BEA MessageQ

Client for Windows
User's Guide

BEA MessageQ for Windows Version 5.0
Document Edition 4.0
October 1998

Copyright

Restricted Rights Legend

Trademarks or Service Marks

MessageQ Client for Windows User's Guide

Document Edition	Date	Software Version
4.0	October 1998	MessageQ Client for Windows, Version 5.0

Table of Contents

3. Configuring the MessageQ Client

4. Using the MessageQ Client

5. Troubleshooting

Index

Preface

Purpose of This Document

This document provides instructions on installing, configuring, and using the MessageQ Client to run MessageQ applications. The MessageQ Client provides applications with the full support of MessageQ features, while using significantly fewer resources than a full message server.

Who Should Read This Document

This document is intended for the following audiences:

- system installers who will install BEA MessageQ on supported platforms
- system administrators who will configure, manage, and troubleshoot BEA MessageQ on supported platforms
- applications designers and developers who will design, develop, build, and run BEA MessageQ applications

How This Document Is Organized

BEA MessageQ Client for Windows User's Guide is organized as follows:

- Chapter 1, "Introduction," describes the MessageQ Client for Windows, including benefits and an architectural overview.

- Chapter 2, “Installing the MessageQ Client,” describes how to install the MessageQ Client for Windows software, including requirements, installation procedure, and postinstallation tasks.
- Chapter 3, “Configuring the MessageQ Client,” describes how to configure the MessageQ Client for Windows, including information on configuring servers, automatic failover, logging, message recovery services, and tracing.
- Chapter 4, “Using the MessageQ Client,” describes how to develop, run, and manage MessageQ Client applications.
- Chapter 5, “Troubleshooting,” describes how to identify and correct problems while running your MessageQ client applications, including information on identifying errors, logging, tracing, and recovery.

How to Use This Document

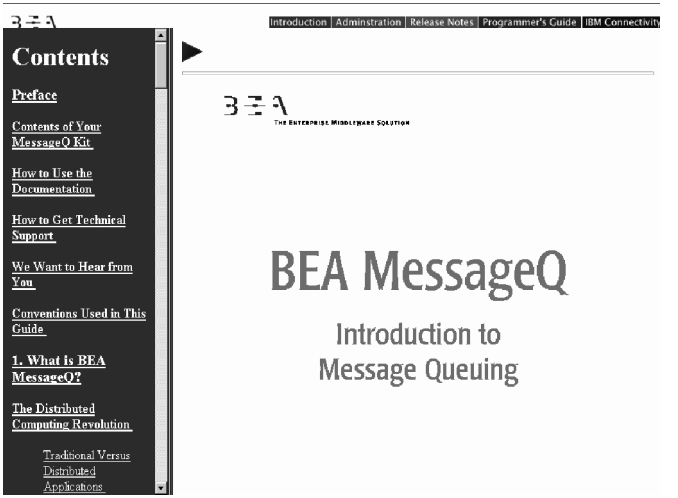
This document is designed primarily as an online, hypertext document. If you are reading this as a paper publication, note that to get full use from this document you should access it as an online document via the BEA MessageQ Online Documentation CD. The following sections explain how to view this document online, and how to print a copy of this document.

Opening the Document in a Web Browser

To access the online version of this document, open the `index.htm` file in the top-level directory of the BEA MessageQ Online Documentation CD. On the main menu, click the Introduction to Message Queuing button. Figure 1 shows the online document with the clickable navigation bar and table of contents.

Note: The online documentation requires a Web browser that supports HTML version 3.0. Netscape Navigator version 3.0 or Microsoft Internet Explorer version 3.0 or later are recommended.

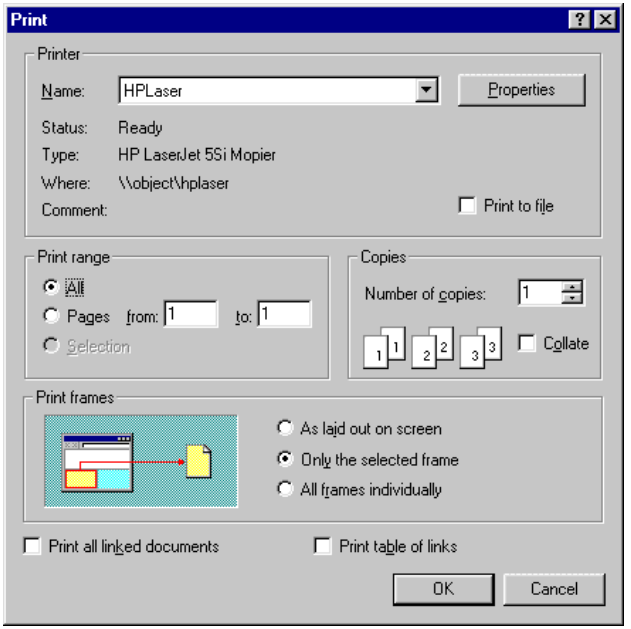
Figure 1 Online Document Displayed in a Netscape Web Browser



Printing from a Web Browser

You can print a copy of this document, one file at a time, from the Web browser. Before you print, make sure that the chapter or appendix you want is displayed and *selected* in your browser.

To select a chapter or appendix, click anywhere inside the chapter or appendix you want to print. If your browser offers a Print Preview feature, you can use the feature to verify which chapter or appendix you are about to print. If your browser offers a Print Frames feature, you can use the feature to select the frame containing the chapter or appendix you want to print. For example:



The BEA MessageQ Online Documentation CD also includes Adobe Acrobat PDF files of all of the online documents. You can use the Adobe Acrobat Reader to print all or a portion of each document. On the CD's main menu, click the Bookshelf button. On the Bookshelf, scroll to the entry for the BEA MessageQ document you want to print and click the PDF option.

Documentation Conventions

The following documentation conventions are used throughout this document.

Convention	Item
boldface text	Indicates terms defined in the glossary.
Ctrl+Tab	Indicates that you must press two or more keys simultaneously.
italics	Indicates emphasis or book titles.

Convention	Item

`monospace text`	Indicates code samples, commands and their options, data structures and their members, data types, directories, and file names and their extensions. Monospace text also indicates text that you must enter from the keyboard.
	Examples:
	`#include <iostream.h> void main ( ) the pointer psz`
	`chmod u+w *`
	`\tux\data\ap`
	`.doc`
	`tux.doc`
	`BITMAP`
	`float`

`monospace boldface text`	Identifies significant words in code. *Example*: `void` **`commit`** `( )`
`monospace italic text`	Identifies variables in code. *Example*: `String` *`expr`*
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. *Example*s: LPT1 SIGNON OR
`{ }`	Indicates a set of choices in a syntax line. The braces themselves should never be typed.

`[ ]`	Indicates optional items in a syntax line. The brackets themselves should never be typed.
	Example:
	`buildobjclient [-v] [-o name ] [-f` *`file-list`*`]...`
	`[-l` *`file-list`*`]...`

`|`	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.

Convention	Item

. . .	Indicates one of the following in a command line:
	♦ That an argument can be repeated several times in a command line
	♦ That the statement omits additional optional arguments
	♦ That you can enter additional parameters, values, or other information
	The ellipsis itself should never be typed.
	Example:
	`buildobjclient [-v] [-o name ] [-f file-list]...`
	`[-l file-list]...`

. . .	Indicates the omission of items from a code example or from a syntax line. The vertical ellipsis itself should never be typed.

Related Documentation

The following sections list the documentation provided with the MessageQ software, related BEA publications, and other publications related to the technology.

MessageQ Documentation

The MessageQ information set consists of the following documents:

BEA MessageQ Introduction to Message Queuing

BEA MessageQ Installation and Configuration Guide for UNIX

BEA MessageQ Installation and Configuration Guide for Windows NT

BEA MessageQ Programmer's Guide

BEA MessageQ System Messages

BEA MessageQ FML Programmer's Guide

BEA MessageQ FML Reference Pages

BEA MessageQ Client for UNIX User's Guide

Note: The BEA MessageQ Online Documentation CD also includes Adobe Acrobat PDF files of all of the online documents. You can use the Adobe Acrobat Reader to print all or a portion of each document.

