Willow Technology

MQSeries Client for Mac™ OS
Version 2-02
October, 1998

MQSeries Clients Addendum

MQSERIES FOR MAC OS RUNTIME CLIENT

MQSeries Clients Addendum

Phone 408.377.7292 • Fax 408.377.7293
email: info@willowtech.com
www.willowtech.com

Trademarks

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

MQSeries

MQ

AIX

AS/400

MVS/ESA

RISC System/6000

OS/2

OS/400

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

Microsoft, Windows and the Windows 95 logo are trademarks or registered trademarks of Microsoft Corporation.

Mac OS, Macintosh, PowerMac and Open Transport are trademarks or registered trademarks of Apple Computer, Inc.

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

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

Table of Contents

Chapter 1

Preparing for Installation

The information in this manual provides information specific to the Mac OS MQI client, and is intended to be read in conjunction with the IBM MQSeries Clients reference; IBM publication number SC33-1632-xx.

This chapter details the platform support and the communications protocol support for Mac OS clients only. You can find Hardware and Software requirements for other supported client platforms in the MQSeries Clients reference, IBM publication number SC33-1632-xx.

For your server platform hardware and software requirements, see the MQSeries System Management Guide for your platform, or the MQSeries for MVS/ESA Program Directory.

For capacity planning information, see the MQSeries Planning Guide.

Support for MQSeries Client for Mac OS

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

> **Note**
>
> Make sure that code conversion from the CCSID of the Mac OS client is supported by the server. See the Language support tables in the MQSeries Application Programming Reference.

The MQSeries server products can accept connections from Mac OS MQI clients:

MQSeries for SCO OpenServer Version 2 (from Willow Technology)

MQSeries for UnixWare Version 2 (from Willow Technology)

MQSeries for AIX Versions 2 and 5

MQSeries for AT&T GIS UNIX Version 2

MQSeries for HP-UX Versions 2 and 5

MQSeries for OS/2 Versions 2 and 5

MQSeries for Windows NT Versions 2 and 5

MQSeries for MVS/ESA Version 1 Release 1.4

MQSeries for OS/400 Version 3 Release 2

MQSeries for SINIX and DC/OSx Version 2

MQSeries for SunOS Version 2.2

MQSeries for Sun Solaris Versions 2 and 5

Communications

TCP/IP over Apple Open Transport version 1.1 or later is the only transmission protocol supported by the MQSeries Client for Mac OS software.

MQI client on Mac OS: hardware and software required

Machine requirements

An MQSeries client can run on the Mac OS, on any PowerPC or 68k based personal computer that is capable of running the MQSeries client code, and which has sufficient random access memory (RAM) and disk storage to meet the combined requirements of the programming prerequisites, the MQSeries client code, the access methods, and the application programs.

Programming requirements

The following are prerequisites for MQSeries applications running an MQSeries client on the Mac OS.

This is the minimum supported software level. Later levels, if any, will be supported unless otherwise stated.

- Mac OS System 7.6
- Open Transport release 1.1

Supported compilers

The following compilers have been tested and are supported for both C and C++ bindings:

- Metrowerks CodeWarrior Pro 1
- Microsoft Visual C++ Cross Development Environment (CDE) 4.0

Supported scripting environments

- HyperCard 2.4

Chapter 2

Installing the MQSeries Client for Mac OS

The MQSeries Client for Mac OS is a fully compliant MQI client. The product is developed, sold and supported by Willow Technology under license from IBM.

The MQSeries Client for Mac OS is licensed by Willow Technology for use on a single personal computer, and is distributed on CD-ROM.

The program, "MQSeries Client for Mac OS Installer", located on the CD-ROM, is used to install the files necessary to run MQI applications on your computer.

Before installing the software, please consult the "Read Me" file located with the installer for the latest information, known problems and fixes. Invoking the installer will also give you an opportunity to read and print the "Read Me" file.

Installation Instructions

1. Double-click the "MQSeries Client for Mac OS Installer" icon to begin the installation procedure.

2. When the Willow logo is displayed, press the "Continue" button.

3. Review (and print, if needed) the "Read Me" that is displayed. When finished, press the "Continue" button.

4. Select the Install Location for the MQSeries Client for Mac OS software. You can select a Disk or specific Folder using the drop-down list object for this. A folder entitled "MQSeries Client" will be created at this location. The Mac OS client documentation, sample applications, the Trigger Monitor and other client specific files will be installed into this folder. The installer will also install several files into your System Folder, Extensions and Control Panel folders. For the exact files copied onto you system, and their locations, refer to the installer Log File copied into the "MQSeries Client" folder.

5. When the installer has finished, you will be asked to either press Continue if you wish to perform additional installations, or press Quit if finished. In general, you will want to press “Quit” to exit the installer.

6. After the installer has terminated, the “MQSeries Client” folder is opened on the desktop.

7. Verify that you installation is working properly by following the Verification steps described in Chapter 3.

Chapter 3

Verifying the installation

The supplied samples can be used to verify that the installation has been completed successfully and that the communication link is working.

