This uses the definitions in the file CREATEMQ.INI (supplied as VENUS.INI) to:
   - Create and start a queue manager named VENUS.
   - Run the MQSC command file named VENUS.TST. This creates the three queues and two channel definitions on MARS that are shown in Figure 7 on page 69.
   - Create and start a channel group named VENUSGroup.
   - Start the Advanced Controls utility.
   - Start the listener.

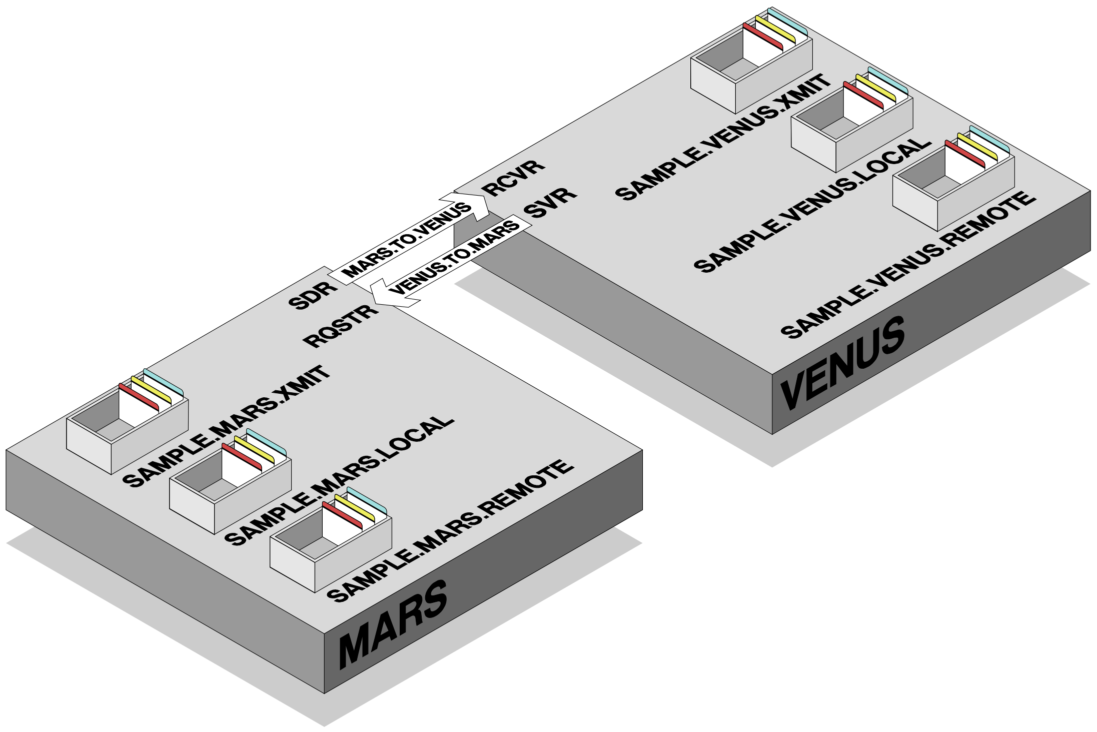

Figure 7. The queues and channels that the supplied INI files create

If the Create and Go utility is successful, a confirmation message is displayed and the Advanced Controls utility is started. If the Advanced Controls utility successfully starts the listener, a check mark appears beside the Channel Group icon in the window of the Advanced Controls utility.

VENUS is now ready to respond to incoming channel-connect requests.

Setting up MARS

On your second workstation, set up MARS as follows:

1. If you have installed MQSeries for Windows on a drive other than C:, edit the file MARS.INI (supplied in the directory \MQW\SAMPLES) to change references to this drive.
2. Make a backup copy of the supplied file named CREATEMQ.INI.

   For example:

   ```
   copy C:\MQW\CREATEMQ.INI C:\MQW\CREATEMQ.OLD
   ```
3. Copy the file C:\MQW\SAMPLES\MARS.INI to replace the supplied file C:\MQW\CREATEMQ.INI.

   This is because the Create and Go utility uses only a file named CREATEMQ.INI.

Two queue managers

4. Edit the MQSC command file named MARS.TST. (This file is in the \MQW\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. Select the Create and Go utility. This uses the definitions in the file CREATEMQ.INI (supplied as MARS.INI) to:
   - Create and start 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 7 on page 69.
   - Create and start a channel group named MARSGroup.
   - Start the Advanced Controls utility.

When the Advanced Controls utility starts the MARSGroup channel group, this sends channel-connect requests to VENUS. If this is successful, a check mark appears beside the channel group icon in the window of the Advanced Controls utility.

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 using the Advanced Controls utility.

- There is a TCP/IP problem. Restart the workstation and try again. You do not need to re-create the queue manager.

Now you can use the sample programs to verify that your queue managers are operating correctly. Proceed to “Verifying the configuration” on page 73.

Setting up the two queue managers yourself

Note

Read this section only if you do not want to use the Create and Go utility to set up your queue managers. You may want to use the information supplied here to help you understand how to create your own queues, channels, and channel groups.

Before you can set up your two queue managers, install MQSeries for Windows on each of the two workstations you are going to use. Make sure you install the Toolkit and samples component 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 queue managers on Windows” on page 67.

Setting up VENUS yourself

Set up VENUS as follows:

1. Create a queue manager called VENUS, using the Create Components utility.

   Specify the MQSC command file VENUS.TST. In a default installation, this is in the \MQW\SAMPLES subdirectory.

2. Create a channel group named VENUSGroup on the VENUS queue manager, noting the following:

   - Ensure that the group does not contain any channels. If there are already some channels listed in the **Name** field, deselect them. To do this, select the first name, press and hold down the Ctrl key, then select each of the remaining names.

   - Ensure that you mark the **Start Listener** box.

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 69.

2. Create a queue manager called MARS, using the Create Components utility.

   Specify the MQSC command file MARS.TST.

Two queue managers

3. Create a channel group named MARSGroup on the MARS queue manager. Add only the following two channels to the group:

   - MARS.TO.VENUS (this is a sender channel)
   - VENUS.TO.MARS (this is a requester channel)

   Note: To add more than one channel to a group:

   a. Select the first channel.
   b. Press and hold down the Ctrl key.
   c. Select every channel you want to add to the group.
   d. Release the Ctrl key.

   You do not need a listener.

Starting the two queue managers and their channel groups

When you have created the VENUS and MARS queue managers, and their channel groups, you are ready to start them. Perform the following steps:

1. Use the Advanced Controls utility to start queue manager VENUS and channel group VENUSGroup.

   If this is successful, a check mark appears beside the channel group icon in the window of the Advanced Controls utility.

   This queue manager is now ready to respond to incoming channel-connect requests.

2. Use the Advanced Controls utility to start queue manager MARS and channel group MARSGroup.

   This sends channel-connect requests to the VENUS queue manager. If this is successful, a check mark appears beside the channel group icon in the window of the Advanced Controls utility.

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 Advanced Controls utility.

- There is a TCP/IP problem. Restart the workstation and try again. It is not necessary to re-create the queue manager.

Now you can use the sample programs to verify that your queue managers are operating correctly. Proceed to “Verifying the configuration” on page 73.

Verifying the configuration

When you have set up the two queue managers so that they can send messages to each other, use the sample programs supplied with MQSeries for Windows to verify that they are set up correctly:

The Putting Messages sample
Run the Putting Messages sample on MARS to put messages on the local queue owned by VENUS.

The Getting Messages sample
Run the Getting Messages sample on VENUS to get messages from the same queue.

Putting messages on a queue

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. Perform the following steps:

1. Start the Putting Messages sample by double-clicking on its icon in the MQSeries for Windows program group.

   The sample automatically connects to MARS because this is the active queue manager.

2. In the **Queue** field of the Putting Messages Sample window, type the following name in uppercase letters:

   ```
   SAMPLE.MARS.REMOTE
   ```

   Notes:

   a. The queue SAMPLE.MARS.REMOTE is defined in the file MARS.TST. It is a remote queue.

   b. 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 7 on page 69 shows these queues.

3. Select the **Open** push button to open the queue, and check that the reason code displayed in the **API Return Code** field is zero.

   If the reason code is not zero, this means the program cannot open the queue. The reason codes are listed in Appendix E, “Reason codes” on page 175. You are most likely to see the following reason codes:

   2059 This means the queue manager is not running. If this happens, close the sample by selecting **Exit** from the File menu. Start the queue manager using the Standard Controls utility or the Advanced Controls utility before you retry 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. Select the **Put** push button. This puts the message on the queue and the message text also appears in the **Log list** box.
6. Check that the reason code displayed in the **API Return Code** field is zero.
7. Repeat steps 4 and 5 to put other messages on the queue.

You can leave the sample running so you can continue putting messages after you have used the Getting Messages sample program to remove some from the queue.

When you have finished using the Putting Messages sample program:

1. Select the **Close** push button. This closes the queue and disconnects the sample from the queue manager.
2. Close the window of the sample program.

Getting messages from the queue

On VENUS, use the Getting Messages sample program to get messages from the queue named SAMPLE.VENUS.LOCAL. Perform the following steps:

1. Start the Getting Messages sample by double-clicking on its icon in the MQSeries for Windows program group.

   The sample automatically connects to VENUS because this is the active queue manager.
2. 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.
3. Select the **Open** push button to open the queue, and check that the reason code displayed in the **API Return Code** field is zero.

   If the reason code is not zero, this means the program cannot open the queue. This is likely to be for one of the reasons explained in “Putting messages on a queue” on page 73.
4. Select the **Get** push button. This retrieves the oldest message from the queue and displays it in the **Data list** box. The **Length** field shows the length of the message data.

5. Check that the reason code displayed in the **API Return Code** field is zero.

   You are most likely to see the following reason codes:

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.
2219	This means the sample is waiting for a message to arrive. If there are no messages on the queue, the sample waits for 15 seconds, then it returns reason code 2033. If you select the **Get** push button within this waiting period, code 2219 tells you that the program is busy.

