Willow Technology

MQSeries® Client for MPE/iX™
Version 2-01

MQSeries Clients Addendum

MQSERIES CLIENT FOR MPE/IX

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, Ltd.

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

Hewlett-Packard and MPE/iX are trademarks or registered trademarks of Hewlett-Packard, Inc.

Willow Technology and the Willow logo are trademarks of Willow Technology, Inc.

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

Table of Contents

Trademarks ..... *ii*

Support for MQSeries Client for MPE/iX **1**

Communications **2**

MQSeries Client for MPE/iX: hardware and software required *3*

Machine requirements **3**

Operating System requirements **3**

Compilers for MQSeries applications on MPE/iX clients **3**

Installing Software From DAT Tape **4**

The verification scenario **6**

Security **6**

Setting up the server **6**

Setting up the MQSeries client **6**

Define a client-connection channel, using MQSERVER **7**

Putting a message on the queue (POSIX) **7**

Getting the message from the queue (POSIX) **8**

Putting a message on the queue (Native Mode) **8**

Getting the message from the queue (Native Mode) **9**

Ending verification **9**

Authentication **11**

User ID and password **11**

POSIX Mode 11

Native Mode 11

Access Control **12**

MQCHLLIB **14**

POSIX: 15

Native Mode: 15

MQCHLTAB **15**

POSIX: 15

Native Mode: 15

MQSERVER **15**

POSIX:..... 16
Native Mode:..... 16

MQCCSID.....16
POSIX:..... 16
Native Mode:..... 16

Creating one definition on the MPE/iX client and the other on the server 18
On the server..... 18
On the MQSeries client..... 18
POSIX:..... 18
Native Mode:..... 18
POSIX:..... 19
Native Mode:..... 19

Creating both definitions on the server.....20
On the server..... 20
Defining the server connection..... 20
Defining the client connection..... 20
On the MQSeries client..... 21
POSIX:..... 21
Native Mode:..... 21

Limiting the size of a message.....23

Choosing client or server coded character set identifier (CCSID).....23
POSIX:..... 23
Native Mode:..... 23

Controlling application in an MPE/iX environment.....24

Designing applications.....24

Using MQINQ.....24

Using syncpoint coordination.....24

Triggering in the MPE/iX client environment.....24

Running applications in the MQSeries environment.....26

Channel exits.....27

Linking C applications with the MQSeries client code.....27
POSIX:..... 27
Native Mode:..... 27

Using MQSERVER.....29

Using DEFINE CHANNEL.....29

Role of the client channel definition table.....29

Error messages with MQSeries clients.....30

Chapter 1

Preparing for Installation

The information in this manual provides information specific to the HP3000 MPE/iX 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 MPE/iX 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 MPE/iX

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 MPE/iX, subject to differences in coded character set identifier (CCSID) and communications protocol.

Note

Make sure that code conversion from the CCSID of the MPE/iX client is supported by the server. See the Language support tables in the MQSeries Application Programming Reference.

These MQSeries products:

MQSeries for SCO OpenServer Version 2.x or later

MQSeries for UnixWare Version 2.x or later

MQSeries for IRIX Version 2.x or later

MQSeries for AIX Version 2.2.1 or later

MQSeries for AT&T GIS UNIX Version 2.2

MQSeries for HP-UX Version 2.2.1 or later

MQSeries for OS/2 Version 2.0.1 or later

MQSeries for Windows NT Version 2.0 or later

MQSeries for MVS/ESA Version 1 Release 1.4 or later

MQSeries for OS/400 Version 3 Release 2 or later

MQSeries for SINIX and DC/OSx Version 2.2

MQSeries for SunOS Version 2.2

MQSeries for Sun Solaris Version 2.2 or later

can accept connection from an MQSeries Client for MPE/iX.

Communications

TCP/IP is the only transmission protocol supported by the MQSeries Client for MPE/iX software.

MQSeries Client for MPE/iX: hardware and software required

Machine requirements

An MQSeries client can run on any computer running a supported version of the HP3000 MPE/iX operating system 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.

Operating System requirements

The following MPE/iX versions are supported:

- MPE/iX 5.5 with PowerPatch 7 or later

Compilers for MQSeries applications on MPE/iX clients

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

Native Mode:

- HP COBOL II/iX programs using the 1985 COBOL (COB85) Compiler

POSIX:

- C programs using the c89 Compiler

Chapter 2

Installing the MQSeries Client for MPE/iX

The MQSeries Client for MPE/iX is developed and supported by Willow Technology under license from IBM. It is licensed by for use on a single computer, and is distributed on DAT Tape.

Before installing the software, please consult the hard copy "README" and "Release Notes" included with the software package for the latest information, known problems and fixes.

Installing Software From DAT Tape

1. Mount the product tape on the HP3000 DAT drive.

2. Logon as MANAGER.SYS.

3. Restore all files on the product tape. For example:
   :file t;dev=7
   :restore *t;@.@.@;show;create;creator

4. See the README.PUB.MQM for a description of the MPE/iX domain components, and /MQM/PUB/opt/mqm/README and /MQM/PUB/opt/mqm/Release_Notes for details on POSIX components.