This chapter gives instructions on how to verify that an MQSeries Client for Mac OS has been installed correctly, by guiding you through the following tasks:

1. Setting up the MQSeries client
2. Putting a message on the queue
3. Getting the message from the queue.

Instruction for setting up the MQSeries server are described in Chapter 4 of the MQSeries Clients reference.

These instructions assume that:

- The full MQSeries product has been installed on a server:

  The Base Product and Distributed Queuing without CICS, and the Client Attachment feature on MVS/ESA.

  The full MQSeries for OS/400 product on OS/400 platforms.

  The Base Product and Server on other platforms.

- The MQSeries Mac OS client software and supplied files have been installed on another the Macintosh or PowerMac systems to be used.

TCP/IP is the only supported transmission protocol. It is assumed that you have TCP/IP configured on the server and the MQSeries client machines, and that it has been initialized on the server.

> **Note**
>
> Compiled samples amqsputc and amqsgetc are included in the "MQSeries Client" folder that you installed, as described in Chapter 2 of this document, "Installing the MQSeries Mac OS client".

The verification scenario

The following example assumes you have created a queue manager called queue.manager.1 (on platforms other than MVS/ESA which has a 4-character restriction on queue manager names), a local queue called QUEUE1, and a server-connection channel called CHANNEL1 on the server. It shows how to create the client-connection channel on the MQSeries Client for Mac OS workstation; and how to use the sample programs to put a message onto a queue, and then get the message from the queue.

> **Note**
>
> MQSeries object definitions are case-sensitive. You must type the examples **exactly** as shown.

Security

The verification example does **not** address any client security issues. See Chapter 5, "Setting up MQSeries Client for Mac OS security" for details if you are concerned with MQSeries client security issues.

Setting up the server

Refer to Chapter 4, "Verifying the Installation" of the MQSeries Clients base reference manual for details on setting up you MQSeries server environment.

Setting up the MQSeries client

When an MQSeries application is run on the MQSeries Client for Mac OS, the information it requires is the name of the MQI channel, the communication type, and the address of the server to be used. You provide this by defining a client-connection channel. This example uses the MQSERVER environment variable to do this - the simplest way, although not the only one. The name used must be same as the name used for the server-connection channel defined on the server.

Before starting, **ping** the **server-address** (where **server-address** is the TCP/IP hostname of the server) to confirm that your MQSeries client and server TCP/IP sessions have been initialized. You can use the network address, in the format

n.n.n.n, in the ping instead of the hostname. If the ping fails, check that your TCP/IP software is correctly configured and operational.

> **Note**
>
> Apple's MacTCP Ping is not compatible with Open Transport., and will not be updated. MacPing from Dartmouth (ftp://ftp.dartmouth.edu), OTTool from Neon Software (ftp://ftp.neon.com) and Mac TCP Watcher v2.0 from Peter N. Lewis and Stairways Software (ftp://ftp.share.com) are Open Transport-compatible alternatives.

Define a client-connection channel, using MQSERVER

1. Open the MQSetup Control Panel to create a client-connection channel. The initial and default Configuration is "MQ Configuration". You can rename or create a new Configuration if desired. Remember to set the configuration you will be using as the default by pressing the "Set Default" button if you change the existing default.

2. In the MQServer field, type the following:

   CHANNEL1/TCP/server-address(port)

   where **server-address** is the TCP/IP hostname of the server, **port** is optional and is the TCP/IP port number the server is listening on. The default port number is 1414 if no other was specified on the Start Listener or inetd commands on the server.

3. All other fields can be defaulted at this time.

4. Close the MQSetup Control Panel by clicking in the box in the top left corner of the Control Panel window. This will save your configuration information and close the Control Panel.

Putting a message on the queue

On the MQSeries Client for Mac OS workstation, put a message on the queue using the amqsputc sample program:

1. Open the Samples folder in your "MQSeries client" folder.

2. Double-click on the amqsput icon.

3. A command line will prompt you to enter the name of the queue and queue manager which you wish to connect and send to. Enter:

   QUEUE1 qmgr

   where qmgr is the name of the queue manager on the server (queue.manager.1 in the non-MVS/ESA example above).

4. An output window will be opened and the following message is displayed:

   Sample AMQSPUT0 start

   target name is QUEUE1

5. Type some message text and then press Enter or Return.

6. The following message is displayed in the output window:

   Sample AMQSPUT0 end

7. The message is now on the queue.

8. Select the File/Quit menu option or click in the box in the top left corner of the output window to terminate the amqsputc application.

Getting the message from the queue

On the MQSeries client, get a message from the queue using the amqsgetc sample program:

Change to the directory containing the sample programs, and then enter the following command:

1. Open the Samples folder in your "MQSeries client" folder if this is not already open.

2. Double-click on the amqsget icon.

3. A command line will prompt you to enter the name of the queue and queue manager which you wish to connect and send to. Enter:

   QUEUE1 qmgr

   where qmgr is the name of the queue manager on the server (queue.manager.1 in the non-MVS/ESA example above).

4. An output window will be opened and the message(s) on the queue is(are) removed and displayed in the window.

5. Select the File/Quit menu option or click in the box in the top left corner of the output window to terminate the amqsgetc application.

Ending verification

The verification process is now complete.