   The reason codes are listed in Appendix E, "Reason codes" on page 175.

6. You can keep repeating step 4 to get any other messages from the queue.

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. Select the **Close** push button. This closes the queue and disconnects the sample from the queue manager.
2. Close the window of the sample program.

Notes:

1. The messages are retrieved in the same order in which they were put on the queue.
2. The put and get operations are independent, so the put and get programs can operate at the same time, or one at a time.
3. Getting the messages from the queue removes them from the queue.

Sending messages the other way

You should also test that you can send messages the other way, that is, from VENUS to MARS. The configuration tasks you performed in "Setting up the two queue managers on Windows" on page 67 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 Getting Messages sample on MARS.
4. Get the messages from the queue named SAMPLE.MARS.LOCAL.

When one of your queue managers 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 69.
5. Run the sample programs as described in “Verifying the configuration” on page 73.

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 69.
3. Set up the VENUS queue manager on Windows by following the procedure described in “Setting up VENUS” on page 68.
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 “Verifying the configuration” on page 73.

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 may want to try

When you are satisfied that your queue managers are operating as you would expect, you may 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 both use the same queue, but note that the Getting Messages 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.

Two queue managers

Part 4. Supporting users of MQSeries for Windows

Chapter 9. Administering a queue manager

If you have to support other users who are running MQSeries applications on Windows workstations, MQSeries for Windows provides facilities to help you. This chapter introduces those facilities and provides information on the Advanced Controls utility.

The administration facilities

The administration facilities that MQSeries for Windows provides are:

The Advanced Controls utility
This utility is intended for developers and administrators who may need to change quickly the configuration of an MQSeries for Windows queue manager on a workstation. You may need to do this for development, testing, or trouble-shooting purposes. The Advanced Controls utility provides windows in which you can see the current values of attributes, and that show you the values you can use. It is described in “The Advanced Controls utility” on page 82.

The MQSC Commands utility
This utility allows you to change an MQSeries for Windows queue manager by typing MQSC commands and by running an MQSC command file. It is described in Chapter 10, “Using MQSC commands” on page 85.

The Create and Go utility
This utility allows you to provide a customized initialization file to your users so that they can create a working queue manager without having to set it up themselves. It is described in Chapter 11, “Creating an INI file for the Create and Go utility” on page 95.

Transport links
You can provide a transport link program that is invoked when you use a dial-up device (such as a modem) for your communications. You could use this program to control the cost of such a connection.

Transport links are described in Chapter 12, “Working with transport links” on page 113.

The Service Information utility
This utility provides information that may help you to solve a problem with an MQSeries for Windows queue manager. It provides information about the version of MQSeries for Windows that is installed and about the drive on which it is installed.

The utility is described in “The Service Information utility” on page 119.

The Service Trace utility
This utility allows you to collect trace information to help you to isolate a problem. It is intended primarily for use under the guidance of IBM Service personnel. It is described in “The Service Trace utility” on page 119.

In addition, Chapter 14, “Diagnosing problems” on page 123 gives you help in solving some common problems.

The Advanced Controls utility

The Advanced Controls utility performs the same functions as the Standard Controls utility (described in “Monitoring queue managers and queues using the Standard Controls utility” on page 46), but in addition, you can:

- Change the attributes of a queue manager
- Change the attributes of queues
- Change the attributes of channel groups
- Add channels to, and delete them from, channel groups
- Add the listener to, or remove it from, a channel group
- Change the attributes of channels
- Change the attributes of transport links
- Start the MQSC Commands utility

Changing an attribute

The Advanced Controls utility provides similar windows to those of the Standard Controls utility so, if you have followed the instructions in Chapter 6, “Working with a queue manager” on page 43, you should be familiar with them. The main difference is that, after you have viewed the value of an attribute, you can select from a list the value you want to set it to. You can also reset the attribute to the value it had when you displayed the attributes.

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

1. Start the Advanced Controls utility by clicking on its icon in the MQSeries for Windows program group.
2. Go to the Queue Managers view of the Advanced Controls utility and select the active queue manager.
3. Select **Queues** from the Selected menu.

   The resulting Queues for Queue Manager window shows all the queues that the active queue manager owns.
4. Select the queue you want to work with, then select **Attributes** from the Selected menu.
5. In the resulting window, select the InhibitPut attribute. The current value of this attribute is displayed.
6. Select the **Change** push button.
7. Change the value of the attribute by selecting the value PUT_INHIBITED from the list box.
8. Select the **OK** push button.

If you want more information on the attributes, see the online help.

Issuing MQSC commands

To issue MQSC commands, either singly or from a command file, use the MQSC Commands utility. You can start this utility without leaving the Advanced Controls utility. Select **Redefine** from the Selected menu, then use the MQSC Commands utility as described in Chapter 10, "Using MQSC commands" on page 85.

Advanced Controls utility

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 describes:

- "How to issue MQSC commands"
- "The MQSC Commands utility" on page 86
- "Writing MQSC command files" on page 89

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 Appendix C, "MQSC commands supported by MQSeries for Windows" on page 169 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*.

MQSeries for Windows does not support the MQSeries Command Server, so you cannot run MQSC commands from programs (known as Programmable Command Formats, or PCFs) and you cannot issue commands on one workstation to run them on another (known as remote administration).

How to issue MQSC commands

You can type MQSC commands one at a time, or you can save many commands in a file and run that file. MQSeries for Windows provides the MQSC Commands utility to make both of these operations easy. This utility performs and extends the functions of the RUNMQSC command that other MQSeries products provide. You can also run MQSC command files using some of the other MQSeries for Windows utilities. Table 4 on page 86 shows which utilities you can use.

MQSC Commands utility

Table 4. How you can issue MQSC commands in MQSeries for Windows

	To type MQSC commands	**To run an MQSC command file**	**To use the DISPLAY command**	**To use the DEFINE command**
Create Components utility	No	Yes	No	(1)
Create and Go utility	No	Yes	No	No
MQSC Commands utility	Yes	Yes	No	Yes
Standard Controls utility	No	No	(2)	No
Advanced Controls utility	(3)	No	(2)	No

Notes:

1. You cannot type the DEFINE command in this utility, but you can use its windows to create components.
2. You cannot type the DISPLAY command in this utility, but you can use its windows to display the attributes of objects.
3. You cannot type MQSC commands in this utility, but you can use its windows to change the attributes of existing MQSeries objects.

Specifying MQSeries object names

In the examples showing MQSC commands in this chapter, some of the objects have long names. This is to help you identify the type of object you are dealing with.

When you are issuing MQSC commands, you need specify only the local name of the object on which you are operating. In the examples, 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.

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.

The MQSC Commands utility

The MQSC Commands utility allows you to type MQSC commands and to run MQSC command files. 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. Any errors are reported and can be viewed from the utility. 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 you can load it next time you use the utility (provided you have not added something else to the clipboard or restarted your workstation).

Recall commands
The utility stores the last 10 successful commands. You can recall them using the PageUp and PageDown keys, then rerun any of them.

Log commands in a window
The utility logs in its Result window 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
The utility 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. The utility creates an empty log every time it starts.

Save user preferences
Each time you end the utility, it saves your preferences (such as window size and position) and 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*. You can open this book without leaving the utility.

Differences from other platforms

The main differences between the MQSC Commands utility and the RUNMQSC utility provided by other MQSeries products are:

- MQSeries for Windows does not support the DISPLAY command. To view the attributes of MQSeries objects, use the Standard Controls utility or the Advanced Controls utility.
- 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 89.

MQSC Commands utility

Starting the MQSC Commands utility

You can start the MQSC Commands utility in two ways:

- Start either the Standard Controls utility or the Advanced Controls utility, then start the queue manager you want to work with. Then double click on the icon of the MQSC Commands utility in the MQSeries for Windows program group.
- Start the Advanced Controls utility and start the queue manager you want to work with. Then select **Redefine...** from the Selected menu in the Advanced Controls utility.

Note: You cannot start the MQSC Commands utility unless either the Standard Controls utility or the Advanced Controls utility is running.

When the MQSC Commands utility starts, a message box warns you that any commands you issue will affect the active queue manager. The name of the active queue manager is shown in the title bar of the MQSC Commands utility.

If the MQSC Commands utility cannot start, MQSeries for Windows displays the following error message:

```
A queue manager is not running. (AMQ3581).
```

If this happens, start a queue manager from either the Standard Controls utility or the Advanced Controls utility, then try again.

Issuing MQSC commands from the utility

You can issue MQSC commands either interactively (by typing them in a field of the MQSC Commands utility window) or by saving them in an MQSC command file and running that file.

Issuing commands interactively

To issue a command interactively, type it in the Command window of the MQSC Commands utility. 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 Result window.

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

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 a command file, select **Run MQSC File** from the File menu. This displays a window that allows you to select the file you want to run.

You can view MQSC files by selecting the **View** push button. This loads the file into the Windows Notepad program. As well as viewing, Notepad allows limited editing.

When you have chosen the file you want to use, select the **OK** push button. This runs the file against the active queue manager. If any errors or warnings occur, the utility provides an option to view the log file. A message indicating the success or failure of the command is displayed in the Result window.

Stopping the MQSC Commands utility

You can stop the MQSC Commands utility either by selecting **Exit** from the File menu, or by selecting **Close** from the system menu. When you close the MQSC Commands utility, the message box displayed by the Standard Controls utility or the Advanced Controls utility is removed and the window is refreshed.

Note: You cannot close the Standard Controls utility or the Advanced Controls utility while the MQSC Commands utility is running.

Writing MQSC command files

MQSC commands are written in ASCII text. By convention they have a file-name extension of TST.

Figure 8 is an extract from the MQSC command file VENUS.TST that is supplied with MQSeries for Windows in the \MQW\SAMPLES directory. This extract shows the command DEFINE QLOCAL and its attributes.

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

Figure 8. An extract from VENUS.TST

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 5 on page 90 shows differences you should note.