5. Remove the product tape from the DAT drive.

6. Done!

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 MPE/iX client 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 Client for MPE/iX software and supplied files have been installed on the HP3000 system 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 both the machines.

Note

Compiled POSIX C samples amqsputc and amqsgetc are included in the "**/MQM/PUB/opt/mqm/samp**".folder.

Compiled Native Mode COBOL samples PMQ0PUT0 and PMQ0GET0 are in the **COBOL.MQM** group

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 MPE/iX client 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 MPE/iX 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 MPE/iX, 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.

Define a client-connection channel, using MQSERVER

Create a client-connection channel by setting the MQSERVER environment variable. For applications linked with the *POSIX* libraries (libmqicg.a or libmqicbg.a), enter the following command from the **POSIX** shell:

export MQSERVER=CHANNEL1/TCP/server-address(port)

For applications linked with the **Native Mode** library (COBMQXL), enter the following command:

:setvar MQSERVER "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.

Imporant Note

The second parameter, TCP, is case sensitive. It **MUST** be entered in upper case!

Putting a message on the queue (POSIX)

On the MQSeries client workstation, put a message on the queue using the amqsputc sample program:

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

   amqsputc QUEUE1 qmgr

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

2. The following message is displayed:

   Sample AMQSPUT0 start

   target name is QUEUE1

3. Type some message text and then press Enter **twice**.

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

Sample AMQSPUT0 end

5. The message is now on the queue.

Getting the message from the queue (POSIX)

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

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

   amgsgetc QUEUE1 qmgr

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

2. The message on the queue is displayed and then deleted from the queue.

Putting a message on the queue (Native Mode)

On the MQSeries client workstation, put a message on the queue using the amqsputc sample program:

1. Change to the group containing the sample COBOL programs, and then enter the following command:

   : **RUN PMQ0PUT0;XL="COBMQXL"**

2. The following message is displayed:

   AMQ0PUT0 start

   Please enter the name of the target queue

3. The following message will be displayed on the HP3000 console:

   ?hh:mm/#Snnn/<nn>/Awaiting REPLY for COBOL ACCEPT statement. (MAX CHARS.=31)? Where <nn> is the message number.

4. At the console, enter the queue name using the <Control-A>REPLY command:

   =REPLY <nn>,QUEUE1

5. The client workstation will display the following message:

   Please enter the message(s)

6. Type some message text and then press Enter **twice**.

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

   AMQ0PUT0 end

   END OF PROGRAM

8. The message is now on the queue.

Getting the message from the queue (Native Mode)

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

1. Change to the group containing the sample COBOL programs, and then enter the following command:

   :**RUN PMQ0GET0;XL="COBMQXL"**

2. The following message is displayed:

   AMQ0GET0 start

   Please enter the name of the source queue

3. The following message will be displayed on the HP3000 console:

   ?hh:mm/#Snnn/<nn>/Awaiting REPLY for COBOL ACCEPT statement. (MAX CHARS.=31)? Where <nn> is the message number.

4. At the console, enter the queue name using the <Control-A>REPLY command:

   =**REPLY <nn>,QUEUE1**

5. The message on the queue is displayed and then deleted from the queue.

6. After a delay of approximately 15 seconds, th e following messages are displayed:

   no more messages

   AMQ0GET0 end

   END OF PROGAM

Ending verification

The verification process is now complete.

Chapter 4

Configuration

MQSeries Client for MPE/iX software only supports TCP/IP. All that is required is that TCP/IP is initialized on the MPE/iX system.

Refer to your MQSeries documentation for TCP/IP configuration and initialization requirements for your MQSeries server.

Chapter 5

Setting up MQSeries Client for MPE/iX 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 MPE/iX clients.

User ID and password

If a security exit is not defined on an MQSeries Client for MPE/iX, 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.

POSIX Mode

1. Type **export MQ_USER_ID=<myuserid>** (without the <.>).
2. Type **export MQ_PASSWORD=<mypassword>** (without the < >).

Native Mode

1. Type :**setvar MQ_USER_ID "myuserid"** (including the "").
2. Type :**setvar MQ_PASSWORD "mypassword"** (including the "").

Access Control

Access control in MQSeries is based upon the user identifier associated with the process making MQI calls. For MPE/iX 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)
- Default MCAUSER value (from SYSTEM.DEF.SVRCONN)
  This value is used if no value is specified for MCAUSER when the server channel is defined.

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.	
Not Set or Set	Not Set	TCP/IP: User ID from inetd.conf entry.	2

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 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

This chapter describes the environment variables that you can use with MQSeries Client for MPE/iX MQI applications:

- MQCHLLIB
- MQCHLTAB
- MQ_PASSWORD
- MQSERVER
- MQCCSID
- MQ_USER_ID

MQSeries uses default values for those variables that you have not set. Update your system profile to make a permanent change; issue the command from the command line to make a change for this session only, or if you want one or more variables to have a particular value dependent on the application running, you can add commands to a command script file used by the application.