Chapter 4

Configuration

MQSeries Client for Mac OS software only supports TCP/IP using Apple Open Transport. Refer to the Open Transport documentation for details concerning configuring your TCP/IP connection.

There is no specific initialization of the TCP/IP transport required other than it be installed on your Mac OS system. You may need to restart your system after installing Open Transport.

Chapter 5

Setting up MQSeries Client for Mac OS security

You must consider MQSeries client security, so that the client applications do not have unrestricted access to resources on the server. There are two aspects to security between a client application and its queue manager server: authentication and access control.

Authentication

Authentication is described in Chapter 6 of the MQSeries Clients reference. There are no special considerations for Mac OS clients.

User ID and password

If a security exit is not defined on an MQSeries Client for Mac OS, the values of two environment variables MQ_USER_ID and MQ_PASSWORD will be transmitted to the server and will be available to the server security exit in the Channel definition when it is invoked. These values may be used to verify the identity of the MQSeries client.

Note

Note that MYUSERID and MYPASSWORD must be in uppercase if the MQSeries client is going to communicate with an MQSeries server on OS/400.

1. Open the MQSetup Control Panel:
2. Type the User ID, <MYUSERID> (without the < >), you wish to use into the MQ_USER_ID field.
3. Type the password, <MYPASSWORD> (without the < >), for this User ID into the MQ_PASSWORD field.

Access Control

Access control in MQSeries is based upon the user identifier associated with the process making MQI calls. For Mac OS clients, the process that issues the MQI calls is the server Message Channel Agent. The user identifier used by the server MCA is that contained in the MCAUserIdentifier field of the MQCD. The contents of MCAUserIdentifier are determined by the following:

- Any values set by security exits
- MQ_USER_ID environment variable
- MCAUSER (in server-connection channel definition)

Depending upon the combination of settings of the above, MCAUserIdentifier is set to the appropriate value. If security exits are provided, MCAUserIdentifier may be set by the exit. Otherwise MCAUserIdentifier is determined as shown in the following table:

MQ client ID MQ_USER_ID	Server channel MCAUSER	Value Used	Notes
Not Set or Set	Set	MCAUSER	1
Set	Blanks	MQ_USER_ID	1
Not Set	Blanks	For MVS/ESA: The value used is the user ID assigned to the channel initiator started task by the MVS/ESA started procedures table. TCP/IP (non-MVS/ESA): User ID from inetd.conf entry. SNA (non-MVS/ESA): User ID from SNA Server entry	2
Not Set or Set	Not Set	TCP/IP: User ID from inetd.conf entry. SNA: User ID from SNA Server entry	2,3

Notes

1. For Windows NT and UNIX servers, the MCAUSER from the channel definition is changed to lowercase before being used. so MCA user identifiers with one or more uppercase letters will not work if placed in the MCAUSER field of the channel definition. They will work however if they are put in the client environment variable MQ_USER_ID and MACUSER is blank.

2. For OS/2, no user ID is available from either Communications Manager/2.

3. For MVS/ESA the channel user ID takes the value of MCAUserIdentifier as determined above. See the MQSeries for MVS/ESA System Management Guide for more information.

Chapter 6

MQSeries environment variables (MQSetup Control Panel)

This chapter describes the environment variable equivalents that you can use with MQSeries for Mac OS MQI applications. Environment variables are viewed, set, changed or cleared using the MQSetup Control Panel.

Equivalents are provided for the following environment variables:

Environment Variable	MQSetup Control Panel Equivalent
MQDATA	User Directory Button.
MQCHLLIB	ChLib Folder button
MQCHLTAB	ChLib Filename field.
MQNAME	NOT SUPPORTED. This parameter is used to define the local NetBIOS name. Mac OS does not support NetBIOS.
MQ_PASSWORD	Password field.
MQSERVER	MQServer field.
MQCCSID	CCSID field.
MQTRACE	Trace On checkbox. Trace options field.
MQ_USER_ID	User ID field.

MQSeries for Mac OS uses default values for those variables that you have not set.

If you need to run multiple applications, you can specify a different configuration for each application. Note that only a single set of environment variables can be active at any one time.

MQDATA

This holds the path to the directory containing the trace and error and files.

> **Note**
>
> The MQSeries Client for Mac OS does not use a **qm.ini** file. All configuration information is stored in the standard Mac OS preferences folder.

The default location is the "MQFolder" folder located in the Mac OS System Folder.

To choose a new folder for the trace and error files, select the "User Directory" button in the MQSetup Control Panel. This will put up a standard Mac OS file dialog that allows you to select the new folder.

The trace and error files are:

MQ.TRC for trace messages.

AMQERR01.FDC for First Failure Data Capture messages.

AMQERR01.LOG for error messages.

An error message will always be added to the end of the log, so the files must be deleted periodically to avoid the files getting too large. At the time a record needs to be added to one of these files, if the file does not exist, it will be created.

These files are written in binary format. Use the **RUNMQFMT** application supplied with MQSeries to reformat these files into a readable form.

MQCHLLIB

This holds the path to the folder containing the client channel definition table, on the MQSeries client. If MQCHLLIB is not set, the path defaults to:

<Boot Volume>:System Folder:MQFolder

Consider keeping this folder on a central file server to make administration easier.