Contact Information

The following sections provide information about how to obtain support for the documentation and software.

Documentation Support

If you have questions or comments on the documentation, you can contact the BEA Information Engineering Group by e-mail at **docsupport@beasys.com**. (For information about how to contact Customer Support, refer to the following section.)

Customer Support

If you have any questions about this version of ProductName, or if you have problems installing and running ProductName, contact BEA Customer Support through BEA WebSupport at `www.beasys.com`. You can also contact Customer Support by using the contact information provided on the Customer Support Card, which is included in the product package.

When contacting Customer Support, be prepared to provide the following information:

- Your name, e-mail address, phone number, and fax number
- Your company name and company address
- Your machine type and authorization codes

- The name and version of the product you are using
- A description of the problem and the content of pertinent error messages

CHAPTER

1 Introduction

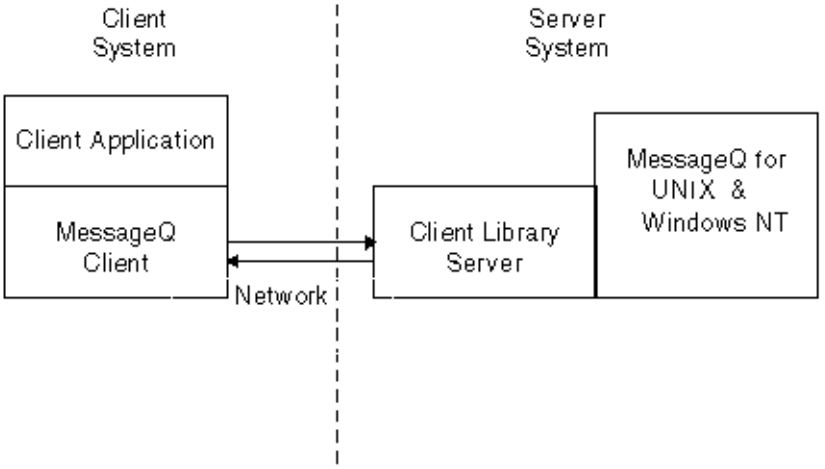
What is the MessageQ Client?

The MessageQ Client, Version 5.0 is a client implementation of the MessageQ Application Programming Interface (API). It provides message queuing support for distributed network applications using a MessageQ Server to provide reliable message queuing for distributed multi-platform network applications. The MessageQ Client is referred to as a "light-weight" implementation of MessageQ because it requires fewer system resources (disk space and memory) and less configuration and management than a MessageQ Server.

The MessageQ Client is connected to the message queuing bus through a network connection with a Client Library Server (CLS) on a remote MessageQ Server. The CLS acts as a remote agent to perform message queuing operations on behalf of the MessageQ Client. The CLS runs as a background server to handle multiple MessageQ Client connections. The MessageQ Client establishes a network connection to the CLS when an application attaches to the message queuing bus. The CLS performs all communication with the client application until the application detaches from the message queuing bus. The network connection to the CLS is closed when the application detaches from the message queuing bus.

MessageQ Clients are available for Windows 95, Windows NT, most popular UNIX systems, OpenVMS (MessageQ Version 4.0A), and IBM MVS (MessageQ Version 3.2B) systems. See Figure 1-1 for a diagram of the MessageQ Client and Server components.

Figure 1-1 MessageQ Client and Server Components



The MessageQ Client allows multiple applications to connect to separate queues on the message queuing bus. A separate network connection to the CLS is maintained for each MessageQ Client application. The message queuing operations and network activities of each client are isolated from other clients. The total number of applications that can connect to the message queuing bus is limited by the number of TCP/IP sessions. To provide robust network connections, a backup CLS can be configured for automatic failover if the primary CLS becomes unavailable. The MessageQ Client can also automatically reconnect to the CLS following a network failure.

When the connection to the CLS is unavailable, the MessageQ Client provides recoverable messaging using a local store-and-forward (SAF) journal to store recoverable messages. When the connection to the CLS is reestablished, all messages in the SAF journal are sent before new messages are processed.

The MessageQ Client supports a variety of popular application development environments and languages including C/C++, Visual Basic, PowerBuilder and others. In addition, it includes several utility programs to monitor and test applications. Example programs in C and Visual Basic demonstrate the use of various features of the MessageQ API. On Windows systems, the MessageQ Client Custom Controls offer additional support for Visual Basic developers by providing a simple yet powerful means for integrating MessageQ software into Windows applications.

Benefits of Using the MessageQ Client

The MessageQ Client provides the following benefits:

- Reduces system resource load
- Reduces system management overhead
- Provides network protocol independence

The MessageQ Client provides message queuing capabilities for MessageQ applications using fewer system resources (shared memory and semaphores) and running fewer processes than a MessageQ Server. Therefore, the MessageQ Client enables distributed MessageQ applications to run on smaller, less powerful systems than the systems required to run a MessageQ Server.

Run-time configuration of the MessageQ Client is extremely simple. A minimal configuration requires only the name of the server system, the network endpoint to be used by the CLS, and the desired network transport. Running the MessageQ Client makes it unnecessary to install and configure a MessageQ Server on each system in the network. Instead, a distributed MessageQ environment can consist of a single system running a MessageQ UNIX or Windows NT Server and one or more systems running MessageQ Clients.

For example, suppose a small business has 10 networked workstations that need to run a MessageQ application. Prior to MessageQ Version 4.0, it would be necessary to install, configure, and manage a message queuing group on each workstation. Now, however, by using the MessageQ Client, a MessageQ Server need be installed and configured only on a single workstation. Installing the MessageQ Client on the remaining nine workstations provides message queuing support for all other MessageQ applications in the distributed network.

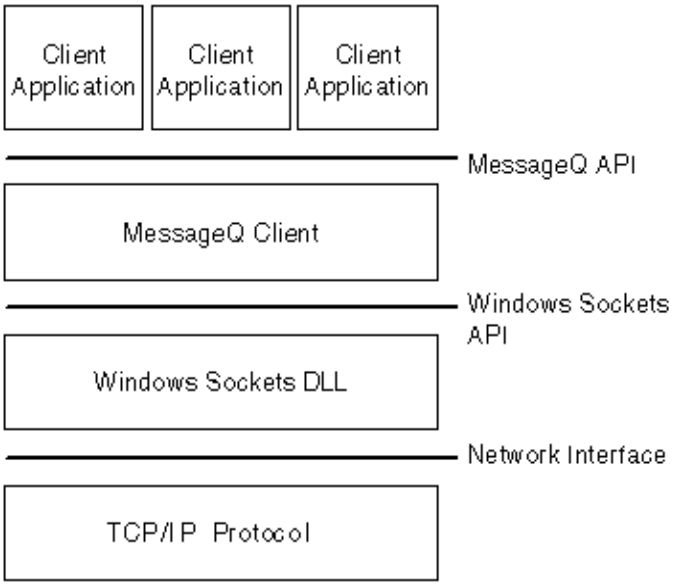
In this example, only one workstation needs to be sized and configured to optimize performance, reducing the burden of system management to a single machine. System management and configuration for the remaining systems is drastically simplified because managing the MessageQ Client consists mainly of identifying the MessageQ Server that provides full message queuing support. The MessageQ Client can be reconfigured quickly and easily and multiple clients can share the same configuration settings to further reduce system management overhead.

The MessageQ Client performs all network operations for client applications, making it unnecessary for a client program to be concerned about the underlying network protocol. The MessageQ Client enhances the portability of applications, enabling them to be ported to a different operating system and network environment supported by MessageQ with no change to the application code.

Architectural Overview

The MessageQ Client for Windows provides a reentrant 32-bit Dynamic Link Library (DLL) supporting multiple MessageQ-enabled Windows applications. Both Intel (Windows NT and Windows 95) and Alpha (Windows NT) DLLs are provided. Figure 1-2 shows the MessageQ Client for Windows architecture.

Figure 1-2 MessageQ Client for Windows Architecture



The MessageQ Client allows multiple applications to connect to separate queues on the message queuing bus. A separate network connection to the CLS is created for each client application. The total number of applications that can connect to the message queuing bus is limited by the number of TCP/IP sessions. On Windows systems, the Client DLL uses the Windows Sockets API for network services.

