™ Willow Technology

MQSeries Client for UnixWare
Version 5

# MQSeries Clients Addendum

MQSERIES CLIENT FOR UNIXWARE

# MQSeries Clients Addendum

Phone +1.408.296.7400 • Fax +1.408.296.7700
email: info@willowtech.com
www.willowtech.com

## Trademarks

# Table of Content

Chapter 1

# Preparing for Installation

*The information in this manual provides information specific to the UnixWare MQI client, and is intended to be read in conjunction with the IBM MQSeries Clients reference; IBM publication number GC33-1632-05 (or later edition).*

This chapter details the platform support and the communications protocol support for UnixWare clients only.

For your server platform hardware and software requirements, see the MQSeries System Administration (MQSeries Version 5 products) and the relevant System Management Guide for other platforms.

For capacity planning information, see the MQSeries Planning Guide.

## Support for MQSeries Client for UnixWare

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

Note

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

These MQSeries products:

MQSeries for AIX Versions 2.2 or later

MQSeries for AT&T GIS UNIX Version 2.2

MQSeries for DYNIX/ptx Versions 1 or later

MQSeries for HP-UX Versions 2.2 or later

MQSeries for IRIX Versions 2.2 or later

MQSeries for Linux (Alpha Edition) Versions 5.2 or later

MQSeries for Linux (Intel Edition) Versions 5.2 or later

MQSeries for Linux (SPARC Edition) Versions 5.2 or later

MQSeries for MVS/ESA Version 1 Release 1.4 or later

MQSeries for OS/2 Versions 2.0.1 and 5.0

MQSeries for OS/400 Version 3 Release 2 or later

MQSeries for SCO OpenServer Versions 2.2 or later

MQSeries for SINIX and DC/OSx Version 2.2

MQSeries for SunOS Versions 2.2

MQSeries for Sun Solaris Versions 2.2 or later

MQSeries for UnixWare Versions 2.2 or later

MQSeries for Windows NT Versions 2 or later

can accept connection from an MQSeries for UnixWare client.

## Communications

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

## MQSeries Client for UnixWare: hardware and software required

### Machine requirements

An MQSeries client can run on any computer running a supported version of the UnixWare 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 UnixWare versions are supported:

- UnixWare 2.1.3 or later
- UnixWare 7.0.1 or later

### Compilers for MQSeries applications on UnixWare clients

The following compilers have been tested and are supported:

- UnixWare Software Development C and C++ Compilers
- UnixWare and OpenServer Development Kit 7 (UDK) C and C++ Compilers
- Micro Focus/Merant Object COBOL Developers Suite V4.1 or later.

Chapter 2

# Installing the MQSeries Client for UnixWare

The MQSeries Client for UnixWare client is developed and supported by Willow Technology under license from IBM. It is licensed for use in accordance with Willow's International Program License Agreement, License Information and Proof of Entitlement, and is distributed on CD-ROM.

Before installing the software, please consult the "README.1st" and "README " files located on the installation CD-ROM with for the latest information, known problems and fixes.

This chapter tells you how to install MQSeries Client for UnixWare and how to verify that your installation has been successful.

## Components you can install

When you install MQSeries Client for UnixWare, you can choose which components to install. The components are as follows:

| **Component** | **What it Is** |
|---|---|
| Man pages | man pages for MQI calls. |
| Sample programs | Sample source code and executable programs |
| UnixWare Client Libraries | MQI client runtime libraries, including English language messages. |
| German message catalog | German language messages |
| Spanish message catalog | Spanish language messages |
| French message catalog | French language messages |

| | |
|---|---|
| Italian message catalog | Italian language messages |

## Installation requirements

The installation requirements depend on which components you install and how much working space you need. This, in turn, depends on the number of queues that you use, the number and size of the messages on the queues, and whether the messages are persistent or not. You also require archiving capacity on disk, tape, or other media.

### Disk storage

These are the approximate storage requirements in the file system containing the **/opt** directory:

| | |
|---|---|
| • UnixWare Client Libraries | 8MB. |
| • Man pages: | 100K |
| • Sample programs: | 1MB |

Working data for MQSeries Client for UnixWare is stored by default in **/var/mqm**.

## Preparing for installation

For MQSeries Client for UnixWare the name of the installation directory is **/opt/mqm/**

This section guides you through some of the steps you must perform before you install MQSeries for UnixWare.

### Before installation...

Before you can install MQSeries Client for UnixWare you:

- must create a group with the name mqm
- must create a user ID with the name mqm.
- are recommended to create and mount a **var/mqm** file system.