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

MQSC command files

Table 5. The format of MQSC command files

On MQSeries for Windows	On other MQSeries platforms
Lowercase characters are ***not*** changed to uppercase. MQSeries for Windows uses the characters you type, regardless of whether or not they are enclosed in single quotation marks.	Lowercase characters not enclosed in quotation marks are changed to uppercase.
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.
A string that contains the following special characters ***must*** be enclosed in single quotation marks: Period (.) Forward slash (/) Underscore (_) Percent (%) For example: `DEFINE QLOCAL ('TEST.QUEUE')`	A string that contains the following special characters does not need to be enclosed in single quotation marks: Period (.) Forward slash (/) Underscore (_) Percent (%)
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 C:\MQW\QMGRS\TEST\MQSC.LOG (if you installed MQSeries for Windows in directory C:\MQW). If the name of the queue manager is more than 8 characters long, MQSeries for Windows creates a directory whose name is based on those 8 characters. The name of the queue manager appears near the top of the log file.

To look at the contents of the log file, edit it using any editor.

If you get an error from an MQSC command that is run when you are creating a queue manager, you are prompted to view the MQSC log. If there are errors in the syntax of the command, the message in the log file tells you what is wrong.

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, there are more errors following those first reported.

If you need more information about these errors, see Appendix F, “Error messages” on page 179.

If a command fails even when its syntax is correct, this means that the queue manager was unable to run the command. If this happens, the command returns a code that shows the reason for its failure; Appendix E, “Reason codes” on page 175 lists these codes.

Examples of MQSC command files

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

- Figure 9 on page 92 shows the file MARS.TST
- Figure 10 on page 93 shows the file VENUS.TST

Both of these files are supplied in directory \MQW\SAMPLES. They define the queues and channels you use if you follow the procedure described in Chapter 8, “Setting up and verifying two queue managers” on page 67.

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 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 place
* its messages.
* The sample application should get messages from this queue.
DEFINE QLOCAL('SAMPLE.MARS.LOCAL') REPLACE +
       DESCR('Local queue') +
       DEFPSIST(YES)  +
       SHARE
```

Figure 9. 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 10. The supplied file VENUS.TST

MQSC command files

Chapter 11. Creating an INI file for the Create and Go utility

This chapter describes how to create an initialization (INI) file for use with the Create and Go utility. The user of an MQSeries for Windows application can use this utility to create and configure automatically the MQSeries components they need on their workstation.

Note: This chapter is for the MQSeries administrator who wants to create system definitions that application users can run using the Create and Go utility on their workstations. ***If you want to run the Create and Go utility yourself, you do not need to read this chapter. Instead, see "Creating components using the Create and Go utility" on page 44.***

This chapter describes:

- "The format of the INI file" on page 96
- "Processing the INI file" on page 99
- "Defining a queue manager" on page 100
- "Defining a channel group" on page 104
- "Defining a transport link" on page 107
- "Defining controls" on page 110

When a user runs the Create and Go utility, the utility reads an initialization file that contains definitions of the MQSeries components the user needs on their workstation before they can run your application. The utility creates those MQSeries components, so the user does not have to type values into the Create Components utility to create their own components before they can run their applications. The utility can also either add one of the MQSeries for Windows utilities to the user's Windows StartUp group or it can start one of the utilities.

When you have created your own INI file to define the components needed for your users' applications, you can replace the default INI file on the installation diskettes you give to your users. All the users need to do is to run the installation program on the diskettes, then run the Create and Go utility to create the MQSeries objects they need. The Create and Go utility remembers the date, time, and size of the INI file. If they run the Create and Go utility again and these values are the same as the previous time, they are asked whether or not they want to proceed. If they do proceed, the Create and Go utility re-creates the components defined in the INI file only if the file specifies that the existing components should be replaced.

The names of queue managers must be unique on a workstation, so you must tailor the INI file for a particular user. However, you can create the INI file in such a way that the Create and Go utility prompts the user to type the name of their own queue manager.

Note: Chapter 5, "Verifying your installation and configuration" on page 35 uses the default INI file to create a sample queue manager you can use to test an installation. The procedures in that chapter will not work if you replace the default INI file.

Create and Go utility

You can create or change an INI file using any editor. Also, you can use the Create Components utility to add to an INI file, or to replace the file.

The format of the INI file

The Create and Go utility uses only the INI file named CREATEMQ.INI, and this file must be in the \MQW directory. To use another INI file, you must first rename it to CREATEMQ.INI and copy it to this directory. A file of this name is supplied on the MQSeries for Windows installation diskettes and the installation process puts the file into the MQW directory. For a description of this file, see “Example INI files” on page 98.

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

```
   [component name]
   ComponentType=type
   keyword=value

 ⋮
   keyword=value
```

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

Note: Always take a backup copy of an INI file before you edit it.

Component names

The component names must be of the form [Component_*n*], where *n* is an integer. You must enclose the component names in square brackets [ ].

There must be a `Component_1` section in the INI file, but the sections do not have to be in numeric order. The Create and Go utility processes the sections in numeric order and stops when the next number in the sequence is missing.

For example, consider this INI file:

```
    [Component_1]
    ComponentType=QueueManager
  ⋮

    [Component_3]
    ComponentType=TransportLink
  ⋮

    [Component_2]
    ComponentType=ChannelGroup
  ⋮
```

When the Create and Go utility processes this file, it first creates the queue manager, then the channel group, then the transport link.

Now consider this INI file:

```
    [Component_1]
    ComponentType=QueueManager
  ⋮

    [Component_5]
    ComponentType=TransportLink
  ⋮

    [Component_2]
    ComponentType=ChannelGroup
  ⋮

    [Component_3]
    ComponentType=ChannelGroup
  ⋮
```

When the Create and Go utility processes this file, it first creates the queue manager, then the first channel group, then the second channel group, then stops. It does not create the transport link because there is a gap in the sequence of component definitions.

Create and Go utility

Component types

You can define sections using the following component types:

QueueManager
Defines the properties of the queue manager.

ChannelGroup
Defines the properties of a channel group.

TransportLink
Defines the properties of a transport link.

Controls
Defines when the MQSeries for Windows utilities are started.

Note: To run an MQSeries 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 INI file.

Keywords

The *keyword*=*value* statements define the attributes of the component. The keywords you can use are described in this chapter; they must be followed immediately by an equals (=) sign.

This chapter describes the keywords you can use within each section.

Example INI files

The following example is the supplied file CREATEMQ.INI. It creates the objects for the sample programs to use.

```
[Component_1]
ComponentType=QueueManager
Name=SAMPLE_QM
Description=Queue Manager for testing the samples
LoadSamplesMQSC=yes
Autostart=yes
Replace=yes

[Controls]
RunOnCompletion=StandardControls
```

The following example is the supplied sample file, MARS.INI:

```
[Component_1]
ComponentType=QueueManager
Name=MARS
Description=Queue manager to communicate with Venus
LoadUserMQSC_1=C:\MQW\SAMPLES\MARS.TST
Autostart=yes
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
Autostart=yes
Replace=yes

[Controls]
RunOnCompletion=AdvancedControls
```

Processing the INI file

When the Create and Go utility processes the INI file, it writes any warning or error messages to a log file named CREATEMQ.LOG in the \MQW directory. This file is overwritten each time the Create and Go utility is started. If any errors or warnings occur during the processing of the INI file, the Create and Go utility displays a window which gives the following choices to the user:

- View the log file
- End the Create and Go utility (it deletes all components it has created, and restores any components it has replaced to their former state)
- Continue with the Create and Go utility

Defining a queue manager

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

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

Note: To run an MQSeries 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 INI file.

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 must be unique on the workstation. It can have a maximum of 48 characters.
	
	If you use a nonvalid character, the Create and Go utility replaces it with a period (.) and logs a warning.
	
	To prompt the user to type the name of their queue manager, specify the following in the INI file:
	
	`Name=?`
	
	You can control the wording of the window that the user sees by using the NamePrompt and NameInformationText keywords.
	
	If you do not specify a name, the Create and Go utility 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. This is valid only when you specify `Name=?` in the Name keyword. You can use a maximum of 100 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. This is valid only when you specify `Name=?` in the Name keyword. You can use a maximum of 200 characters. The default text is: `Type the name of the new queue manager.`
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 or not to load the MQSC command file that is needed to run the MQSeries for Windows sample programs. 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. 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 the Create and Go utility to create when it creates the queue manager. If the Create and Go utility 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, the Create and Go utility puts an entry in its log file, and that entry refers you to an MQSC log file.
Autostart=	Specify whether or not the queue manager is started automatically when the Standard Controls utility or Advanced Controls utility is started. Specify either `yes` or `no`. The default value is `yes`. **Note:** Only one queue manager can be started automatically. If you specify `Autostart=yes` for more than one queue manager, the last one you define is started automatically.

Replace=	If a queue manager of this name already exists, specify whether or not 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.

Example queue manager components

The first example defines a queue manager named MY_QUEUE_MANAGER using the default values:

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

This creates a queue manager that has the default queues and channels. It is started automatically when the Standard Controls utility or the Advanced Controls utility starts, but if there is already a queue manager of that name running on the workstation, it is not replaced. The queue manager has no description.

The second example is from MARS.INI, the sample file that creates one of the queue managers you can use to verify you have correctly configured two communicating queue managers (see Chapter 8, “Setting up and verifying two queue managers” on page 67):

```
[Component_1]
ComponentType=QueueManager
Name=MARS
Description=Queue manager to communicate with VENUS
LoadUserMQSC_1=C:\MQW\SAMPLES\MARS.TST
Autostart=yes
Replace=yes
```

When the queue manager in the second example is created, the file C:\MQW\SAMPLES\MARS.TST is run, and this file defines the objects the queue manager needs when it communicates with the queue manager named VENUS. The queue manager is started automatically when either the Standard Controls utility or the Advanced Controls utility is started, and if there is already a queue manager named MARS on the workstation, it is replaced.

The third example defines a queue manager using the name that the user is prompted to supply:

```
[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
Autostart=yes
Replace=yes
```

When the queue manager in the third example is created, the Create and Go utility 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.

Defining a channel group

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

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

You can define zero or more channel group sections.

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, the Create and Go utility replaces it with a period (.) and logs a warning.
	
	If you do not specify a name, the Create and Go utility 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.
QueueManagerName=	Specify the name of the queue manager to which the channel group is to belong. If the queue manager you specify does not exist, the Create and Go utility logs an error. If you do not specify the name of a queue manager, and the Create and Go utility 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, the Create and Go utility logs an error.
StartListener=	Specify whether or not a listener is started when the channel group is started. Specify either `yes` or `no`. The default value is `no`.

AllUserChannels=	Specify whether or not to include in the channel group all the user-defined channels that are on the owning queue manager. Specify either `yes` or `no`. The default value is `yes`. **Note:** If you use the Create Components utility to create your INI file, `AllUserChannels=yes` is added to your file. If you do not want to include all the channels, you must edit the INI file like this: • Change the keyword to `AllUserChannels=no` • Add a `Channel_`*n* keyword for each channel you want to create If you specify `AllUserChannels=no`, you must use Channel_*n* keywords to specify one or more channel names.
Channel_*n*=	Specify the name of a channel to be included in the channel group. You can specify this keyword many times, substituting successive integers for *n*, starting with 1. The entries are read in numeric order. If the Create and Go utility cannot find the specified channel, it logs an error and continues processing the next component. If you specify `AllUserChannels=yes`, the Create and Go utility ignores all `Channel_`*n* keywords.
Autostart=	Specify whether or not the channel group is started automatically when the Standard Controls utility or Advanced Controls utility is started. Specify either `yes` or `no`. The default value is `yes`. **Note:** Only one channel group can be started automatically. If you specify `Autostart=yes` for more than one channel group, the last one you define is started automatically.
Replace=	If a channel group of this name already exists, specify whether or not to replace it with this one. Specify either `yes` or `no`. The default value is `no`.

Example channel group components

The first example defines a channel group named MY_CHANNEL_GROUP on the queue manager named MY_QUEUE_MANAGER, using all the default values:

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

This creates a channel group that includes all the user-defined channels. It is started automatically when the Standard Controls utility or the Advanced Controls utility starts, but if there is already a channel group of that name defined on the workstation, it is not replaced. The channel group has no description.

The second example is from MARS.INI, the sample file that creates one of the channel group you can use to verify you have correctly configured two communicating queue managers (see Chapter 8, “Setting up and verifying two queue managers” on page 67):

```
[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
Autostart=yes
Replace=yes
```

When the channel group in the second example is created, it includes only two channels (named MARS.TO.VENUS and VENUS.TO.MARS). The channel group is started automatically when either the Standard Controls utility or the Advanced Controls utility is started, and if there is already a channel group named MARSGroup on the workstation, it is replaced.

Defining a transport link

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

```
[Component_n]
ComponentType=TransportLink
```

You can define zero or more transport link sections.

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

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

Name=	Specify the name of the transport link you want to create.
	You can use a maximum of 48 characters. If you use a nonvalid character, the Create and Go utility replaces it with a period (.) and logs a warning.
	If you do not specify a name, the Create and Go utility selects the first name that has not already been used from the following list:
	`TransportLink`
	`TransportLink_1`
	`TransportLink_2`
	⋮
	`TransportLink_n`

Description=	Describe the transport link. Use this keyword to add a text description of the transport link. You can use a maximum of 64 characters. The default value is all blanks.
StartFilename=	Specify the name and full path of the program that is called when the transport link is started. If you do not specify a file name, the Create and Go utility uses the file name C:\STRTLINK.BAT. If the Create and Go utility cannot find the file you specify, it logs a warning and continues processing the next component.

StartTimeout=	Specify the time (in seconds) that the Standard Controls utility or the Advanced Controls utility wait after calling the program you specify in the StartFilename keyword. This timeout is to allow time for the transport link to start. After this time, the Standard Controls utility or the Advanced Controls utility reads the status file (STRTLINK.DAT) to see if the link has started. The default timeout is 60 seconds.
StopFilename=	Specify the name and full path of the program that is called when the transport link is stopped. If you do not specify a file name, the Create and Go utility uses the file name C:\STOPLINK.BAT. If the Create and Go utility cannot find the file you specify, it logs a warning and continues processing the next component.
StopTimeout=	Specify the time (in seconds) that the Standard Controls utility or the Advanced Controls utility wait after calling the program you specify in the StopFilename keyword. This timeout is to allow time for the transport link to stop. After this time, the Standard Controls utility or the Advanced Controls utility reads the status file (STOPLINK.DAT) to see if the link has stopped. The default timeout is 30 seconds.
Replace=	If a transport link of this name already exists, specify whether or not to replace it with this one. Specify either `yes` or `no`. The default value is `no`.

Example transport link components

The first example defines a transport link named PLANETS_TL, using all the default values:

```
[Component_3]
ComponentType=TransportLink
Name=PLANETS_TL
```

This creates a transport link that calls the default programs when it starts and stops, and allows the default times for the programs to run. If there is already a transport link of that name defined on the workstation, it is not replaced. The transport link has no description.

The second example defines a transport link using other values of the keywords:

```
[Component_3]
ComponentType=TransportLink
Name=JUPITER_TL
Description=Transport link for Jupiter
StartFilename=STRTJUP.BAT
StartTimeout=30
StopFilename=STOPJUP.BAT
StopTimeout=20
Replace=yes
```

Defining controls

To define options for running the Standard Controls utility or the Advanced Controls utility, use a section that starts with the line:

```
[Controls]
```

The Controls section is optional, but if you do use it, you can define only one section.

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

Use this keyword	***To do this***
AutostartControls=	Specify which utility is added to the Windows StartUp group. The utility you specify is started automatically when Windows is started. You can specify only one of the following values: `None` — Add nothing to the StartUp group. `StandardControls` — Add the Standard Controls utility to the StartUp group. `AdvancedControls` — Add the Advanced Controls utility to the StartUp group. If you specify anything else, the Create and Go utility logs a warning and adds nothing to the StartUp group. The default value is `None`. If you specify `None`, or the Create and Go utility uses the default value, and either of the utilities is already in the StartUp group, it is removed.
RunOnCompletion=	Specify which utility to run when the Create and Go utility has completed processing. You can specify only one of the following values: `None` — Run nothing. `StandardControls` — Run the Standard Controls utility. `AdvancedControls` — Run the Advanced Controls utility. If you specify anything else, the Create and Go utility logs a warning and runs nothing. The default value is `None`.

Examples of Control sections

The first example defines a control section that automatically starts the Standard Controls utility every time Windows is started:

```
[Controls]
AutostartControls=StandardControls
```

The second example defines a control section that automatically runs the Advanced Controls utility and adds the Standard Controls utility to the StartUp group:

```
[Controls]
RunOnCompletion=AdvancedControls
AutostartControls=StandardControls
```

Create and Go utility

Chapter 12. Working with transport links

This chapter describes how to set up and use transport links, which are named links to other queue managers. When you are communicating with another queue manager using a LAN connection, you need only a channel group. When you are communicating using a dial-up device (such as a modem), you can use a transport link in addition to a channel group.

The users of MQSeries for Windows can start and stop the transport link whenever they like, so they can use it to control the duration (and hence the cost) of their dial-up connection. When you define a transport link, you specify the names of two programs: one that runs when you start the transport link, another that runs when you stop the link. These programs can perform whatever functions you like, but they must at least start and stop the dial-up device.

This chapter has the following sections:

Transport link programs

If you use a transport link, you need to provide two programs: one to start the dial-up device and one to stop it. You must write these programs using the commands provided by the device. In the definition of a transport link (see "Creating transport links" on page 114), you must specify the names of these programs.

MQSeries for Windows provides two sample files. They are named STRTLINK.BAT and STOPLINK.BAT, and they are supplied in the \MQW\SAMPLES directory. The sample files contain comments that give guidance on how to write programs that control transport links.

In the definition of the transport link, you must specify timeout values that define how long MQSeries for Windows waits for your programs to run. Within these times, these programs must write to status files to show whether or not they have completed successfully. The default status files are named STRTLINK.DAT and STOPLINK.DAT. If the program completes successfully, it must write the character `0` to the status file; if the program is unsuccessful, it must write any other integer.

Your programs are called by the Standard Controls utility or the Advanced Controls utility: one when you start the transport link, the other when you stop it. The names of the status files are passed as parameters to the programs.

Creating transport links

The procedure for creating a transport link is similar to that for creating a queue. When you have started the Create Components utility, select the **Transport Link** push button in the Create Components window, then complete the fields in the Create Transport Link window. You must specify:

- The name of the queue manager for which you want to create a transport link
- The name of the transport link you want to create
- The names of the programs you want to run when the transport link starts and stops (examples of these are supplied in the \MQW\SAMPLES directory)
- The names of the status files that you want the programs to write to (examples of these are supplied in the \MQW\SAMPLES directory)
- The timeout values that specify how long MQSeries for Windows allows for the programs to run

When you have created your transport link, remember to close the Create Components utility.

Monitoring transport links

The Standard Controls utility allows you:

- View the settings of a transport link, using the Transport Links view
- View the status of a transport link in relation to other components, using the Connection Monitor view
- Start a transport link
- Stop a transport link
- View the attributes of a transport link

If you want to change any attributes, you must use the Advanced Controls utility.

The Transport Links view

The Transport Links view of the Standard Controls utility shows the status of all the transport links owned by the active queue manager. This is what the icons in the status window mean:

There is one **Transport Link** icon for each transport link you have created. If the icon contains a status indicator (a check mark or an exclamation mark), the transport link is active (that is, it is running). If there is no status indicator, the transport link is stopped.

The name to the right of this icon is the name of the transport link.

The **Monitor** icon identifies the transport link whose details are displayed when you move to the Connection Monitor view.

To monitor a different transport link, select that transport link in the transport links view, then select **Monitor** from the Selected menu.

The **Autostart** icon identifies the transport link that is started automatically when you restart the Standard Controls utility.

To automatically start a different transport link, select that transport link in the transport links view, then select **Autostart** from the Selected menu.

Monitoring transport links

The Connection Monitor view

The Connection Monitor view shows the current status of all the components associated with the selected queue manager. When we saw it in “The Connection Monitor view” on page 62, this view contained icons for the leaf node, the queue manager, the channel group, and the server. But when you use transport links to communicate using a dial-up device, this view shows the transport links as well. This is what the icons in the status window mean:

The **Leaf Node** icon represents your workstation—this is your end of the connection.

The **Queue Manager** icon represents the queue manager you are using. The icon shows the status of the queue manager.

The name to the right of this icon is the name of the queue manager.

The **Channel Group** icon represents the channel group you are using. The icon also shows the status of the channel group.

The name to the right of this icon is the name of the channel group.

The **Transport Link** icon represents the transport link you are using. If the icon contains a check mark, the transport link is active. If there is no check mark, the transport link is stopped.

The name to the right of this icon is the name of the transport link.

The **Server** icon represents the server end of the connection. This is the queue manager with which you are communicating.

For an MQSeries application (represented by the leaf node icon at the top of the status window) to be able to communicate with a remote queue manager (represented by the server icon at the bottom of the Connection Monitor view), all the components between them must be in the active state. So for a successful connection, all the icons between the leaf node and the server must contain check marks. This view can give a quick visual confirmation that communication is possible.

Monitoring transport links

Starting a transport link

In MQSeries for Windows, only one transport link can be active (that is, running) at a time. If you try to start a transport link when another one is active, you are prompted to confirm that you want to stop the active one.

To start a transport link, select its name in the transport links view of the Standard Controls utility. Double click on the name, or select **Start** from the Selected menu. Wait until the transport link starts and the program you have specified starts the communications device. When it does, a check mark is added to the transport link icon. You can also start a transport link using the same method in the Connection Monitor view.

Stopping a transport link

To stop a transport link, select its name. Double-click on it, or select **Stop** from the Selected menu.

You do not have to stop a transport link before you start another one; when you start another transport link, you are prompted to confirm that you want to stop the active one.

When you stop a transport link, the program you have specified is called. This program could stop the dial-up device.

When you stop a transport link, the active channel group is quiesced; that is, it is put into a state from which it can automatically restart if you restart the transport link.

Note: The transport link stops when you close the Standard Controls utility.

Viewing the attributes of a transport link

You can view the attributes of any transport link at any time:

1. Go to the Transport Links view and select the transport link you want to view.
2. Select **Attributes...** from the Selected menu.

   In the resulting window, you can see the names of the programs (and their timeout values) associated with the transport link.

For more information on these attributes, see the online help. If you need to change any of the attributes, you must use the Advanced Controls utility.

Deleting transport links

Deleting a transport link

When you no longer need to use a transport link again, delete it using the Delete Components utility.

To delete a transport link:

1. Close any MQSeries for Windows utilities that are running.
2. Click on the icon of the Delete Components utility in the MQSeries for Windows program group and select the **Transport Link** push button.

   This displays the Delete Transport Link window.
3. Select from the list the name of the queue manager that owns the transport link you want to delete.
4. Select from the list the transport link you want to delete.

   When you select a transport link, its description is displayed. The description may help you select the correct transport link to delete.

   Note: You can delete only one transport link at a time.
5. Click on the **OK** push button to start the delete process.

   You are prompted to confirm that you want to delete the transport link.

When the transport link has been successfully deleted, you can reuse its name when you create another transport link.

If you uninstall the product, all transport links are automatically deleted.

Chapter 13. Service tools

This chapter describes the service tools that MQSeries for Windows provides:

- The Service Information utility
- The Service Trace utility

The Service Information utility

The Service Information utility displays information that may help you to solve the problems of users of an MQSeries for Windows application. The information it displays includes:

- The name of the directory in which MQSeries for Windows is installed
- The amount of free disk space remaining on the drive on which MQSeries for Windows is installed
- The name of the national language version of MQSeries for Windows that is installed
- The release levels of the MQSeries for Windows files, so you can determine if any maintenance fixes have been applied

In addition, the utility writes the release levels of the MQSeries for Windows files to a file named AMQLEVLW.LOG in the \MQW\DATA directory.

To start the Service Information utility, click on its icon in the MQSeries for Windows program group.

The Service Trace utility

To trace the operation of an MQSeries for Windows application, use the Service Trace utility. It 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.

Service trace

Starting tracing

Before you use the Service Trace utility

Service Trace is active only for threads that are started ***after*** the utility itself. Before using trace, ***close down all MQSeries for Windows applications*** on the workstation on which you are going to use the trace. If you do not do this, the trace output will be incomplete.

To start tracing, click on the icon for the Service Trace utility in the MQSeries for Windows program group.

Controlling the trace output

You can choose to direct the trace output either to the window of the Service Trace utility, to a file, or to both. You do this by selecting an option from the File menu in the window of the Service Trace utility. At any time, you can change where the output is sent. The default action is to send trace output to a file only.

Also, you can choose which trace points to trace, but you should do this only under the guidance of IBM service personnel.

Tracing to a file

If you send trace output to 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 C:\MQW\DATA directory.

To send the trace output to a different file, you must supply the file name, and the path to it, after the start trace command. For example, to send trace output to the file NEWTRACE.LOG in the C:\MTRACE directory:

1. In the Windows Program Manager, select **Run...** from the File menu.
2. In the Run window, type the following:

```
AMQTRACW C:\MTRACE\NEWTRACE.LOG
```

The new file name is displayed in the title bar of the Service Trace utility.

Each time you start the Service Trace utility, the previous contents of the trace file are overwritten. To save the trace, rename the trace output file before you restart the Service Trace utility.

Controlling autosave when tracing to a file

If you send trace output to a file, you can control how often the trace output is written to the disk file. This can be one of:

- After each line of trace output is generated
- After approximately every 50 trace records (these are 16 KB blocks)

To write the output after every line, select **Autosave every line** from the Options menu.

Recommendation

The **Autosave every line** option can be extremely slow, especially during startup, and it may cause timeouts in the application to be exceeded, resulting in failure of the application. Use this option only under the direction of IBM service personnel. You are advised ***not*** to use this option in production systems.

Stopping tracing

Stop tracing either by selecting **Exit** from the File menu of the Service Trace utility, or by pressing ALT+F4 in the window of the Service Trace utility.

When you do this, all tracing is stopped and the window of the Service Trace utility is closed.

Service trace

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 covers:

- “Preliminary checks”
- “Problems with queues and queue managers” on page 124
- “Problems with channels and channel groups” on page 125
- “Problems with messages” on page 126

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 Install utility. 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 use this utility, see Chapter 5, “Verifying your installation and configuration” on page 35.

Is there enough disk space?

You may not have enough free disk space available to run MQSeries for Windows. Use the Service Information utility 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.

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 Advanced Controls utility, or use the MQSC Commands utility to run the MQSC command ALTER 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 having problems:

1. Display the attributes of each queue, using either the Standard Controls utility or the Advanced Controls utility.
2. Perform the following checks:
   - 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 that they are correct:
     - The Shareability attribute defines whether or not 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 or not 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 channels groups, you should use the Standard Controls utility or the Advanced Controls utility to check the status of the channels; for information on how to do this, see "Viewing the status of a channel" on page 64.

Are the channels working?

You can see if a channel is currently sending messages by looking at the status indicators in the Connection Monitor view of the Standard Controls utility or the Advanced Controls utility. You can get more information by checking the status of the channel. Amongst 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 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 Standard Controls utility or the Advanced Controls utility to stop the sending end of the channel.

Why does the channel group not start?

If the channel group icon shows an exclamation mark status indicator 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 does not convert between the two code pages.
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.

Diagnosing problems

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.
Message sequence error	The channel has 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 system 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.
- Error messages are displayed on the workstations at both ends of the message 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 message channel. You can:

- Check the status of the channel in the Channel Status window of the Standard Controls utility or the Advanced Controls utility
- 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 MaxMsgL 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?

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 recovery 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. Therefore, any data conversion must be done by the queue manager that sends a message to an MQSeries for Windows queue manager.

Part 5. Application programming

Chapter 15. Writing applications using the MQI on Windows

When you write an MQSeries application to run on 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 141

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, “Application programming restrictions” on page 149 to understand the differences.

Examples of existing applications
Chapter 17, “Sample programs” on page 155 describes the sample programs supplied with MQSeries for Windows.

You can write applications for MQSeries for Windows using either of the following programming languages:

- Visual C++
- Visual Basic

For a list of the compilers you can use, see “Required software” on page 13.

Using the C programming language

This section contains:

- “Considerations for the C language”
- “MQI calls in C” on page 135
- “Elementary data types in C” on page 138
- “Structure data types in C” on page 139

Considerations for the C language

This section gives information you need before you start to use the MQI in the C programming language.

The MQI in C

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 6.

Table 6. 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

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"
```

For 16-bit applications, link the programs with the library MQM16.LIB. For 32-bit applications, link the programs with the library MQM.LIB.

The include files (.H) are supplied in the \MQW\INCLUDE subdirectory (by default). The library files are included in the \MQW\LIB subdirectory (by default).

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, namely 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 MQBYTEn 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 “MQXXX_DEFAULT”, where “MQXXX” 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 MQXXX_DEFAULT macro variable to initialize the structure. This instance becomes the “model” for other instances:

   ```
   MQMD Model = {MQMD_DEFAULT};  /* declare model instance */
   ```

   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 */
   ```

[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.

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 */
```

MQCMIT

```
MQCMIT (Hconn, &CompCode, &Reason);
```

Declare the parameters as follows:

```
MQHCONN  Hconn;     /* Connection handle */
MQLONG   CompCode;  /* Completion code */
MQLONG   Reason;    /* Reason code qualifying CompCode */
```

The MQI in C

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 */
```

The MQI in C

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 7. 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;`

Structure data types in C

MQDLH–Dead-letter header

MQSeries for Windows does not support dead-letter queues, so you cannot use the MQDLH structure.

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;        /* Reserved */
  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;
```

MQTM–Trigger message

MQSeries for Windows does not support triggering, so you cannot use the MQTM structure.