If the network connection to the CLS is lost or unavailable, the MessageQ Client optionally stores messages in a local journal file for later retransmission.

The Client Library Server

The Client Library Server (CLS) is a MessageQ application that runs as a background server. The CLS performs all communication with the MessageQ Client for each client application until the application detaches from the message queuing bus. The message queuing operations and network activity of each client are isolated from other clients.

The CLS supports multiple client connections using the following techniques:

- On Windows NT Server systems, the CLS is multi-threaded.
- On UNIX Server systems, the CLS uses a separate process to handle each MessageQ Client connection.
- On OpenVMS Server systems (MessageQ Version 4.0A), the CLS can operate in two modes:

Single-client mode	A separate CLS process is created to support each remote client.
Multi-client mode	A single CLS process supports multiple clients using asynchronous message queuing operations provided by MessageQ for OpenVMS systems.

When the CLS starts, it initializes a listener process (or thread) that establishes a network endpoint and waits for connections from a client application. The endpoint on which the CLS listens is determined by the command-line arguments used to start the CLS.

MessageQ Client applications attempt to connect to the CLS when they initiate an attach queue operation. The MessageQ Client uses configuration information in the `dmq.ini` configuration file or the Windows Registry to identify the location of the CLS. If a `dmq.ini` configuration file exists in your search path, then that configuration file is used. If no `dmq.ini` configuration file is found, then the configuration file from the Windows Registry is used. The CLS creates a server subprocess (or server thread) for each new client connection. The server subprocess terminates when the client detaches from the bus, or the network connection is closed.

The CLS can use a security file (located on the server system) to control client access to the message bus. Client access can be restricted to specific queues or CLS endpoints.

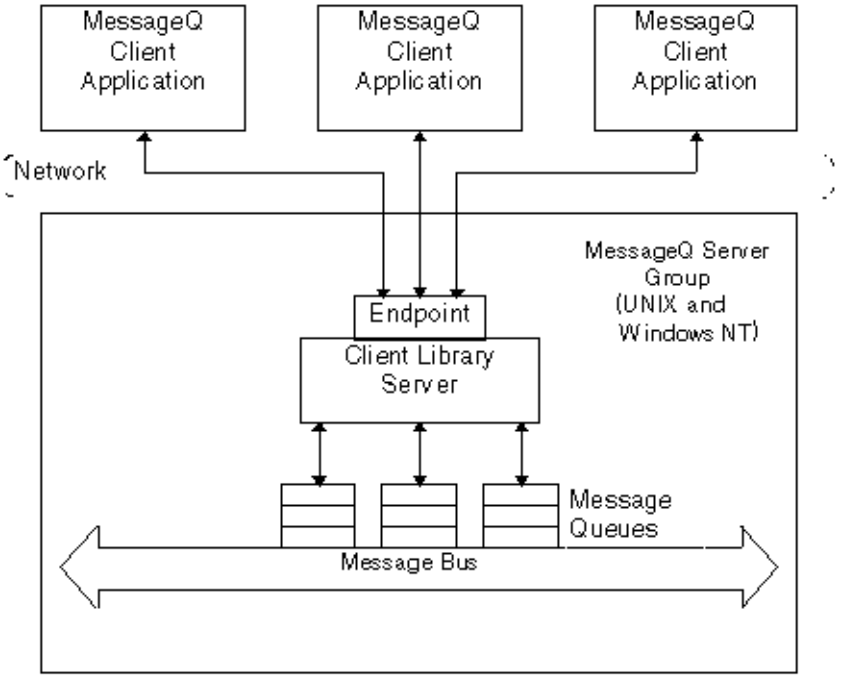
How the MessageQ Client and CLS Work Together

The MessageQ Client uses a request/response protocol to communicate with a Client Library Server (CLS) running on a MessageQ server system. The MessageQ Client is called a light-weight client connection to the MessageQ message queuing bus because it relies on a MessageQ Server for the following:

- Message queues for all MessageQ Client applications are implemented on a remote system running a MessageQ Server group.
- Message delivery to target queues is provided by the Queuing Engine, a server process that runs on a MessageQ Server.
- Message routing and cross-group transport among multiple MessageQ Server systems and other MessageQ Client applications are provided by the MessageQ Server group.
- Guaranteed message delivery is provided by the MRS capability of the MessageQ Server group. The MessageQ Client provides a local store-and-forward (SAF) journal for temporarily storing recoverable messages when the connection to the CLS is not available.

Figure 1-3 shows the relationship of the MessageQ Client and CLS to the MessageQ message queuing bus.

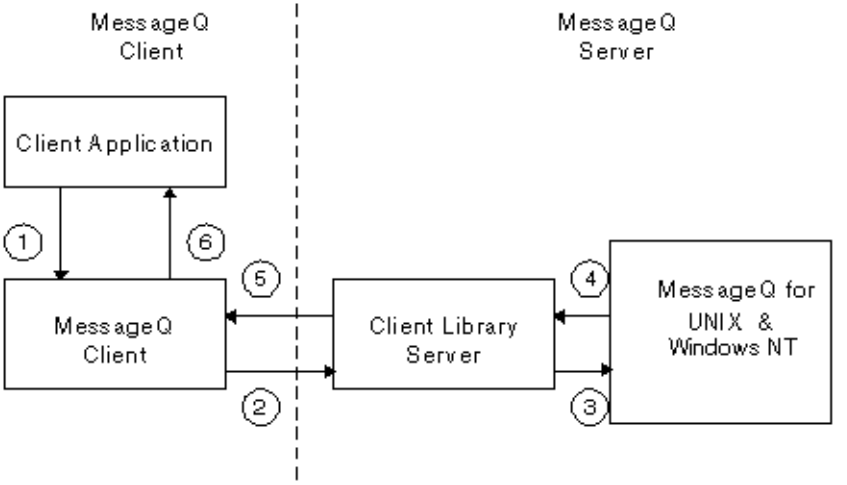
Figure 1-3 MessageQ Client and CLS Architecture



All MessageQ Client API functions supported by the CLS are processed using the following sequence of events as shown in Figure 1-4.

1. The client application makes a MessageQ function call to the MessageQ Client.
2. The MessageQ Client verifies the function call arguments and sends them in a request to the CLS, which is waiting to receive client requests.
3. When a request arrives, the CLS makes the corresponding MessageQ function call in the MessageQ Server group.
4. The MessageQ function completes, and returns the results to the CLS.
5. The CLS sends the return parameters and function status in a response back to the MessageQ Client that initiated the request.
6. The MessageQ Client function call returns to the application with the return arguments and function status.

Figure 1-4 How Client Application Requests are Processed



2 Installing the MessageQ Client

This chapter describes how to install the MessageQ Client for Windows software. It includes the following topics:

- Installation Prerequisites
- Installation Procedures
- Uninstalling MessageQ Client for Windows

Installation Prerequisites

To successfully install MessageQ Client software on your client machine, you must ensure that your environment meets the following installation requirements:

- Hardware
- Software
- Disk space
- System disk backup

Hardware Requirements

To perform the installation, you need the following hardware:

- Processor: Intel 486 or higher, or Digital Alpha
- CD-ROM drive to read software distribution media
- Approximately 3MB of free disk space

Software Requirements

Your system must meet the following software requirements before you can run the MessageQ Client software.

- The MessageQ for UNIX or Windows NT product must be installed on a server machine and a message queuing group must be configured to support the requirements of your messaging application environment. The MessageQ Client applications use messaging resources, including message queues, message buffers, and system resources on the server system. See the *Installation and Configuration Guide* for your MessageQ server system and review the system resource requirements for using MessageQ in your environment.
- When using TCP/IP, the host names for the client and server systems must be properly identified in the `hosts` files on both the client system and the MessageQ server system. Refer to the TCP/IP documentation provided by your vendor for the `hosts` file location.
- A Windows Sockets, Version 1.1-compliant TCP/IP protocol implementation must be available to communicate with the remote MessageQ server system. On Windows NT and Windows 95 this is provided by `wsock32.dll`.
- Before attempting to use the MessageQ Client, verify the TCP/IP connection between the client and server system using the `ping` utility. For more information about TCP/IP networking, see the documentation for your client and server systems.
- A programming development environment, such as Visual Basic, Visual C++, or Borland C++ that will allow you to use the MessageQ Client API or Custom Controls.