Note that only a single set of environment variables can be active at any one time.

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:

/var/mqm/

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, type:

POSIX:

export MQCHLLIB=pathname

Native Mode:

:setvar MQCHLLIB "HFS pathname"

where "HFS pathname" is the POSIX pathname to the channel table, i.e. "/var/mqm".

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, type:

POSIX:

export MQCHLTAB=filename.

Native Mode:

:setvar MQCHLTAB "filename"

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, type:

POSIX:

export MQSERVER=ChannelName/TCP/ConnectionName

Native Mode:

:setvar MQSERVER "ChannelName/TCP/ConnectionName"

> **Imporant Note**
>
> The second parameter, TCP, is case sensitive. It **MUST** be entered in upper case!

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 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/TCP/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, type:

POSIX:

export MQCCSID=number

Native Mode:

:setvar MQCCSID "number"

Note

The default CCSID on the MPE/iX client is set to 850, a code page that is supported by most MQSeries servers.

Chapter 7

Defining channels

Creating one definition on the MPE/iX 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 MPE/iX 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 MPE/iX as follows:

POSIX:

```
export MQSERVER=ChannelName/TCP/ConnectionName
```

Native Mode:

```
:setvar MQSERVER "ChannelName/TCP/ConnectionName"
```

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

The second parameter, transport type, must be **TCP** (in upper case).

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.

Cancelling MQSERVER: To nullify MQSERVER and return to the client channel definition table pointed to by MQCHLLIB and MQCHLTAB, enter:

POSIX:

unset MQSERVER

Native Mode:

:deletevar MQSERVER

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 MPE/iX 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, and Windows NT(versions prior to MQSeries V5.1):

  \mqm\qmgrs\queuemanagername\@ipcc

- For Windows NT(MQSeries version V5.1):

  \Program Files\MQSeries\qmgrs\queuemanagername\@ipcc

- For UNIX systems:

/mqmtop/qmgrs/QUEUEMANAGERNAME/@ipcc

Note

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 directory containing the channel definition file. If not specified, the default used is *DefaultPrefix* from the mqs.ini file.

Note

The channel definition file is not automatically created in the *DefaultPrefix* directory. If you do not specify the MQCHLLIB environment variable, you will have to copy the channel definition file that you want the client to use to the *DefaultPrefix* directory.

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 MPE/iX, type:

POSIX:

export MQCHLTAB=AMQCLCHL.TAB

Native Mode:

:setvar MQCHLTAB "AMQCLCHL.TAB"

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 MPE/iX 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.

Note

If you use *ftp* to copy the file, remember to set **binary** mode; do not use ascii or labels (tenex) 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 MPE/iX 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 MPE/iX 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:

POSIX:

```
set MQCCSID=850
```

Native Mode:

```
setvar MQCCSID "850"
```

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

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

Controlling application in an MPE/iX environment

The MQSeries client enables you to start up more applications or work on something else until an 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.

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 MPE/iX 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 trigger monitor provided runs in the MPE/iX POSIX environment only. To run it, type:

runmqtmc [-m QmgrName] [-q InitQ]

in the POSIX shell.

The default 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 and is the same as *runmqtrn* except that it links the client libraries.

The command string, passed by the queue manager on the server to the trigger monitor on the MPE/iX client, is built as follows:

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.

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. If the application has a lot of processing to do, this may mean that the trigger monitor cannot keep up with the number of trigger messages arriving. You have two options:

- Have more trigger monitors running
- Run the started applications in the background

If you choose to have more trigger monitors running you have control over the maximum number of applications that can run at any one time. If you choose to run applications in the background, there is no restriction imposed by MQSerie. on the number of applications that can run.

To run the started POSIX application in the background in an MPE/iX system, you must put an '&' at the end of the `envrdata` of the PROCESS definition.

Chapter 9

Building applications for MQSeries clients

If an application is to run in an MPE/iX environment, you can write it in COBOL (Native Mode) or C (POSIX). It must be linked with the appropriate library. It is also possible to call the C library from Native Mode applications.

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

Running applications in the MQSeries 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.

However, the libraries at link-edit time determine the environment your application must run in.

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 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 MPE/iX 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. 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, 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.

POSIX:

C library:	**libmqicg.a**
COBOL library:	**libmqibcg.a**

Native Mode:

COBOL library:	**;XL="COBMQXL"**

Chapter 10

Running applications on MPE/iX clients

This chapter explains the various ways in which an application running in an MPE/iX client environment can connect to a queue manager. It covers the relationship of the MQSERVER environment variable, 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 *DefaultPrefix* in the mqs.ini file. 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

Solving Problems

MQSeries client for MPE/iX 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 on the MQSeries client machine.

On MPE/iX, the error log can be found in the POSIX filesystem at /var/mqm/AMQERR01.LOG.

To view the contents of the log file, use the **/MQM/PUB/opt/mqm/bin/runmqfmt** utility. Runmqfmt must be run from the POSIX shell, and it expects to read the error log at /var/mqm/AMQERR01.LOG.