> **Note**
>
> If you are using MQSeries for MVS/ESA or OS/400 as your server, the client channel definition table file cannot be kept on these hosts.

To change the location of the client channel definition table, select the "ChLib Folder" button in the MQSetup Control Panel. This will put up a standard Mac OS file dialog that allows you to select an existing folder accessible by your computer.

MQCHLTAB

This specifies the name of the client channel definition table. The default file name is **AMQCLCHL.TAB**. This is found on the server machine, in the directory:

- For OS/2, Windows 3.1 and Windows NT:

  \mqm\qmgrs\queuemanagername\@ipcc

- For UNIX systems:

  /mqmtop/qmgrs/QUEUEMANAGERNAME/@ipcc

Note that QUEUEMANAGERNAME is case sensitive for UNIX systems. For MVS/ESA systems it is kept with all other object definitions on pageset zero.

To point to a different client channel definition table, enter the filename in the "ChLib Filename" field.

MQNAME

Not supported with MQSeries Client for Mac OS. NetBIOS does not apply to Mac OS systems.

MQSERVER

This is used to define a minimal channel. It specifies the location of the MQSeries server and the communication method to be used. Note that ConnectionName must be a fully qualified network name.

To change the MQSERVER variable, enter the following into the MQServer field:

ChannelName/TransportType/ConnectionName

If your application specifies a queue manager name on the MQCONN call, and this is not the queue manager name specified to the listener, the MQCONN call will fail. By default, for TCP/IP, MQSeries assumes that the channel will be connected to port 1414. You can change this by:

Adding the port number in brackets as the last part of the ConnectionName:

ChannelName/TransportType/ConnectionName(PortNumber)

All MQCONN requests then attempt to use the channel you have defined.

> Note
>
> The MQSERVER environment variable takes priority over any client channel definition pointed to by MQCHLLIB and MQCHLTAB, irrespective of any queue manager name specified in a MQCONN call.

MQCCSID

This specifies the coded character set number to be used and overrides the machine's configured CCSID.

To change the MQCCSID variable, enter the CCSID number into the CCSID field.

> Note
>
> The default CCSID on the Mac OS client is set to 850, a code page that is supported by most MQSeries servers.
>
> Please note that the Apple standard CCSID, 1275, is not yet implemented on most MQSeries servers. As a result, Apple specific characters transmitted by your application may not be translated correctly by default.
>
> A recent update to MQSeries for Windows NT 2.0 (CSD U200060) implements the Apple code pages. Check with Willow Technology for status of Apple code pages support on other MQSeries server platforms.

MQTRACE

This sets tracing on and off, as required. Unless you change it, tracing is turned off.

To turn tracing on, check the Trace On checkbox. To turn tracing off, uncheck the Trace On checkbox. If tracing is on, the trace information will be saved in the MQ.TRC file located in the user directory.

Trace options are set in the “Trace Options” field of the MQSetup Control Panel.

For example, to direct the communication flow trace entries to MQ.TRC file and overwrite the previous trace file each time the program runs, set the “Trace

Chapter 7

Defining channels

Creating one definition on the Mac OS client and the other on the server

Use MQSeries commands (MQSC) to define the server connection channel on the server. On MQSeries for OS/400 you can use MQSC and the CL commands. You are limited to defining one simple channel on the Mac OS client because MQSC is not available on a machine where MQSeries has been installed as an MQSeries client only.

On the server

Define a channel with your chosen name and a channel type of server connection. This channel definition is kept in the channel definition table associated with the queue manager running on the server.

For example:

DEFINE CHANNEL(CHAN1) CHLTYPE(SVRCONN) TRPTYPE(TCP) + DESCR('Server connection to Client_1')

On the MQSeries client

You cannot use MQSC on the MQSeries client. However, when you require a simple channel definition, without specifying all the attributes, you can use a single environment variable, MQSERVER (see Chapter 6, “Using MQSeries environment variables (MQSetup Control Panel).

A simple channel may be defined on Mac OS as follows:

Set the MQServer variable in the MQSetup Control Panel to:

ChannelName/TransportType/ConnectionName

ChannelName must be the **same** name as defined on the server.

TransportType must be **TCP**.

The **ConnectionName** is the name of the server machine or its IP address.

For example:

CHAN1/TCP/MCID66499

or:

CHAN1/TCP/9.20.4.56

On the MQSeries client, all MQCONN requests then attempt to use the channel you have defined.

> Note
>
> The MQSERVER environment variable takes priority over any client channel definition pointed to by MQCHLLIB and MQCHLTAB.

Canceling MQSERVER: To nullify MQSERVER and return to the client channel definition table pointed to by MQCHLLIB and MQCHLTAB, clear (delete) the entry on the MQServer field of the MQSetup Control Panel.

Creating both definitions on the server

On the server machine use MQSeries commands (MQSC) to define the channel. For more details about the MQSC, refer to the MQSeries Command Reference.

On the server

Define the server connection and then define the client connection.

Defining the server connection

On the server machine, define a channel with your chosen name and a channel type of server connection.