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;
```

Using the Visual Basic programming language

This section contains:

- “Considerations for the Visual Basic language”
- “MQI calls in Visual Basic” on page 142
- “Elementary data types in Visual Basic” on page 145
- “Structure data types in Visual Basic” on page 146

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 8.

Table 8. Visual Basic header files

File name	Contents
CMQB3.BAS	Call declarations, data types, and named constants for the 16-bit MQI. Use this with Microsoft Visual Basic, Version 3.
CMQB4.BAS	Call declarations, data types, and named constants for both the 16-bit and 32-bit MQI. Use this with Microsoft Visual Basic, Version 4.

The form files (.BAS) are supplied in the \MQW\INCLUDE subdirectory (by default). The library files are included in the \MQW\LIB subdirectory (by default).

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

The MQI in Visual Basic

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:

```
Hconn    As Long 'Connection handle'
CompCode As Long 'Completion code'
Reason   As Long 'Reason code qualifying CompCode'
```

MQCLOSE

```
MQCLOSE (Hconn, Hobj, Options, CompCode, Reason)
```

Declare the parameters as follows:

```
Hconn    As Long 'Connection handle'
Hobj     As Long 'Object handle'
Options  As Long 'Options that control the action of MQCLOSE'
CompCode As Long 'Completion code'
Reason   As Long 'Reason code qualifying CompCode'
```

MQCMIT

```
MQCMIT (Hconn, CompCode, Reason)
```

Declare the parameters as follows:

```
Hconn    As Long 'Connection handle'
CompCode As Long 'Completion code'
Reason   As Long 'Reason code qualifying CompCode'
```

MQCONN

```
MQCONN (Name, Hconn, CompCode, Reason)
```

Declare the parameters as follows:

```
Name     As String*48 'Name of queue manager'
Hconn    As Long      'Connection handle'
CompCode As Long      'Completion code'
Reason   As Long      'Reason code qualifying CompCode'
```

MQDISC

```
MQDISC (Hconn, CompCode, Reason)
```

Declare the parameters as follows:

```
Hconn    As Long 'Connection handle'
CompCode As Long 'Completion code'
Reason   As Long 'Reason code qualifying CompCode'
```

MQGET

```
MQGET (Hconn, Hobj, MsgDesc, GetMsgOpts, BufferLength, Buffer,
      DataLength, CompCode, Reason)
```

Declare the parameters as follows:

```
Hconn        As Long   'Connection handle'
Hobj         As Long   'Object handle'
MsgDesc      As MQMD   'Message descriptor'
GetMsgOpts   As MQGMO  'Options that control the action of MQGET'
BufferLength As Long   'Length in bytes of the Buffer area'
Buffer       As String 'Area to contain the message data'
DataLength   As Long   'Length of the message'
CompCode     As Long   'Completion code'
Reason       As Long   'Reason code qualifying CompCode'
```

MQINQ

```
MQINQ (Hconn, Hobj, SelectorCount, Selectors, IntAttrCount, IntAttrs,
      CharAttrLength, CharAttrs, CompCode, Reason)
```

Declare the parameters as follows:

```
Hconn          As Long   'Connection handle'
Hobj           As Long   'Object handle'
SelectorCount  As Long   'Count of selectors'
Selectors      As Long   'Array of attribute selectors'
IntAttrCount   As Long   'Count of integer attributes'
IntAttrs       As Long   'Array of integer attributes'
CharAttrLength As Long   'Length of character attributes buffer'
CharAttrs      As String 'Character attributes'
CompCode       As Long   'Completion code'
Reason         As Long   'Reason code qualifying CompCode'
```

The MQI in Visual Basic

MQOPEN

```
MQOPEN (Hconn, ObjDesc, Options, Hobj, CompCode, Reason)
```

Declare the parameters as follows:

```
Hconn    As Long 'Connection handle'
ObjDesc  As MQOD 'Object descriptor'
Options  As Long 'Options that control the action of MQOPEN'
Hobj     As Long 'Object handle'
CompCode As Long 'Completion code'
Reason   As Long 'Reason code qualifying CompCode'
```

MQPUT

```
MQPUT (Hconn, Hobj, MsgDesc, PutMsgOpts, BufferLength, Buffer, CompCode,
      Reason)
```

Declare the parameters as follows:

```
Hconn        As Long   'Connection handle'
Hobj         As Long   'Object handle'
MsgDesc      As MQMD   'Message descriptor'
PutMsgOpts   As MQPMO  'Options that control the action of MQPUT'
BufferLength As Long   'Length of the message in Buffer'
Buffer       As String 'Message data'
CompCode     As Long   'Completion code'
Reason       As Long   'Reason code qualifying CompCode'
```

MQPUT1

```
MQPUT1 (Hconn, ObjDesc, MsgDesc, PutMsgOpts, BufferLength, Buffer,
       CompCode, Reason)
```

Declare the parameters as follows:

```
Hconn        As Long   'Connection handle'
ObjDesc      As MQOD   'Object descriptor'
MsgDesc      As MQMD   'Message descriptor'
PutMsgOpts   As MQPMO  'Options that control the action of MQPUT1'
BufferLength As Long   'Length of the message in Buffer'
Buffer       As String 'Message data'
CompCode     As Long   'Completion code'
Reason       As Long   'Reason code qualifying CompCode'
```

MQSET

```
MQSET (Hconn, Hobj, SelectorCount, Selectors, IntAttrCount, IntAttrs,
      CharAttrLength, CharAttrs, CompCode, Reason)
```

Declare the parameters as follows:

```
Hconn          As Long   'Connection handle'
Hobj           As Long   'Object handle'
SelectorCount  As Long   'Count of selectors'
Selectors      As Long   'Array of attribute selectors'
IntAttrCount   As Long   'Count of integer attributes'
IntAttrs       As Long   'Array of integer attributes'
CharAttrLength As Long   'Length of character attributes buffer'
CharAttrs      As String 'Character attributes'
CompCode       As Long   'Completion code'
Reason         As Long   'Reason code qualifying CompCode'
```

Elementary data types in Visual Basic

Table 9. 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`

The MQI in Visual Basic

Structure data types in Visual Basic

MQDLH—Dead-letter header

MQSeries for Windows does not support dead-letter queues, so you cannot use the MQDLH structure.

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      'Reserved'
  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
```

MQTM—Trigger message

MQSeries for Windows does not support triggering, so you cannot use the MQTM structure.