After installation, this user ID (mqm) owns the directories and files that contain the resources associated with the product. This group and user must be defined for any machine on which the MQSeries software is to be installed.

For stand-alone machines, you can create the new user and group IDs locally. For machines administered in a network information services (NIS) domain, you can create the user and group IDs on the NIS master server machine.

## Installation

The MQSeries product is contained in the / subdirectory of the CD-ROM. The publications are installable from the installation program, but are separately stored in the **/pubs** subdirectory.

Carry out the following procedure:

1. Mount the CD-ROM as the **/cdrom** directory.

2. Perform a **pkgadd -d /cdrom/uwX_client50x.img** (where X = 2 for UnixWare 2 and 7 is for UnixWare 7; and x = the update number). Refer to the README file for specific install image names.

For information on pkgadd, refer to the man pages, or your UnixWare documentation..

Note

If you have previously installed MQSeries on your system, you need to remove the product using the pkgrm program.

If the product is present, but not installed correctly, you may need to manually delete the files and directories contained in **/var/mqm** and **/opt/mqm**

3. Install the MQSeries license key before attempting to use the product. For licensing information and details, please refer to the Release Notes that came with the product, as well as the /opt/mqm/README file.

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 UnixWare 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 5 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 UnixWare software and supplied files have been installed on UnixWare 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 samples amqsputc and amqsgetc are included in the **/opt/mqm/samp/bin/** folder.

## 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 Unix system; 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 UnixWare 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 UnixWare, 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 UnixWare, enter the following command:.

**export 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.

## Putting a message on the queue

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

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.

## Ending verification

The verification process is now complete.

Chapter

4

# Configuration

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

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

Chapter

5

# Setting up MQSeries for UnixWare client 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 9 of the MQSeries Clients reference. There are no special considerations for UnixWare clients.

## User ID and password

If a security exit is not defined on an MQSeries for UnixWare client, 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. Type **export MQ_USER_ID=<myuserid>** (without the <.>).
2. Type **export MQ_PASSWORD=<mypassword>** (without the < >).

## Access Control

Access control in MQSeries is based upon the user identifier associated with the process making MQI calls. For UnixWare 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<br>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.<br>For AS/400: Default User ID QMQM.<br>TCP/IP (non-MVS/ESA): User ID from inetd.conf entry. | 2,3,4 |
| Not Set<br>or Set | Not Set | TCP/IP: User ID from inetd.conf entry. | 2,3,4 |

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 MCAUSER is blank.

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

3. For TCP/IP on Windows NT, the value used is the User ID of the person who started the listener..

4. 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 for UnixWare MQI applications:

- MQCHLLIB
- MQCHLTAB
- MQ_PASSWORD (Refer to Chapter 5)
- MQSERVER
- MQCCSID
- MQ_USER_ID (Refer to Chapter 5)

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:

**export MQCHLLIB=pathname**

### 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**

- For VM/ESA:

  **GLOBALV SELECT CENV SETLP MQCHLTAB filename**

Note that **queuemanagername** is case sensitive for UNIX systems.

To point to a different client channel definition table, type:

**export MQCHLTAB=filename.**

Note

If you change this environment variable on an MQSeries server, MQSeries will not be able to find any client channel definition table you may have defined before. You must then move your old client channel definition table to the new location..

### 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.

Note

When the MQSERVER environment variable is used to define a client channel a MAXMSGL of 4 MB is used, so larger messages cannot flow across this channel. For larger messages a CLNTCONN channel must be defined using DEFINE CHANNEL, on the server, with MAXMSGL set to a larger value.

To change the MQSERVER variable, type:

**export MQSERVER=ChannelName/TCP/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/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:

**export MQCCSID=number**

Note

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

Chapter

7

# Defining channels

## Creating one definition on the UnixWare 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 UnixWare 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 UnixWare as follows:

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

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

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:

```
unset 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 UnixWare 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 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 UnixWare, type:

```
export 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 UnixWare 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 set binary mode; do not use 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 UnixWare clients.

## Limiting the size of a message

The maximum message length (MaxMsgLength) attribute of a queue manager is the maximum length of a message that can be handled by that queue manager. The default maximum message length supported depends on the platform you are using. Details are given in the MQSeries Application Programming Guide.

You can find out the value of MaxMsgLength for a queue manager by using the MQINQ call.

If the MaxMsgLength attribute is changed, no check is made that there are not already queues, and even messages, with a length greater than the new value. After a change to this attribute, applications and channels should be restarted in order to ensure that the change has taken effect. It will then not be possible for any new messages to be generated that exceed either the queue manager's MaxMsgLength or the queue's MaxMsgLength (unless queue manager segmentation is allowed).