For a complete description of MessageQ server systems and TCP/IP transports supported by the MessageQ Client, see the *MessageQ Client for Windows Release Notes*.

Platforms and Transports Supported

This section describes the hardware, operating systems, and network transports supported by MessageQ Client for Windows, Version 5.0.

Client Platforms Supported

The MessageQ Client supports operating system and network environments as listed in Table 2-1.

Table 2-1 Supported Client Platforms

Operating System	Transports
Windows NT 4.0 (Intel or Alpha)	Windows NT TCP/IP
Windows 95/98	Windows 95 TCP/IP

MessageQ Server Compatibility

The MessageQ Client runs with all MessageQ, Version 2.1 or higher message server implementations with the following exceptions:

- The CLS Security Utility requires MessageQ for UNIX or MessageQ for Windows NT, Version 3.1 or later or MessageQ for OpenVMS, Version 4.0 or later.
- Use of features such as self-describing messages, handle-based messages, global naming, and selective broadcast services require MessageQ for OpenVMS, MessageQ for UNIX, or MessageQ for Windows NT systems, Version 4.0 or Version 4.0A.
- Use of features such as FML messaging, double pointers, correlation identifier, and transparent messaging between BEA MessageQ and BEA TUXEDO require Message Q for UNIX or Windows NT Version 5.0.

Disk Space Requirements

MessageQ Client systems require approximately 3 megabytes of free disk space to hold the MessageQ installation files.

Backing Up Your System Disk

We recommend that you back up your system disk before installing any software. For details on performing a system disk backup, see your Microsoft Windows system documentation.

Installing on Systems Already Running the MessageQ Client

If you have a previous version of the MessageQ Client already installed on your system, we recommend that you install Version 5.0 software in the default installation directory, `c:\...\BEA Systems\MessageQ`.

Regardless of the directory in which the product is installed, the icons for the Program Manager are overwritten to point to the new installation. If a previous version is installed, you will not be able to access it from the Program Manager using the icons.

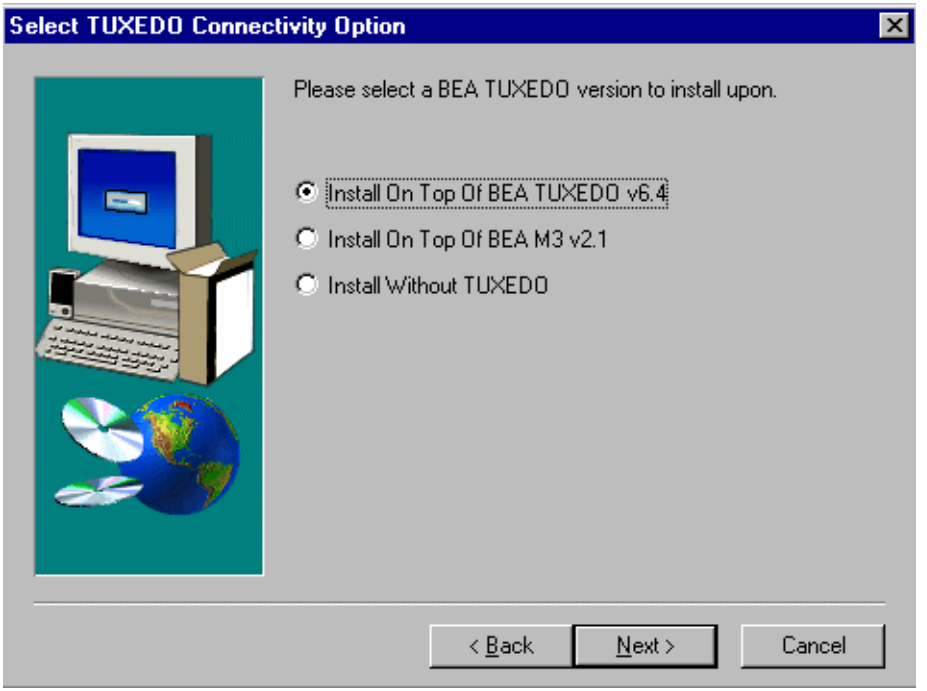
Installing the MessageQ Client

The following procedure describes how to install MessageQ Client for Windows software.

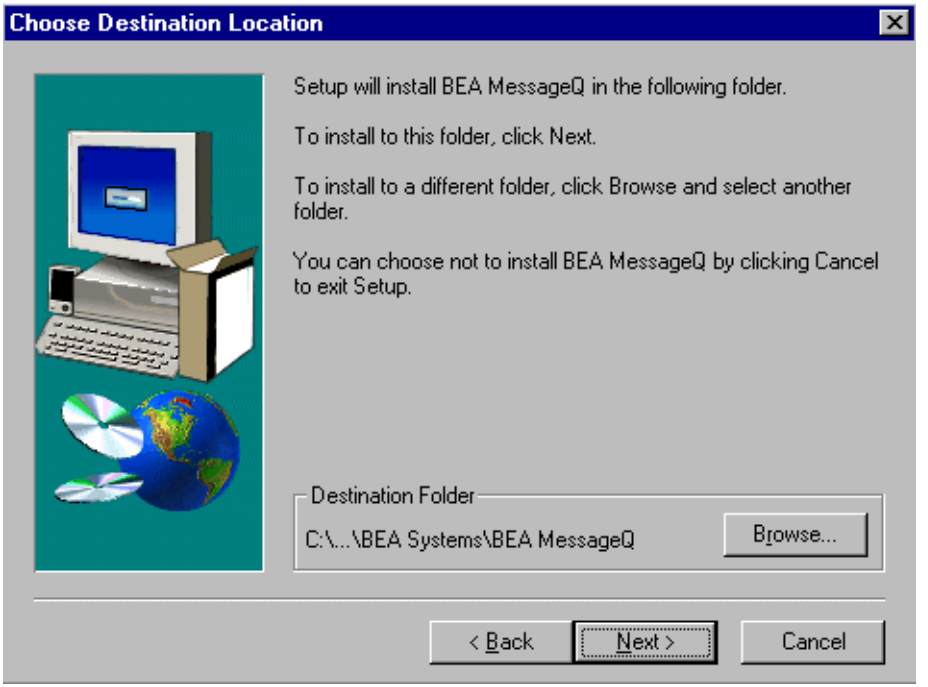
Note: You can stop the installation procedure at any time by clicking Exit in the Installation Options dialog. You can also return to the previous dialog by clicking Back.

1. Log into the Administrator account.

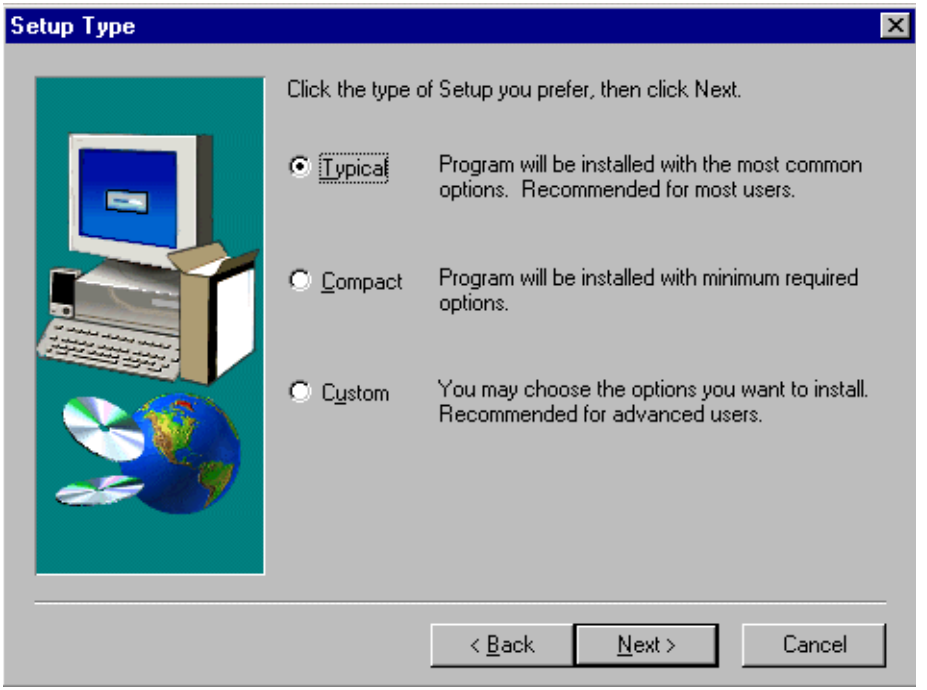
2. Load the MessageQ distribution CD-ROM into the CD reader.
3. Open Windows Explorer.
4. Double-click the applicable folder based on your system processor: WinNT for the Intel version and AlphaNT for the Alpha version.
5. Double-click `setup.exe` to start the installation.
6. Click **Next** after the SETUP utility displays the dialog box that welcomes you to the MessageQ software installation.
7. If you have BEA TUXEDO installed on your system, select a BEA TUXEDO version to install upon and click **Next**. You can also install MessageQ as a standalone product.