For example:

DEFINE CHANNEL(CHAN2) CHLTYPE(SVRCONN) TRPTYPE(TCP) + DESCR('Server connection to Client_2')

This channel definition is kept in the channel definition table associated with the queue manager running on the server.

Defining the client connection

Also on the server machine, define a channel with the **same** name and a channel type of client connection.

The connection name (CONNAME) must be stated. This is the TCP/IP machine name or network address of the server machine. It is a good idea to specify the queue manager name (QMNAME) to which you want your MQSeries application, running on the Mac OS client, to connect.

For example:

DEFINE CHANNEL(CHAN2) CHLTYPE(CLNTCONN) TRPTYPE(TCP) + CONNAME(9.20.4.26) QMNAME(QM2) DESCR('Client connection from Client_2')

For non-MVS/ESA systems this channel definition is kept in the client channel definition table associated with the queue manager running on the server. This file is called AMQCLCHL.TAB and is in the directory:

- For OS/2, Windows 3.1 and Windows NT:

  \mqm\qmgrs\queuemanagername\@ipcc

- For UNIX systems:

  /mqmtop/qmgrs/QUEUEMANAGERNAME/@ipcc

Note that QUEUEMANAGERNAME is case sensitive for UNIX systems. For MVS/ESA systems it is kept with all other object definitions on pageset zero.

On the MQSeries client

On the MQSeries client machine, use the environment variables MQCHLLIB and MQCHLTAB to allow the MQSeries application to access the client channel definition table on the server (not a server on OS/400 or MVS/ESA).

MQCHLLIB specifies the path to the folder containing the channel definition file. If not specified, the default used is <Boot Volume>:System Folder:MQFolder.

MQCHLTAB specifies the name of the file to use. If not specified, the default client channel definition table name (AMQCLCHL.TAB) is used.

To set the environment variables on Mac OS, enter **AMQCLCHL.TAB** in the Channel Table field of the MQSetup Control Panel.

In many cases the MQCHLLIB and MQCHLTAB variables might be used to point to a client channel definition table on a file server that is used by many MQSeries clients.

Alternatively, or if this is not possible, you can copy the client channel definition table, AMQCLCHL.TAB (a binary file) onto the Mac OS client machine and again use MQCHLLIB and MQCHLTAB to specify where the client channel definition table is.

On MVS/ESA, use the COMMAND function of the CSUTIL utility to make a client channel definition file that can then be downloaded to the client machine using a file-transfer program. For details see the MQSeries for MVS/ESA System Management Guide.

If you use FTP to copy the file, remember to type bin to set binary mode; do not use the default ASCII mode.

> **Note**
>
> The MQCHLLIB and MQCHLTAB environment variables are honored by the MQSeries commands when defining client connection channels. Therefore, for client connection channels only, you can use the MQCHLLIB and MQCHLTAB environment variables to override the default name and location, or both, of the generated client channel definition table.

The client channel definition pointed to by MQCHLLIB and MQCHLTAB may be overridden by the MQSERVER environment variable.

Chapter 8

Using the message queue interface (MQI)

When you write your MQSeries application, you need to be aware of the differences between running it in an MQSeries client environment and running it in the full MQSeries queue manager environment.

This chapter explains the things to consider with respect to Mac OS clients.

Limiting the size of a message

The maximum message length in a channel definition can be used to limit the size of a message allowed to be transmitted along a client connection. If any attempt is made by an MQSeries application to use the MQPUT call or the MQGET call with a message larger than this, an error code is returned to the application.

The maximum message size that can be specified on Mac OS is 4 MB (4,194,304 bytes).

Choosing client or server coded character set identifier (CCSID)

The data passed across the MQI from the application to the client stub should be in the local CCSID (coded character set identifier), encoded for the MQSeries client.

If the connected queue manager requires the data to be converted, this will be done by the client support code.

The client code will assume that the character data crossing the MQI in the client is in the CCSID configured for that machine. If this CCSID is an unsupported CCSID or is not the required CCSID, it can be overridden with the MQCCSID environment variable, for example:

- Set the CCSID field in the MQSetup Control Panel to 850.

Set this in the profile and all MQI data will be assumed to be in Code Page 850.

> **Note**
>
> This does not apply to application data in the message.

Controlling application in a Mac OS environment

A Mac OS MQSeries client as a full Mac OS compatible program.

Normally, when you run a non-MQSeries application, as soon as the application issues a request, control is not returned to that application until the request is fulfilled. This is because the Mac OS environment is a cooperative multi-tasking system.

However, the MQSeries client code overrides the locking of the machine and the application to enable you to start up more applications or work on something else until the MQI call has been answered. But, should an application attempt to issue a further MQI call before the previous one has been answered, the application will get a return code indicating that there is still a call in progress and the second call will fail.

Designing applications

When designing an application, consider what controls you need to impose during an MQI call because you need to ensure that the MQSeries application processing is not disrupted in any way.

Mac OS environment

To cooperate fully in the Mac OS multi-tasking environment, issuing an MQI call results in the client code executing a WaitNextEvent loop (for blocking calls only) on behalf of the application. If an application has accelerator keys defined, these will not function until the MQI call returns and control is returned to the WaitNextEvent loop of the application.