The maximum message length in a channel definition limits the size of a message that you can transmit along a client connection. If an MQSeries application tries to use the MQPUT call or the MQGET call with a message larger than this, an error code is returned to the application.

## 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 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 a UnixWare 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 control within an application. If an application runs on an MQSeries client, then it can issue MQCMIT and MQBACK, but the scope of the syncpoint control is limited to the MQI resources.

Applications running in the full queue manager environment, on the server, can coordinate multiple resources (for example databases) via a transaction monitor. On the server you can use the Transaction Monitor supplied with the Version 5 MQSeries products, or another transaction monitor such as CICS. You cannot use a transaction monitor with a client application. The MQSeries verb MQBEGIN is not valid in a client environment.

## Using MQCONNX

MQCONNX can be used from a client but these options are ignored:

- MQCNO_STANDARD_BINDING
- MQCNO_FASTPATH_BINDING

MQCONN and MQCONNX on a client are equivalent calls. The actual call issued at the server depends on the server level and the listener configuration.

Chapter

9

# Building applications for MQSeries clients

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

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

## Running applications in the MQSeries Client for UnixWare 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 or MQCONNX call.

Note

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 or MQCONNX call the application issues.
- An application sends and gets messages synchronously.
- All data conversion is done by the server, but see also “MQCCSID”

- Triggering is supported (see "Triggering in the UnixWare client environment)

## Triggering in the UnixWare client environment

Triggering is explained in detail in the MQSeries Application Programming Guide.

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. The trigger monitor and the application to be started must be on the same system.

### Process definition

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

The process object defines what is to be triggered. If the client and server are not running on the same platform, any processes started by the trigger monitor must define `ApplType`, otherwise the server takes its default definitions (that is, the type of application that is normally associated with the server machine) and causes a failure.

For example, if the trigger monitor is running on a UnixWare client and wants to send a request to an Windows NT server, MQAT_ UNIX must be defined otherwise Windows NT uses its default definitions (that is, MQAT_WINDOWS_NT) and the process fails.

For a list of application types, see the MQSeries Application Programming Reference manual.

### Trigger monitor

The trigger monitor provided runs in the UnixWare client environment. To run it, type:

**runmqtmc [-m QmgrName] [-q InitQ]**

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 UnixWare 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 MQSeries on the number of applications that can run.

To run the started application in the background in a UnixWare system, you must put an ‘&’ at the end of the `envrdata` of the PROCESS definition.

## Channel exits

The channel exits available to the MQSeries Client for UnixWare 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. Exits are explained in the book MQSeries Intercommunication.

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.

#### Paths to exits

This applies to MQSeries clients on AIX, HP-UX, OS/2, Sun Solaris, Windows NT, and Windows 95 systems.

An `mqs.ini` file is added to your system during installation of the MQSeries client.

A default path for location of the channel exits on the client is defined in this file, using the stanza:

```
ClientExitPath:
   ExitsDefaultPath=<defaultprefix>/exits
```

Where `<defaultprefix>` is the value defined for your system in the DefaultPrefix stanza of the `mqs.ini` file.

When a channel is initialized, after an MQCONN or MQCONNX call, the `mqs.ini` file is searched. The ClientExitPath stanza is read and any channel exits that are specified in the channel definition are loaded.

### 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.

To link the client libraries on UnixWare, use the following libraries in your link command:

-lmqic -lmqmcs –lmqmzse –lld –lsocket -lnsl

### Linking C++ applications with the MQSeries client code

If you have a C++ application that you want to run in the client environment, you must recompile the programs to link them with the libimqb23uu.so and libimqc23uu.so C++ client libraries.

To link the C++ client libraries on UnixWare, use the following libraries in your link command:

-limqb23uu –limqc23uu -lmqmcs –lmqmzse –lld –lsocket -lnsl

## Linking COBOL applications with the MQSeries client code

If you have a COBOL application that you want to run in the client environment, you must recompile the programs to link them with the client library, libmqicb.so:

```
cob -xU <prog>.cbl -lmqicb -lmqic -lmqmcs –lmqmzse –lld
-lsocket -lnsl
```

**Note:** `-lmqicb` **mus**t come before `-lmqic` on the command line.

Chapter

10

# Running applications on UnixWare clients

This chapter explains the various ways in which an application running in a UnixWare 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 /var/mqm/AMQCLCHL.TAB.