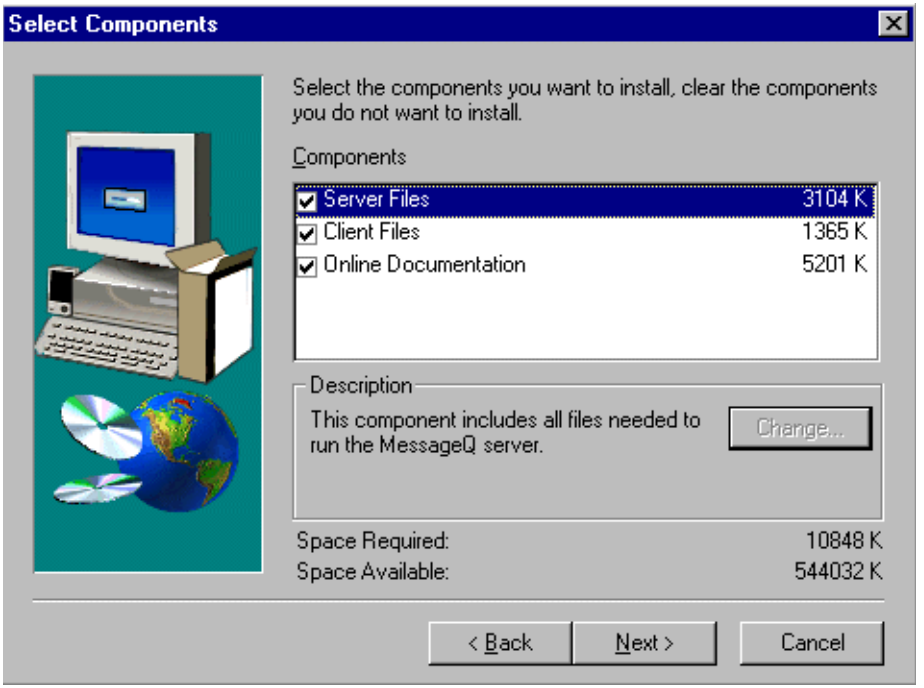
8. Choose the destination location and click **Next**. The default installation for MessageQ is `C:\Program Files\BEA Systems\BEA MessageQ`.



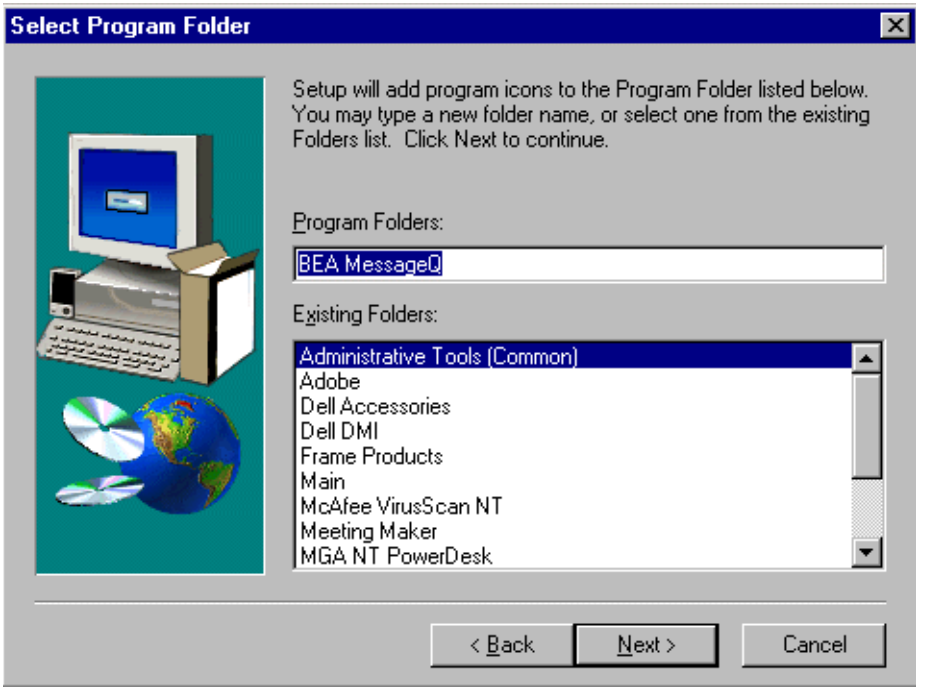
9. Choose between a Compact or Custom installation and click **Next**. (The Compact installation installs the MessageQ Client for Windows files and the Release Notes. The Custom installation allows you to choose specific components to install.)



10. If you select a Custom installation, you are prompted to select the components you want to install and click **Next**. The Select Components box displays a description of the selected component and shows you information on space required and space available.



11. Choose the Program Folder name and click **Next**.



12. After BEA MessageQ has been successfully installed and all files are transferred to your system, click **Finish** to complete the installation.

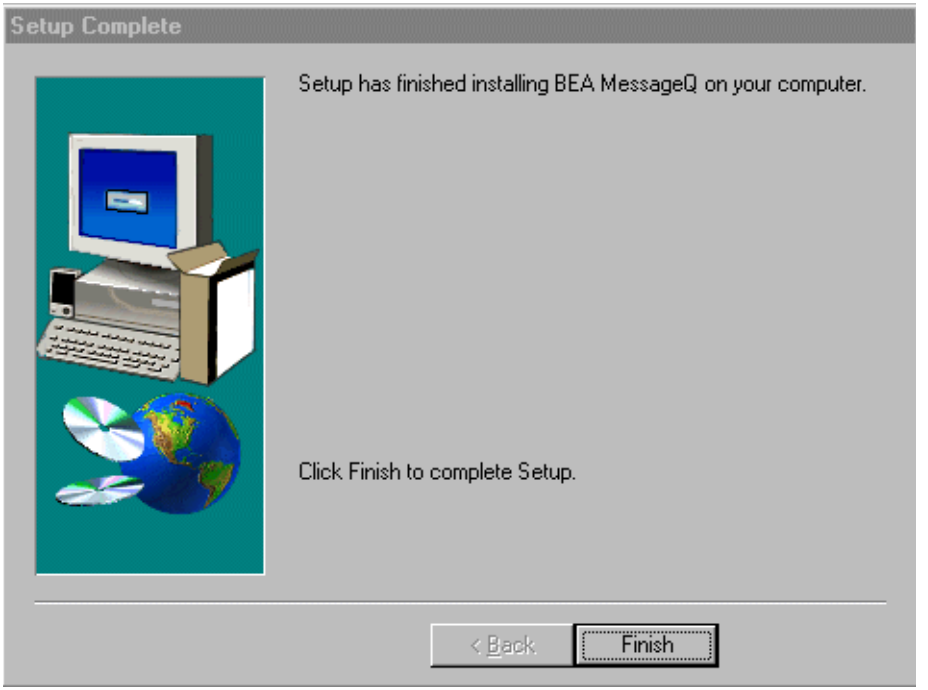


Figure 2-1 shows the MessageQ program group created after a Typical installation is complete.