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
```

The MQI in Visual Basic

Chapter 16. Application programming restrictions

The Message Queue Interface (MQI) is the MQSeries application programming interface. MQSeries for Windows supports nearly all the features of the MQI; this chapter uses the following sections to describe the features MQSeries for Windows does not support:

- "Unsupported features of the MQI calls"
- "Unsupported features of the MQI structures" on page 150
- "MQI attributes on Windows" on page 152

For a description of the full MQI, see the *MQSeries Application Programming Reference*.

Unsupported features of the MQI calls

For each MQI call, this section describes the differences in processing in MQSeries for Windows.

Pointers

MQSeries for Windows cannot 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 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 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

MQI restrictions

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

When you issue an MQGET call using the MQGMO_WAIT option, the call completes only when a suitable message arrives on the queue, or when the wait interval expires. During the time that the call is waiting, Windows program messages (not MQSeries messages) are still sent to the appropriate window procedure, so your program can remain responsive to other requests.

You must ensure that your code that processes Windows program messages does not assume that the MQGET call returns data to the application straightaway. If it attempts to access data that is not yet available, errors can easily occur. Also, if you attempt to make other MQI calls while the MQGET call is waiting, reason code 2219 is returned to show that another call is busy.

For a list of the options that MQSeries for Windows supports for the MQGET call, see "Get-message options structure (MQGMO)."

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 151.

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 152.

Unsupported features of the MQI structures

This section describes how MQSeries for Windows uses the MQI structures in different ways from other MQSeries products.

Dead-letter header structure (MQDLH)

MQSeries for Windows does not support dead-letter queues, so you cannot use the MQDLH structure.

Get-message options structure (MQGMO)

In the Options field of MQGMO structure:

- 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
- MQSeries for Windows ignores the MQGMO_FAIL_IF_QUIESCING value.

MQSeries for Windows supports all the other fields of the MQGMO structure.

Put-message options structure (MQPMO)

In the Options field of 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 the MQPMO structure.

Message descriptor structure (MQMD)

MQSeries for Windows supports all the fields of the MQMD structure. Note the following default values:

- PutApplType = `MQAT_WINDOWS`
- UserIdentifier = `WINDOWS`

PutDate

On MQSeries for Windows, the queue manager uses the TZ environment variable to calculate Greenwich Mean Time (GMT). You must set this variable to get the correct time stamp in your message context. The value for the UK is GMT0BST.

Object descriptor structure (MQOD)

MQSeries for Windows supports all the fields of the MQOD structure, but it ignores the contents of the AlternateUserId field.

Trigger message structure (MQTM)

MQSeries for Windows does not support triggering, so you cannot use the MQTM 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:

- The queue manager attributes are listed in Table 10.
- The queue attributes are listed in Table 11 on page 153.
- The channel attributes are listed in Table 12 on page 154.

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 10. Attributes of queue managers on Windows

Attribute	**Default value**
QMgrName	
QMgrDesc	
Platform	MQPL_WINDOWS (= 5)
CommandLevel	100
MaxPriority	9
CommandInputQName	blanks
DefXmitQName	blanks
CodedCharSetId	
MaxHandles	256
MaxUncommittedMsgs	10000
MaxMsgLength	4 194 304 bytes
SyncPoint	MQSP_AVAILABLE

Table 11. Attributes of queues on Windows

Attribute	Default value
QName	
QType	
QDesc	blank
InhibitGet	MQQA_GET_ALLOWED
InhibitPut	MQQA_PUT_ALLOWED
DefPriority	0
DefPersistence	MQPER_NOT_PERSISTENT
MaxQDepth	5000
MaxMsgLength	4 194 304
BackoutThreshold	0
BackoutRequeueQName	blank
Shareability	MQQA_SHAREABLE
DefInputOpenOption	
HardenGetBackout	MQQA_BACKOUT_NOT_HARDENED
MsgDeliverySequence	MQMDS_PRIORITY
RetentionInterval	999 999 999
DefinitionType	
Usage	MQUS_NORMAL
OpenInputCount	
OpenOutputCount	
CurrentQDepth	
CreationDate	
CreationTime	
RemoteQName	blank
RemoteQMgrName	blank
XmitQName	blank
BaseQName	blank

MQI attributes

Table 12. Attributes of channels on Windows

Attribute	Default value
BatchSize	50
ChannelName	
ChannelType	
ConnectionName	
Desc	
DiscInterval	6000
LongRetryCount	999 999 999
LongRetryInterval	1200
MaxMsgLength	4 194 304
MCAUserIdentifier	
MsgExit	
MsgUserData	
ReceiveExit	
ReceiveUserExit	
SecurityExit	
SecurityUserExit	
SendExit	
SendUserExit	
SeqNumberWrap	999 999 999
ShortRetryCount	10
ShortRetryInterval	60
TransportType	MQXPT_TCP
XmitQName	

Chapter 17. 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 5, "Verifying your installation and configuration" on page 35. That chapter tells you how to use two of the samples to verify you have correctly installed and set up an MQSeries for Windows queue manager. You can also use them to verify that you have correctly installed and connected queue managers on two separate workstations (see Chapter 8, "Setting up and verifying two queue managers" on page 67).

Both the source code and the executable files are supplied with MQSeries for Windows. There is an icon for the executable file for each sample in the MQSeries for Windows program group.

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 executable file is named AMQSPUTW.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 executable file is named AMQSGETW.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 executable file is named AMQSBCGW.EXE.

Sample programs

General design

Each sample program uses a single window, and the design of this window is similar in each program:

- 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 13 shows which MQI calls each sample program demonstrates.

Table 13. MQI calls used in the MQSeries for Windows sample programs

MQI call	**Putting Messages**	**Getting Messages**	**Browsing Messages**
MQCONN	Yes	Yes	Yes
MQOPEN for output	Yes	No	No
MQOPEN for input	No	Yes	No
MQOPEN for browsing	No	No	Yes
MQPUT	Yes	No	No
MQGET	No	Yes	No
MQGET for browsing	No	No	Yes
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 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.

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.

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.

Building the 16-bit executable files

This section describes the files used by the sample programs. It also describes how to build a program if you have modified the sample source code.

In addition to the source files, MQSeries for Windows supplies 16-bit executable and make files for the samples. The file names 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 \MQW\SAMPLES directory.

The following tables list the files that each sample program uses.

Table 14. Files for 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 15. Files for 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 16. Files for 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

32-bit samples

Note: If you rename a source file, you must edit the make file to refer to the new name.

After you have made a backup copy, use the following procedure to rebuild a sample. You must use the 16-bit Microsoft Visual C++ Compiler, Version 1.5.

1. Start a DOS session.

   You can do this either by exiting from Windows or by clicking on the MS-DOS command prompt. (If you are not familiar with Windows 3.1, the MS-DOS command prompt is in the Main program group.)

2. If you have not already done so, run MSVCVARS (which is supplied with Microsoft Visual C++) to ensure that the build environment is set up.

3. From 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, AMQSPUTW.EXE.

Generating 32-bit samples

Although 32-bit samples are not supplied with MQSeries for Windows, you can generate 32-bit programs from the supplied source files. You must provide an NMAKE file for the 32-bit compiler you want to use. Other than specifying compiler and linker options for the new compiler, the only thing to remember is to link the sample with the library MQM.LIB, which provides the 32-bit entry points for the MQI calls.

The sample programs have been tested with Microsoft Visual C++ v2.0.

Note: Although the samples use 32-bit addressing, the MQI calls are passed straight through to the 16-bit MQI. There is no 32-bit implementation of the MQI calls in MQSeries for Windows.

Part 6. Appendixes

Appendix A. How MQSeries for Windows differs from the MQSeries family

MQSeries for Windows is the MQSeries queue manager for the Microsoft Windows platform. It is designed to minimize system requirements so that workstations with relatively modest specifications can use commercial messaging. This 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 instrumentation events). If you use an unsupported attribute in a command or an MQI call, MQSeries for Windows returns a value to show that the attribute is not supported.

Authority checking on the MQOPEN call
MQSeries for Windows does not support the SETMQAUT and DSPMQAUT commands.

Command Server
MQSeries for Windows does not support the MQSeries Command Server, so it does not support any MQSeries feature that uses the command server (for example, PCF commands and remote administration).

Context passing
MQSeries for Windows does not copy context information from messages it receives from other queue managers. This is because MQSeries for Windows is intended to be a leaf node; it is not intended to be an intermediate node in a network of queue managers, where messages received from one queue manager are passed on to another.

Control commands
In other MQSeries products, you can issue control commands from the command line. MQSeries for Windows provides utilities that perform the functions of some of these commands; for example, they start and stop a queue manager. For a comparison with the MQSeries control commands, see Table 17 on page 167.

Data conversion
When an MQSeries for Windows queue manager receives data from a queue manager running on a different platform, it cannot convert the machine encoding, integer representation, or coded character set of the application data. Also, it cannot run data conversion exits. This means that any data conversion that is required must be performed by the other queue manager.

Family differences

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.

Distributed Computing Environment (DCE) directories
MQSeries for Windows does not support DCE directories.

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
MQSeries for Windows does not support instrumentation events. These are facilities that can be used in other MQSeries products to monitor the operation of queue managers in a network of MQSeries systems.

Media recovery and logging
MQSeries for Windows does not support the creation of a sequence of log records that contain an image of an object. Other MQSeries products allow you to create such records and re-create objects from this image.

Message Queue Interface (MQI)
MQSeries for Windows supports a subset of the MQI.

To understand those features of the MQI that MQSeries for Windows does not support, see Chapter 16, “Application programming restrictions” on page 149.

Message retry exit
MQSeries for Windows does not support a message retry exit because you use such an exit only when processing dead-letter queues.

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 Appendix C, “MQSC commands supported by MQSeries for Windows” on page 169.

However, MQSeries for Windows provides the MQSC Commands utility, which allows you to type MQSC commands in a window (and test and reissue them) and run MQSC command files. This is described in Chapter 10, “Using MQSC commands” on page 85.

Family differences

MQSeries client and server support
You cannot use an MQSeries for Windows queue manager as an MQSeries client, nor can you use it to support its own MQSeries clients.

Network support
MQSeries for Windows supports TCP/IP only.

Object Authority Manager (OAM)
MQSeries for Windows does not provide a security manager. It does not support the SETMQAUT and DSPMQAUT commands.

Process definitions
Other MQSeries products use process definitions for setting up the automatic triggering of applications. MQSeries for Windows does not support triggering or process definitions.

Programmable Command Formats (PCFs)
MQSeries for Windows does not support PCFs.

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 the quiescing of a queue manager. This is the ability to allow applications to finish processing before the queue manager is stopped, and to prevent any further applications starting.

Sample programs
MQSeries for Windows provides Windows versions of some of the MQSeries sample programs. The MQSeries for Windows samples are described in Chapter 17, “Sample programs” on page 155.

Security manager
See *object authority manager*.

Signaling MQSeries for Windows does not support signaling. You cannot use the MQGMO_SET_SIGNAL option with the MQGET call.

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

Family differences

Two-phase commit

MQSeries for Windows does not support two-phase commit. This is a protocol for the coordination of changes to recoverable resources when more than one resource manager is used by a single transaction.

However, MQSeries for Windows does allow the queue manager to commit or back out of units of work.

Appendix B. MQSeries control commands

Control commands are commands that, in MQSeries for OS/2 or AIX, you type at a command prompt. In MQSeries for Windows, you use the supplied utilities to perform the functions of some of these control commands. To see which utilities to use, see Table 17.

Table 17 (Page 1 of 2). Control commands and MQSeries for Windows

MQSeries command	Description	Support on MQSeries for Windows
CRTMQCVX	Create data conversion exit	Not supported.
CRTMQM	Create queue manager	This function is provided by the Create Components utility.