> **Note**
>
> There is only one way that an ongoing MQI call can be cancelled - by the application receiving a Quit AppleEvent message.

Using MQINQ

Some values queried using MQINQ will be modified by the client code. **CCSID** is set to the client CCSID, not that of the queue manager. MaxMsgLength is reduced if it is restricted by the channel definition. This will be the lower of:

- The value defined in the queue definition, or
- The value defined in the channel definition.

Using syncpoint coordination

Within MQSeries, one of the roles of the queue manager is syncpoint coordination within an application. If the application has been linked to a client stub, then it can issue MQCMIT and MQBACK, but there will be no syncpoint coordination.

Synchronization is limited to MQI resources only.

Triggering in the Mac OS client environment

Triggering is explained in detail in the MQSeries Application Programming Guide. When using a trigger monitor that runs in a MQSeries client environment, the application that is started by the trigger monitor must be in the same MQSeries client environment.

You must define the PROCESS definition on the server, as this is associated with the queue that has triggering set on.

The MQSeries Client for Mac OS contains two components to support Triggering. The first is the MQTrigger Control Panel which allows you to turn triggering on or off and to specify the Initiation Queue and Queue Manager names. The second component is the MQ Trigger Monitor itself, a Background Only Application, which is controlled by the MQTrigger Control Panel. The MQ Trigger Monitor is installed in the "MQTrigger §" folder located in the "MQSeries Client" installation folder.

There are two methods for starting applications using a trigger. In the first, the PROCESS definition on the MQSeries server specifies a fully qualified or relative pathname and application name to start (i.e. <volume>:<folder>:<folder>:application name or ::<folder>:application name). The Trigger Monitor will look for the application in the designated folder. The second, and more convenient, method is to define just the application name in the PROCESS definition, and place either the application, or an alias to it in the "MQTrigger §" folder. The Trigger Monitor will launch the application pointed to by the alias.

If the MQ Trigger Monitor is unable to find or launch the requested application, the message will be place in the system defined Dead Letter Queue.

The default is Initiation Queue name is SYSTEM.DEFAULT.INITIATION.QUEUE on the default queue manager. It

calls programs for the appropriate trigger messages. This trigger monitor supports the default application type.

The command string, built by the trigger monitor, is as follows:

1. The applicid from the relevant PROCESS definition
2. The MQTMC2 structure, enclosed in quotes, as got from the initiation queue
3. The envrdata from the relevant PROCESS definition

applicid is the name of the program to run.

The parameter passed is the MQTMC2 character structure. A command string is invoked which has this string, exactly as provided, in 'quotes', in order that the system command will accept it as one parameter.

Note

The trigger monitor will not look to see if there is another message on the initiation queue until the completion of the application it has just started.

Note

There can only be a SINGLE Trigger Monitor and consequently triggered application running at a time.

Chapter 9

Building applications for MQSeries clients

If an application is to run in a Mac OS client environment you can write it in C or C++. It must be linked with the appropriate library.

This chapter lists points to consider when running an application in a Mac OS client environment, and describes how to link your application code with the MQSeries client code.

Running applications in the MQSeries Client for Mac OS environment

You can run an MQSeries application in both a full MQSeries environment and in an MQSeries client environment without changing your code, providing:

- It does not need to connect to more than one queue manager concurrently
- The queue manager name is not prefixed with an asterisk (*) on an MQCONN call

> **Note**
>
> You must link your application with the Mac OS PowerPC or 68K library as appropriate.

When working in the MQSeries client environment, remember:

- Each application running in the MQSeries client environment has its own connections to servers. It will have one connection to every server it requires, a connection being established with each MQCONN call the application issues.

- An application sends and gets messages synchronously.
- All data conversion is done by the server.
- Triggering in the MQSeries Client for Mac OS environment is supported.
- Messages sent by MQSeries applications running on MQSeries clients contribute to triggering in exactly the same way as any other messages, and they can be used to trigger programs on the server.

Channel exits

The channel exits available to the MQSeries Client for Mac OS environment are:

- Send exit
- Receive exit
- Security exit

These exits are available at both the client and server ends of the channel.

Remember, exits are not available to your application if you are using the MQSERVER environment variable defined in the MQServer field of the MQSetup Control Panel. Exits are explained in the MQSeries Distributed queuing Guide.

The send and receive exit work together. There are several possible ways in which you may choose to use them:

- Segmenting and reassembling a message
- Compressing and decompressing data in a message
- Encrypting and decrypting user data
- Journaling each message sent and received

You can use the security exit to ensure that the MQSeries client and server machines are correctly identified, as well as to control access to each machine.

Linking C applications with the MQSeries client code

Having written your MQSeries application that you want to run on the MQSeries Client for Mac OS, you must link it to a queue manager. You do this using the client library file, which gives you access to queue managers on a different machine.

MQSeries provides a client library file for both PowerPC and 68k environments:

PowerPC: mqic.dll

68k: mqic68k.dll

Chapter 10

Running applications on MQSeries Client for Mac OS

This chapter explains the various ways in which an application running in an Mac OS client environment can connect to a queue manager. It covers the relationship of the MQSERVER environment variable (accessed using the MQSetup Control Panel), and the role of the client channel definition file created by MQSeries.