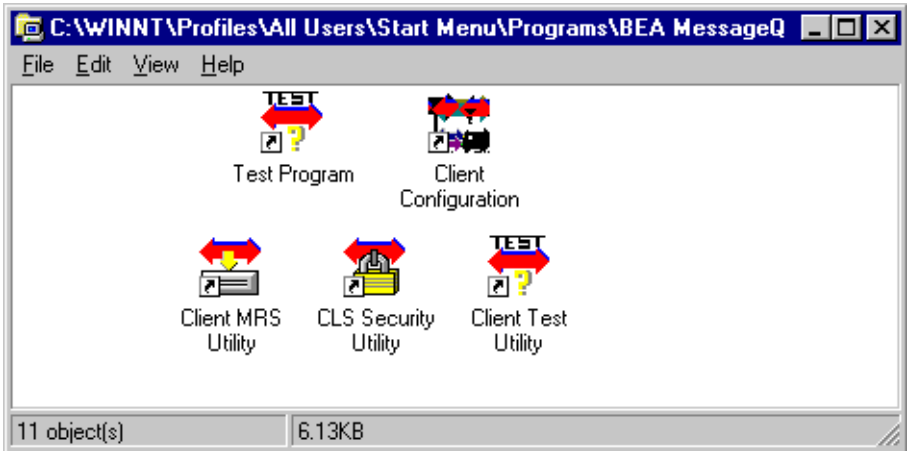


Figure 2-1 MessageQ Program Group Server Icons

Table 2-2 describes the program group utility icons.

Table 2-2 Utility Group Programs

MessageQ Icon	Description
MessageQ Client Configuration Utility	Windows-based program that lets you configure automatic failover, message recovery services, minimize network packets (MPP) protocol, logging, and tracing.
MessageQClient MRS Utility	Windows-based program that lets you view the contents of local store-and-forward (SAF) journals.
MessageQ CLS Security Utility	Windows-based program that allows a MessageQ administrator to define a list of MessageQ client systems that can attach to CLS.
MessageQ Client Test Utility	Windows-based program that enables you to test message exchanges between the queues you configured.

If you encounter a problem while using MessageQ and you believe the error is caused by a problem with MessageQ, contact BEA Customer Support for assistance.

Adding the MessageQ Client File Drive and Directory to Your Path

After completing the installation procedure, you can verify that the MessageQ Client software is successfully installed on your system.

Ensure that the files listed in Table 2-3 are located in directories identified by the PATH environment variable:

Table 2-3 PATH Environment Variable

Windows NT or Windows 95
`dmqcl32.dll`
`dmq.ini` (Optional: Configuration can be stored in the Registry)
TCP/IP DLL

- Add the MessageQ `\bin` directory to your PATH environment variable. The default location is `c:\...\BEA Systems\MessageQ\bin`.
- To compile and link MessageQ Client applications, add the following directories to the INCLUDE and LIB environment variables as shown in Table 2-4.

Table 2-4 INCLUDE and LIB Environment Variables

Environment Variable	Default Directory
INCLUDE	`c:\...\BEA Systems\MessageQ\include`
LIB	`c:\...\BEA Systems\MessageQ\lib`

Uninstalling the MessageQ Client

To remove the MessageQ Client from your system, use the Add/Remove Programs utility on your system's Control Panel.

3 Configuring the MessageQ Client

This chapter describes how to configure the MessageQ Client. Table 3-1 lists and describes the configuration options for the MessageQ Client.

Table 3-1 MessageQ Client Configuration Options

Option	Description	Required?
Default Server	Server host name, endpoint definition, and automatic reconnection options	Yes
Failover	Server host name and endpoint definition for the failover server	No
MRS	Settings for enabling the local store-and-forward (SAF) message journal and configuring the local journal files	No
Logging	Settings for logging error events, network statistics, and tracing messages to a log file	No
Tracing	Settings to enable runtime trace information about the API calls, Client DLL activity, or Custom Control trace flags	No

Run the Client Configuration program to specify the MessageQ Client configuration settings. Configuration settings are stored in the `dmq.ini` file if it is found in the PATH, and in the Registry. The following methods allow users to determine whether they use `dmq.ini` or the Registry for their configuration settings:

- If the environment variable DMQCL_USE_REGISTRY is defined, the Registry will always be used.

- If DMQCL_USE_REGISTRY is not defined, and `dmq.ini` is found in the PATH, then `dmq.ini` will be used for configuration settings.
- If DMQCL_USE_REGISTRY is not defined, and `dmq.ini` is not found in the PATH, then the Registry is used for configuration settings.

Applications may dynamically configure `dmq.ini` at runtime using the Windows `WritePrivateProfileString` function.

Configuring the Server Connection

Configuring the connection to the MessageQ Client Library Server (CLS) consists of the following two items:

- Default server (required)

  The default CLS is used for all connections to the message queuing bus. Applications that are attempting to connect to a server (or lose a connection to the CLS) attempt to reconnect when the network connection to the server is available. If you do not enable automatic reconnect for the default server, you may want to consider configuring the automatic failover server.

- Automatic failover server

  If the primary default server is not available, the MessageQ Client provides the option of connecting to a failover server to ensure robust client connections. However, if automatic reconnect to the default server is enabled, the automatic failover server cannot be used.

Default Server

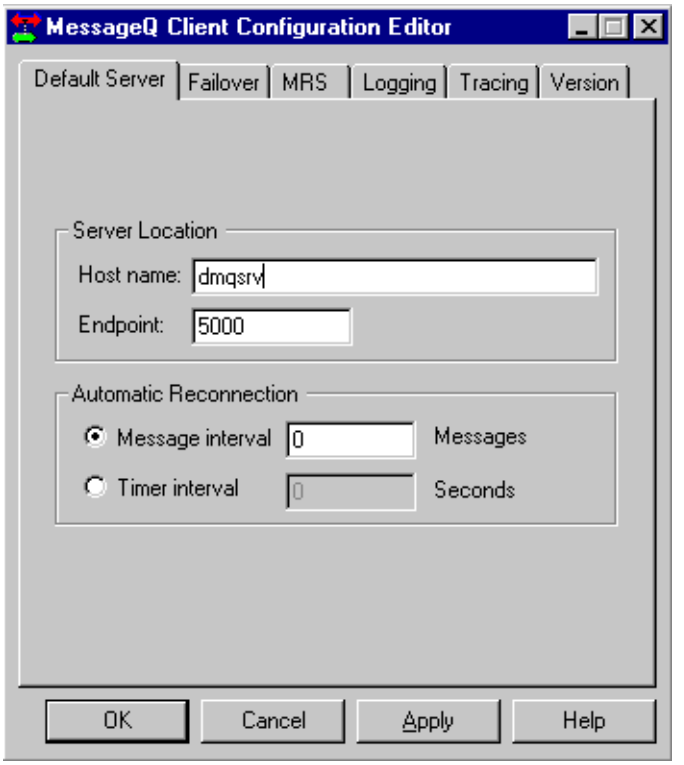
The default server identifies the MessageQ server system for all connections to the message queuing bus. If automatic reconnection is enabled, applications that lose a connection to a server (or lose a connection to the CLS) try to reconnect when the network connection to the server is available. Client applications also reconnect in the

event that the CLS or host server system is stopped and restarted. During an automatic reconnect event, the MessageQ Client attempts to connect only to the default server. Automatic reconnect does not attempt to use the failover server.

After a successful reconnect, the application is automatically attached to the message queuing bus and messaging operations can continue without interruption. All pending messages in the store-and-forward (SAF) journal are sent to the CLS before new operations can be performed. For example, when a `pams_get_msg` function call triggers the reconnect threshold and a successful automatic reconnect and attach operation completes, the SAF file is completely drained before the `pams_get_msg` function call returns.

The server configuration options shown in Figure 3-1 are described in Table 3-1.

Figure 3-1 Default Server Configuration Options



Automatic Failover Server

If the primary (default) server is not available and automatic failover is enabled, the MessageQ Client provides the option of connecting to a failover CLS. By enabling automatic failover, a MessageQ Client will transparently try to attach to the failover server when the CLS on the primary server group is not available. Attempts to connect to the failover server are only made during a call to `pams_attach_q`. Failover is not used if automatic reconnect to the default server is enabled.