DLTMQM	Delete queue manager	This function is provided by the Delete Components utility.
DSPMQAUT	Display authority	Not supported.
DSPMQCSV	Display command server	Not supported.
DSPMQTRN	Display MQSeries transactions	Not supported.
ENDMQM	Stop queue manager	This function is provided by the Standard Controls utility and the Advanced Controls utility.
RCDMQIMG	Record media image	Not supported.
RCRMQOBJ	Recreate object	Not supported.
RSVMQTRN	Resolve MQSeries transactions	Not supported.
RUNMQCHI	Run channel initiator	Not supported.
RUNMQCHL	Run channel	This function is provided by the Standard Controls utility and the Advanced Controls utility.
RUNMQLSR	Run listener	This function is provided by the Standard Controls utility and the Advanced Controls utility. You must use either the Create Components utility or the Create and Go utility to create a channel group that contains the listener. You can add the listener to an existing channel group using the Advanced Controls utility.

Control commands

Table 17 (Page 2 of 2). Control commands and MQSeries for Windows		
MQSeries command	**Description**	**Support on MQSeries for Windows**
RUNMQSC	Run MQSeries commands	This function is provided by the MQSC Commands utility. You can also run an MQSC command file when you create a queue manager using the Create Components utility or the Create and Go utility.
RUNMQTMC	Start client trigger monitor	Not supported.
RUNMQTRM	Start trigger monitor	Not supported.
SETMQAUT	Set authority	Not supported.
STRMQCSV	Start command server	Not supported.
STRMQM	Start queue manager	This function is provided by the Standard Controls utility and the Advanced Controls utility. An autostart facility is also available.

Appendix C. MQSC commands supported by MQSeries for Windows

MQSeries for Windows supports a subset of the MQSeries commands (MQSC). This subset is shown in Table 18. For a description of the syntax of each command, see the online *MQSeries for Windows Command Reference*.

Table 18 (Page 1 of 2). MQSC commands

Command	Available in MQSeries for Windows
ALTER CHANNEL	Yes
ALTER NAMELIST	No
ALTER PROCESS	No
ALTER QALIAS	Yes
ALTER QLOCAL	Yes
ALTER QMGR	Yes
ALTER QMODEL	Yes
ALTER QREMOTE	Yes
ALTER SECURITY	No
ALTER TRACE	No
ARCHIVE LOG	No
CLEAR QLOCAL	Yes
DEFINE BUFFPOOL	No
DEFINE CHANNEL	Yes
DEFINE MAXSMSGS	No
DEFINE NAMELIST	No
DEFINE PROCESS	No
DEFINE PSID	No
DEFINE QALIAS	Yes
DEFINE QLOCAL	Yes
DEFINE QMODEL	Yes
DEFINE QREMOTE	Yes
DEFINE STGCLASS	No
DELETE CHANNEL	Yes
DELETE NAMELIST	No
DELETE PROCESS	No
DELETE QALIAS	Yes
DELETE QLOCAL	Yes
DELETE QMODEL	Yes
DELETE QREMOTE	Yes
DISPLAY CHANNEL	No
DISPLAY CHSTATUS	No
DISPLAY DQM	No
DISPLAY CMDSERV	No
DISPLAY MAXSMSGS	No
DISPLAY NAMELIST	No
DISPLAY PROCESS	No
DISPLAY QMGR	No
DISPLAY QUEUE	No
DISPLAY SECURITY	No
DISPLAY STGCLASS	No
DISPLAY THREAD	No
DISPLAY TRACE	No
DISPLAY USAGE	No
PING CHANNEL	No
PING QMGR	No
RECOVER BSDS	No
REFRESH SECURITY	No
RESET CHANNEL	Yes
RESOLVE CHANNEL	Yes
RESOLVE INDOUBT	No
RVERIFY SECURITY	No

Table 18 (Page 2 of 2). MQSC commands

Command	**Available in MQSeries for Windows**
START CHANNEL	Yes
START CHINIT	No
START CMDSERV	No
START LISTENER	No
START QMGR	No
START TRACE	No
STOP CHANNEL	Yes
STOP CHINIT	No
STOP CMDSERV	No
STOP LISTENER	No
STOP QMGR	No
STOP TRACE	No

Appendix D. 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 contains definitions for the MQSeries for Windows default and system objects. The default object definitions contain a complete set of attributes for that object. When you create a new object, its attributes are inherited from the default object, except the ones 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, with the exception that the PUT attribute has a value of DISABLED, whereas the default is PUT(ENABLED).

The system objects are required for the operation of a queue manager or channel. Table 19 lists the objects defined in the supplied AMQSCOMW.TST file. In a default installation, the file is supplied in the directory C:\MQW\QMGRS.

The objects specified in AMQSCOMW.TST are created automatically when you create a queue manager using the Create Components utility.

Table 19. 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

Note: You can modify these queues to change the default attributes. If you do, remember to keep a copy of the original file.

Objects for the sample programs

The sample MQSC command file AMQSCOSW.TST contains the definition of a queue you can use with the supplied sample programs. This queue is created automatically when you create a queue manager if you select the **Load MQSC file for the sample programs** option in the Create Queue Manager window of the Create Components utility.

Table 20 shows the object defined in the supplied AMQSCOSW.TST file. In a default installation, the file is supplied in the directory C:\MQW\SAMPLES.

Table 20. Object defined in AMQSCOSW.TST

Object name	Description
SYSTEM.SAMPLE.LOCAL	Sample local queue

Objects for verifying the configuration of two workstations

The sample MQSC command files VENUS.TST and MARS.TST contain definitions for queues and channels you can use to connect two workstations. For information on how to do this, see Chapter 8, “Setting up and verifying two queue managers” on page 67.

These queues are created automatically when you create a queue manager if you specify these TST files in the **Load MQSC files for your application** field in the Create Queue Manager window of the Create Components utility.

Table 21 lists the objects defined in the file VENUS.TST. Table 22 on page 173 lists the objects defined in the file MARS.TST. In a default installation, both files are supplied in the directory C:\MQW\SAMPLES.

Table 21. 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 22. 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 E. Reason codes

When an MQI call or MQSC command fails, it returns a reason code in numeric form. Use Table 23 to look up the meaning of this code.

If you want more information about these 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 23 (Page 1 of 4). 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
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

Reason codes

Table 23 (Page 2 of 4). Reason codes returned by MQI calls and MQSC commands

Numeric	Literal
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
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

Table 23 (Page 3 of 4). Reason codes returned by MQI calls and MQSC commands

Numeric	Literal
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
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
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

Reason codes

Table 23 (Page 4 of 4). Reason codes returned by MQI calls and MQSC commands

Numeric	Literal
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
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

Appendix F. Error messages

The error messages shown in this appendix are generated by the MQSC Commands utility. They also appear in the file MQSC.LOG when you run an MQSC command file.

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 E, “Reason codes” on page 175, 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 Commands utility.

Action: Restart the MQSC Commands utility. If the error persists, reinstall MQSeries for Windows, then restart the MQSC Commands utility. 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 Commands utility.

Action: Restart the MQSC Commands utility. If the error persists, reinstall MQSeries for Windows, then restart the MQSC Commands utility. 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 Commands utility.

Action: Restart the MQSC Commands utility. If the error persists, reinstall MQSeries for Windows, then restart the MQSC Commands utility. 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.

Appendix G. Notices

Trademarks

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

AIX	IBM	NetView
AIX/6000	MQ	OS/2
BookManager	MQSeries	Win-OS/2
CICS		

Notices

Part 7. 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

Advanced Controls utility. In MQSeries for Windows, a utility for administrators to start or stop queue managers, channel groups, channels, and transport links. With the utility you can view the status and attributes of these components, and you can change their values. See also *Standard Controls utility*.

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.

autostart. A facility for starting a queue manager, channel group, or transport link.

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.

CCSID. Coded character-set identifier.

channel. See *message channel*.

channel group. A named collection of channels, owned by one queue manager, that you can start and stop 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 MQSeries component that reads commands from the system-command input queue, verifies them, and passes on the valid ones for processing by the command processor. MQSeries for Windows does not have a command server.

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.

Glossary

commit. See *single-phase commit* and *two-phase commit*.

completion code. A return code indicating whether or not 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. (1) In the MQSeries for Windows utilities, a component is a queue manager, queue, channel, channel group, or transport link. (2) When installing MQSeries for Windows, a component is a separately installable part of the product.

connect. In MQSeries, a processing sequence in which an application issues an MQI connect call (MQCONN) and where the queue manager specified on the call returns a connection handle. The application uses this connection handle on subsequent MQI calls.

connection handle. The identifier, or token, by which a program accesses the queue manager to which it is connected.

Connection Monitor. The initial dialog within the Standard Controls and Advanced Controls utilities that shows the current status of queue managers, channel groups, and transport links.

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.

Create and Go utility. In MQSeries for Windows, a utility that allows users to create automatically the MQSeries components they require. An administrator must supply the user with an initialization (INI) file that contains definitions of those components.

Create Components utility. In MQSeries for Windows, a utility that allows you to create queue managers, queues, channels, channel groups, or transport links.

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.

Delete Components utility. In MQSeries for Windows, a utility that allows you to delete queue managers, queues, channels, channel groups, or transport links.

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

exit program. In MQSeries for Windows, a program that contains device commands for starting or stopping a communication device, such as a modem. The program is called by the transport link when the link is started or stopped.

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

leaf node. In a network, a leaf node is a node connected to a server that is on the outer edge of the network. It is intended for use by a single user, and not as an intermediate node between other nodes.

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.

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. Commands in a specific format that change the attributes of MQSeries objects.

MQSC Commands utility. In MQSeries for Windows, a utility that allows you to type and edit MQSC commands, and to run MQSC files.

Glossary

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 entity that defines the properties of a queue manager, a queue, or a channel.

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

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.

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.

program temporary fix (PTF). A solution or by-pass of a problem diagnosed by IBM service engineering as the result of a defect in a current, unaltered, release of a product.

PTF. Program temporary fix.

Q

queue. An MQSeries object. Message queuing applications can put messages on, and get messages from, a queue. A queue is owned and maintained by a queue manager. 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.

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 non-delivery) 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.

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.

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 Information utility. In MQSeries for Windows, a utility that displays service information about an MQSeries for Windows installation. The information includes the amount of free disk space remaining and the release levels of the product files.

Service Trace utility. In MQSeries for Windows, a utility that traces the operation of an MQSeries for Windows queue manager. Use it to help you debug an MQSeries application.

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.

Standard Controls utility. In MQSeries for Windows, a utility that lets end users start or stop queue managers, channel groups, channels, and transport links. It also shows the status of these components and allows you to find out the current values of the attributes of queue managers, queues, and channels. It does not allow you to change any of the values. See also *Advanced Controls utility*.

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.

transport link. In MQSeries for Windows, a named link to another queue manager over a dial-up device. The transport link program calls an exit program that starts or stops the dial-up device.

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*.

V

Verify Install utility. In MQSeries for Windows, a utility that tests that your installation was successful. The utility creates a queue manager, puts messages on a queue, reads the messages from a queue, then deletes the queue manager.

Index

Index

Index

Index

Index

Index

Index

Index

Sending your comments to IBM

MQSeries for Windows**

User’s Guide

GC33-1822-00

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

GC33-1822-00

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**
User's Guide GC33-1822-00

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.).

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 ____________________

4 Fasten here with adhesive tape

1 Cut along this line

1 Cut along this line

Program Number: 5622-960

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

GC33-1822-00