When an application running in an MQSeries client environment issues an MQCONN call, the client code identifies how it is to make the connection:

1. If the MQSERVER environment variable is set, the channel it defines will be used.

2. If the MQCHLLIB and MQCHLTAB environment variables are set, the client channel definition table they point to will be used.

3. Finally, if the environment variables are **not** set, the client code searches for a channel definition table whose path and name are established from the <Boot Volume>:System Folder:MQFolder. If this fails, the client code will use the paths:

- OS/2: rootdrive:mqm\amqclchl.tab

- UNIX systems: /var/mqm/amqclchl.tab

- Windows NT: rootdrive:mqm\amqclchl.tab

  where rootdrive is obtained from the Software\IBM\MQSeries\CurrentVersion registry entry under HKEY_LOCAL_MACHINE. This value is established when the

MQSeries client software is installed. If it is not found a value of 'C' is used for rootdrive.

Notes

1. If the client code fails to find any of these, the MQCONN call will fail.

2. The channel name established from either the first segment of the MQSERVER variable or from the client channel definition table, must match the SVRCONN channel name defined on the server for the MQCONN call to succeed.

3. See "Migrating from MQSeries for OS/2 V2.0 and MQSeries for AIX V2.1 or V2.2" in the MQSeries Clients reference if you receive a MQRC_Q_MGR_NOT_AVAILABLE return code from your application with an error message in the error log file of AMQ9517 - File damaged.

Using MQSERVER

If you use the MQSERVER environment variable to define the channel between your MQSeries client machine and a server machine, this is the only channel available to your application and no reference is made to the client channel definition table. In this situation, the 'listening' program that you have running on the server machine determines the queue manager that your application will connect. It will be the same queue manager as the listener program is connected to.

If the MQCONN request specifies a queue manager other than the one the listener is connected to, the MQCONN request fails with return code MQRC_Q_MGR_NAME_ERROR.

Using DEFINE CHANNEL

If you use the MQSC **DEFINE CHANNEL** command, the details you provide are placed in the client channel definition table. It is this file that the client code accesses, in channel name sequence, to determine the channel an application will use.

The contents of the Name parameter of the MQCONN call determines what processing will be carried out at the server end.

Role of the client channel definition table

Refer to Chapter 11 of the MQSeries Clients reference for a detailed explanation of client channel definition tables and how they work.

Chapter 11

HyperMQ interface from HyperCard

This chapter details the use of HyperMQ™, the MQI interface from HyperCard.

Adding HyperMQ to your application

HyperMQ is shipped as a code resource in a HyperCard stack called HyperMQ.stack. To add HyperMQ to your application, you need to use a resource editor (such as ResEdit) to copy the XCMD resource from HyperMQ.stack to your application stack.

HyperMQ.stack also contains sample code showing the basic operations of MQSeries in a HyperCard environment.

The HyperMQ programming interface

All HyperMQ calls follow the following format:

mqi {callName} [parm1] [parm2] …

where {} indicates a variable string and [ ] indicates an optional parameter.

> **Note**
>
> All values including “mqi” are case sensitive.

The values for {callName} are:

MQCONN - connect to a Queue Manager

MQDISC - disconnect

MQOPEN - Open an object (generally a queue)

MQCLOSE - Close an object

MQGET - get a message from a Queue

MQPUT - put a message onto a Queue

MQCMIT – commit all messages since the last syncpoint

MQBACK – back out all messages since the last syncpoint

MQPUT1 - do an open, put a single message, and close a Queue

MQINQ – inquire on the attributes of an object

MQSET – set the attributes of an object

SETOD - set Object Descriptor Options

SETMD - set Message Descriptor Options

SETPMO - set Put Message Options

SETGMO - set Get Message Options

SETENV - set Environment Variables

HyperMQ return messages

If an error occurs, the XFCTN will return a string beginning with "ERROR" followed by an error message. Otherwise it will return null or a proper return value, depending on the call.

Error messages returned include:

ERROR: unknown command
ERROR: unknown variable name
ERROR: unknown constant value
ERROR: too few parameters
ERROR: not connected
ERROR: already connected
ERROR: open object limit exceeded
ERROR: object not found
ERROR: missing queue name
ERROR: {nnnn} where {nnnn} represents an MQSeries Reason Code. Refer to the MQSeries Application Programming Reference for explanation of the reason codes.

HyperMQ MQI calls

The following is a list of the each supported MQI call and its parameters.

MQCONN

mqi MQCONN, {QUEUE_MANAGER_NAME}

Returns error string or null. Only one connection can be open at a time.

MQDISC

mqi MQDISC

Returns error string or null.

MQOPEN

OI = **mqi MQOPEN,** {QUEUE_ NAME}| MQOD[, {MQOO_*}[, {MQOO_*}]…

To open a queue on the local queue manager, just pass the queue name. Otherwise, set up the Object Descriptor parameters using the SETOD call, and pass MQOD as the object name. Returns an Object Identifier to be used in subsequent calls. Maximum number of open objects is 10.