Using the failover capability requires additional planning and work in order for messages to be sent and received correctly. The message queues used by MessageQ Client applications are implemented by the MessageQ server group. The message queues, and any recoverable message journals, are located on the server system.

When connecting to the failover group, the queue address used by the MessageQ Client is likely to change (unless the MessageQ group started on the failover system has the same group ID as the primary server group). Recoverable messages sent to the client using the queue address of the primary server group are not delivered to the client when it reattaches to the failover server in a different MessageQ server group.

The simplest use of automatic failover is when the MessageQ Client attaches to a temporary queue and uses a request/response style of messaging. The client sends requests to one or more servers that send responses back to the queue address that sent the request. If failover occurs, the MessageQ Client is automatically reattached to a new temporary queue and request messages are sent and responses delivered to the new queue address. The application is unaware that a failover event occurred, except that any pending response is not received.

When clients attach to a specific permanent queue and receive recoverable messages sent to that queue address, they depend on the message queuing resources of that MessageQ group. Recoverable messages sent to the queue address while the client is not attached are saved on that system. If the client reconnects to the same queue name or number, but on a different (failover) MessageQ group, the recoverable messages on the MessageQ group where the default CLS is located are not delivered to the new queue address used by the MessageQ Client.

Automatic failover is not appropriate for all applications. The MessageQ server group and all disk-based queuing resources must also fail over to another system so that messages sent to the MessageQ Client are received after a failover transition. For example, the MessageQ server group can recover by restarting the MessageQ group on another node in an OpenVMS cluster.

Figure 3-2 shows the automatic failover server configuration options.

Figure 3-2 Automatic Failover Server Configuration Options

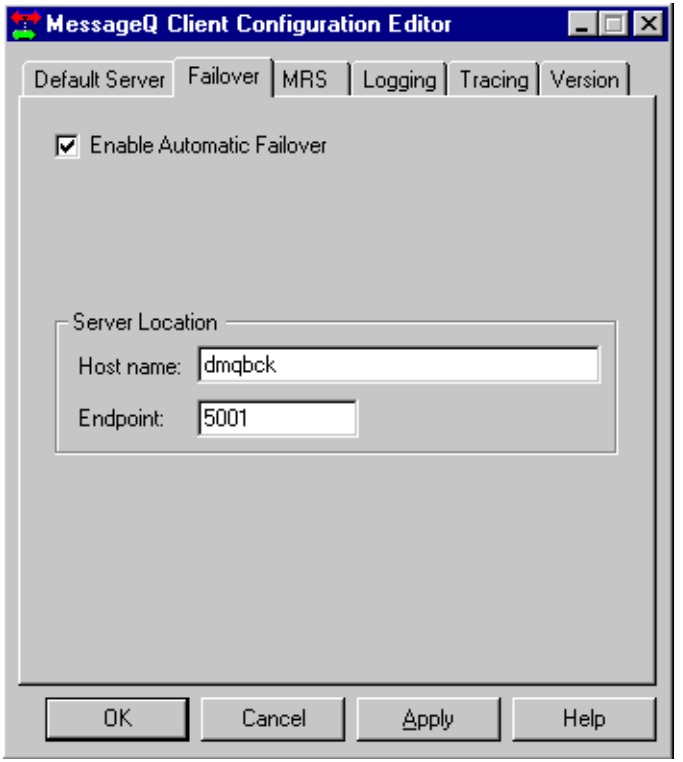


Table 3-2 lists and describes the automatic failover server configuration options.

Table 3-2 Options to Configure the Automatic Failover Server

Option	Description
Enable Automatic Failover	If checked, automatic failover is enabled. The failover server is used when the default server is not available and automatic failover is enabled. The use message interval option (see Table 3-2) must be greater than 0.
Host name	The name of the host running the MessageQ CLS. The host name must have a corresponding entry in the local hosts file or DECnet database. MessageQ Version 5.0 only supports TCT/IP as a network transport.
Endpoint	The endpoint used by the MessageQ Client and the MessageQ CLS. The endpoint identifies either the DECnet object name or the TCP/IP port number the MessageQ Client used to locate the CLS. The CLS uses the same endpoint to listen for client connections. MessageQ Version 5.0 only supports TCT/IP as a network transport. For DECnet transport, the endpoint can range from 1 to 65535 inclusive. The endpoint value is concatenated with the prefix DMQCLS_ to create a unique DECnet object name (for example, DMQCLS_05000). For TCP/IP transport, the endpoint ranges from 1024 to 65535 inclusive. Port numbers less than 1024 are reserved.

Configuring Message Recovery Services

Message Recovery Services (MRS) are the MessageQ services that manage the automatic redelivery of critical messages. Messages that are sent using a recoverable delivery mode are written to the local store-and-forward (SAF) journal when the connection to the server system is not available.

The MessageQ Client ensures delivery of recoverable messages to the CLS on the MessageQ Server by providing a store-and-forward (SAF) journal (`dmqsaf.jrn`) to store recoverable messages when the connection to a CLS is not available. Local SAF

journal processing is available when Message Recovery Services (MRS) are enabled in the MessageQ Client configuration. The location of the journal file can be set when configuring MRS.

If MRS is enabled, the message recovery journal is turned on when the client application first initiates an attach operation. If the CLS is not available at the time of an attach, the journal file is opened and the attach operation completes with return a status of PAMS__JOURNAL_ON.

When the journal is on, messages sent using the following reliable delivery modes are saved to the journal:

- PDEL_MODE_WF_MEM (using PDEL_UMA_SAF)
- PDEL_MODE_WF_DQF
- PDEL_MODE_AK_DQF
- PDEL_MODE_WF_SAF
- PDEL_MODE_AK_SAF

When the connection to the CLS is reestablished, all messages in the SAF journal are sent before new messages are processed. The SAF messages are transmitted in first-in/first-out (FIFO) order. When the connection to CLS is reestablished, a return status of PAMS__LINK_UP is used to indicate that journal processing is no longer active.

Messages are sent from the SAF when one of the following events occurs:

- The connection to the CLS is established successfully and pending messages exist in the SAF.
- The connection to the CLS is lost and the application continues to send recoverable messages. Additional message operations trigger an automatic reconnect to the CLS that is successful, and messages are pending transmission in the SAF.

The MessageQ Client MRS configuration options allow the SAF journal to be configured as follows:

- A fixed-size file that does not reuse disk blocks
- A fixed-size file that reuses (cycles) disk blocks
- A dynamic file that grows indefinitely until no more disk blocks are available

These options allow you to determine how disk resources are used for message journals. Journal files that grow indefinitely periodically allocate an extent of disk blocks as needed to store messages. When all messages are sent from the SAF and the journal is empty, the disk blocks used by the journal are freed and the journal file returns to its original size.

This section is *optional* if recoverable messaging is not used.

Figure 3-3 shows the MRS configuration options.

Figure 3-3 MRS Configuration Options

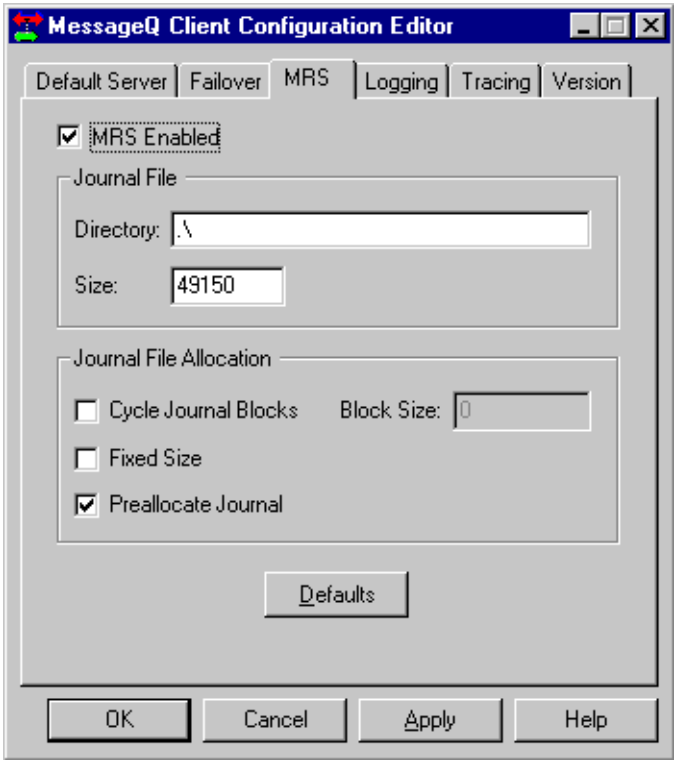


Table 3-3 describes the MRS configuration options.