Notes

1. If the client code fails to find any of these, the MQCONN or MQCONNX 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 or MQCONNX 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 or MQCONNX request specifies a queue manager other than the one the listener is connected to, the MQCONN or MQCONNX 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 or MQCONNX call determines what processing will be carried out at the server end.

## Role of the client channel definition table

Refer to Chapter 7 "Using Channels" of the MQSeries Clients reference for a detailed explanation of client channel definition tables and how they work.

Note

The same file may be used by more than one MQSeries client. You change the name and location of this file using the MQCHLLIB and MQCHLTAB MQSeries environment variables. See Chapter 6, "Using MQSeries environment variables" on for details of these and all the other MQSeries environment variables.

## Multiple Queue Managers

You may choose to define connections to more than one server machine because:

- You need a backup system.
- You want to be able to move your queue managers without changing any application code.

- You need to access multiple queue managers, and this requires the least resource.

Note

Define your client-connection and server-connection channels on one queue manager only, including those channels that connect to a second or third queue manager. Do **no**t define them on two queue managers and then try to merge the two client channel definition tables; this cannot be done. Only one client channel definition table can be accessed by the client.

Chapter

11

# Solving Problems

This chapter discusses the return codes, error logs, and error messages. It examines some common problems when running applications in the MQSeries client environment. Trace tools are also covered.

An application running in the MQSeries client environment receives MQRC_* reason codes in the same way as MQSeries server applications. However, there are additional reason codes for error conditions associated with MQSeries clients.

For example:

- Remote machine not responding
- Communications line error
- Invalid machine address

The most common time for errors to occur is when an application issues an MQCONN or MQCONNX and receives the response MQRC_Q_MQR_NOT_AVAILABLE. Look in the client error log for a message explaining the failure. There may also be errors logged at the server, depending on the nature of the failure. Also, check that the application on the MQSeries client is linked with the correct library file.

## MQSeries client fails to make a connection

When the MQSeries client issues an MQCONN or MQCONNX call to a server, socket and port information is exchanged between the MQSeries client and the server. For any exchange of information to take place, there must be a program on the server machine whose role is to 'listen' on the communications line for any activity. If there is no program doing this, or there is one but it has problems of its own, the MQCONN or MQCONNX call fails and the relevant reason code is returned to the MQSeries application.

If the connection is successful, MQSeries protocol messages are then exchanged and further checking takes place. It is not until all these checks are successful that the MQCONN or MQCONNX call will succeed.

During the MQSeries protocol checking phase, some aspects are negotiated while others cause the connection to fail.

For full details of the MQRC_* reason codes, see the MQSeries Application Programming Reference.

## Stopping MQSeries clients

Even though an MQSeries client has stopped, it is still possible for the process at the server to be holding its queues open. The queues will be closed when the communications layer detects that the partner has gone.

## 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 the /var/mqm/errors directory of the MQSeries client machine.

## Using trace on UnixWare

MQSeries Client for UnixWare uses the following commands for the MQSeries client trace facility:

- **strmqtrc -**e to start early tracing
- **endmqtrc -**e to end early tracing
- **dspmqtrc <filenam**e> to display a formatted trace file

For more information about the trace commands, see the MQSeries System Administration book for Version 5 products, or the System Management Guide for your platform for non-Version 5 products.

The trace facility uses a number of files, which are:

- One file for each entity being traced, in which trace information is recorded
- One additional file on each machine, to provide a reference for the shared memory used to start and end tracing
- One file to identify the semaphore used when updating the shared memory

Files associated with trace are created in a fixed location in the file tree, which is `/var/mqm/trace`.

All queue managers tracing, all early tracing and all @SYSTEM tracing takes place to files in this directory.

Note

You can handle large trace files by mounting a temporary file system over this directory.

## File names for trace files

Trace file names are constructed in the following way:
AMQppppp.TRC

where `ppppp` is the process ID (PID) of the process producing the trace.

Notes

1. The value of the process id can contain fewer or more digits than shown in the example.

2. There will be one trace file for each process running as part of the entity being traced.

## How to examine FFSTs

FFST information on UnixWare is recorded in a file in the `/var/mqm/errors` directory.

These are normally severe, unrecoverable errors and indicate either a configuration problem with the system or an MQSeries internal error.

The files are named `AMQnnnnn.mm.FDC`, where:
`nnnnn` is the process id reporting the error

`mm` is a sequence number, normally 0

When a process creates an FFST it also sends a record to syslog. The record contains the name of the FFST file to assist in automatic problem tracking.

The syslog entry is made at the "user.error" level.

The MQSeries trace utility is explained in detail in the MQSeries System Administration (MQSeries Version 5 products) and the relevant System Management Guide for other platforms.