MQOO_* represents one or more Open Options.

MQCLOSE

mqi MQCLOSE, {OI}[, MQCO_DELETE | MQCO_DELETE_PURGE]

{OI} is the string object identifier returned from the MQOPEN call. Only one of the delete options can be present. Omitting both is equivalent to the MQCO_NONE option.

MQGET

message = **mqi MQGET,** {OI}[, MQMD] [MQGMO]

If the string MQMD is present, then the call uses Message Descriptor set up by the SETMD call, otherwise a default MD structure is used. If the string MQGMO is present, then the call uses the Get Message Options set up by the SETGMO call, otherwise a default GMO structure is used.

Note

1. The maximum message size that can be get or put in HyperMQ is 512,000 bytes.

MQPUT

mqi MQPUT, {OI}, {message}[, MQMD][, MQPMO]

If the string MQMD is present, then the call uses Message Descriptor set up by the SETMD call, otherwise a default MD structure is used. If the string MQPMO is present, then the call uses the Put Message Options set up by the SETPMO call, otherwise a default PMO structure is used. As usual in HyperCard, the message must be surrounded by quotes if it contains any blanks.

MQCMIT

mqi MQCMIT

Commit all the message gets and puts since the last syncpoint.

MQBACK

mqi MQBACK

Back out all the message gets and puts since the last syncpoint.

MQPUT1

mqi MQPUT1, {QUEUE_NAME}|MQOD, {message}[, MQMD][, MQPMO]

This call performs an open, put, and close on the specified queue name. MQOD can be substituted for QUEUE_NAME, as in the MQOPEN call. The other parameters are similar to the MQPUT call.

MQINQ

mqi MQINQ, {OI}, {Attribute}

Back out all the message gets and puts since the last syncpoint.

MQINQ/MQSET Notes

1. {Attribute} is a case sensitive constant. Refer to the CMQC.H file for exact case definition.
2. Refer to the MQSeries Application Programming Reference for specifics on which attributes can be queried or set.
3. The HyperMQ interface is different from other language interfaces for MQINQ and MQSET. Other interfaces allow the programmer to query or set arrays or multiple attributes. In HyperMQ, only a single attribute may be set at a time. If you need multiple attributes, multiple calls are required.

MQSET

mqi MQSET, {OI}, {Attribute}, {value}

Back out all the message gets and puts since the last syncpoint.

HyperMQ interface to set up MQI call structures

Where possible, the varName parameters and values for the SETOD, SETMD, SETPMO, and SETGMO correspond to the structure field name definitions found in CMQC.H..

> Notes
>
> 1. {varName} is case sensitive constant. Refer to the CMQC.H file for exact case definition.
> 2. If you need to set multiple options, it is necessary to issue multiple SETxxx calls.

Use a {varname} of DEFAULT with no {value} parameter to reset the structure to its default state.

SETOD

mqi SETOD, {varName}, {value}

SETMD

mqi SETMD, {varName}, {value}

SETPMO

mqi SETPMO, {varName}, {value}

SETGMO

mqi SETGMO, {varName}, {value}

SETENV

mqi SETENV, {varName}, {value}

The following environment variables which are usually configured using the MQSetup Control Panel can be changed from within a HyperCard application:

USERID

Corresponds to the “User ID” Attribute of the MQSetup Control Panel.

USERPWD

Corresponds to the “Password” Attribute of the MQSetup Control Panel.

MQSERVER

Corresponds to the “MQServer” Attribute of the MQSetup Control Panel.

CONFIGNAME

The name of one of the MQSeries Configurations defined in the MQSetup Control Panel.

Note

. The MQI connection must be closed before changing any of these environment variables.

Chapter 12

Solving Problems

MQSeries for Mac OS specific return codes, error logs, and error messages are discussed.

Error messages with MQSeries clients

When an error occurs with an MQSeries client system, error messages are put into the error files associated with the server, if possible. If the error cannot be placed there, the MQSeries client code attempts to place the error message in an error log in the root directory of the MQSeries client machine.

MQSeries clients on Mac OS systems

Error messages for MQSeries clients on Mac OS systems are placed in the error logs in the same way that they are for the respective MQSeries server systems. These files appear in <Boot Volume>:System Folder:MQFolder, unless specified differently by the MQDATA environment variable.

The log file **AMQERR01.LOG** is held in <Boot Volume>:System Folder:MQFolder unless the MQDATA environment variable is used to override the default

How to read the error log and FFDCs

The MQ Trace Formatter program reformats the trace, error and FFDC files. MQ Trace Formatter requires you enter the name of the file to be processed. The output is displayed to the application window. The contents can be saved to a file, printed or both.

To print the output, select Print from the File Menu.

Your application program should handle any MQI reason codes to allow your program to end in a controlled manner as there is no MQI error handling within the product.

How to get a trace file

Use the MQTRACE environment variable to set tracing. You can further define the use of this file by the optional use of flags:

c Trace the communications flow

m Do not query the configuration of the machine your application is running on. Use this option if exceptions occur when normal tracing is switched on.

W Write a new instance of the trace file for each program. If this is not set, the trace entries continue to be added to a single trace file.