Table 3-3 Options to Configure MRS

Option	Description
MRS Enabled	When checked, MRS is enabled.
Journal File Path	Specifies the path where the MessageQ journal file, `dmqsaf.jrn`, is located. The default location is the current working directory.
Journal File Size	Initial size, in bytes, of the journal file
Cycle Journal Blocks	If enabled, the journal *cycles* (reuses) disk blocks when full and overwrites previous messages. The Cycle Journal Blocks option automatically sets the Fixed Size allocation option. When Cycle Journal Blocks is enabled, all read/write operations to the journal use fixed-size journal message blocks.
Fixed Size	Determines if the journal size is fixed or allowed to grow. If Cycle Journal Blocks is enabled, Fixed Size is also enabled. **Note:** Journals that do not cycle and are not fixed can grow until the disk is full.
Preallocate Journal	If enabled, the journal file disk blocks are preallocated when the journal is initially opened.
Journal Message Block Size	Defines the file I/O block size, in bytes. Used for journal read/write operations only when Cycle Journal Blocks is enabled. Because an 80-byte MRS message header is written to the journal for each user message, you must add 80 bytes to the largest user message when calculating the Journal Message Block Size.

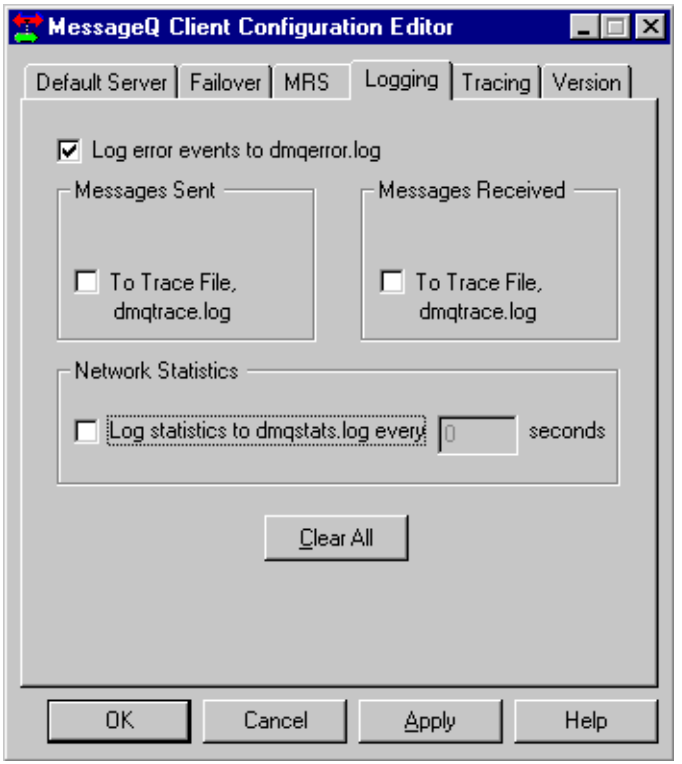
Configuring Logging

The MessageQ Client allows you to log error events and messages to a log file, as well as trace messages and write the output to a trace file. All logging information goes to `dmqerror.log` in the current working directory.

Message logging allows you to obtain a complete history of the messaging activity of your application. Tracing messages to the trace file is an effective way to monitor the run-time behavior of the application.

Figure 3-4 shows the logging options.

Figure 3-4 Logging Configuration Options

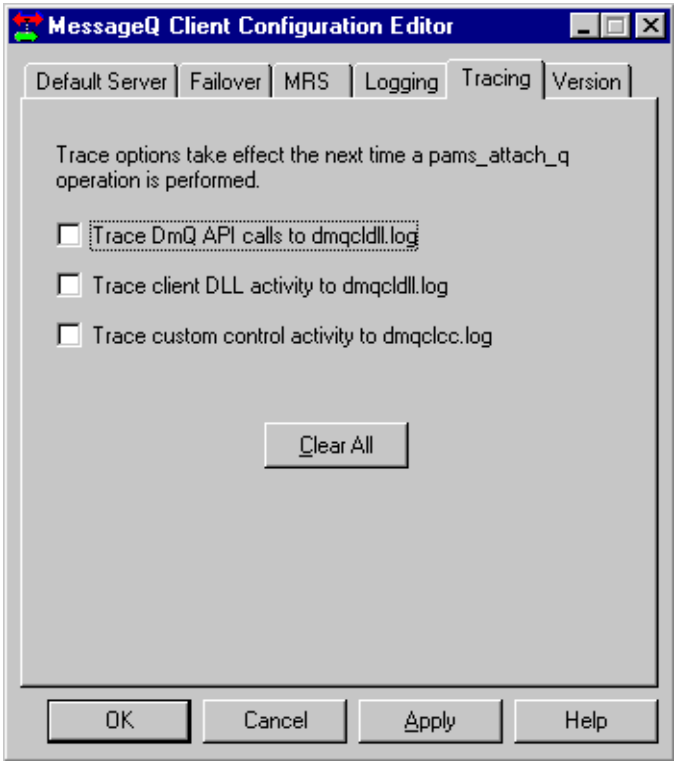


Configuring Tracing

Tracing can be a useful debugging tool, because it allows you to observe MessageQ Client processing activity. The trace output may create large output files on your system, and should only be used to monitor specific application behavior. Client message tracing (sent and received messages) configured using `dmqconf` is written to the file `dmqtrace.log` in the current working directory.

Figure 3-5 shows the tracing options.

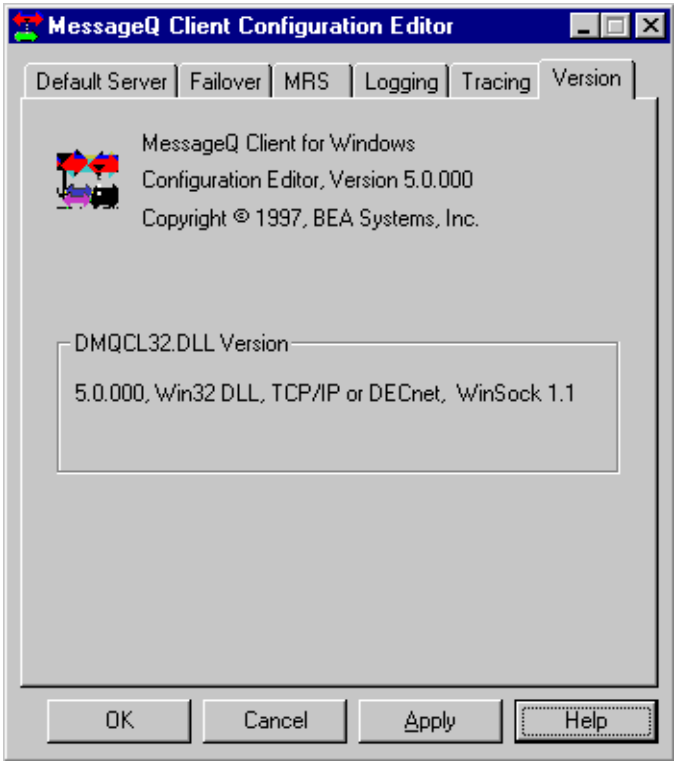
Figure 3-5 Tracing Configuration Options



Determining MessageQ Version

To determine the version of the MessageQ Client DLL, open the configuration utility. The window shown in Figure 3-6 is displayed. Click on the tab marked Version.

Figure 3-6 Client DLL Version Number



It is also possible to obtain DLL version information by displaying the file properties using Windows Explorer or File Manager.

MessageQ Utilities

In addition to the Configuration Editor, the MessageQ Client provides the utilities listed in Table 3-4 to assist in the configuration process.

Table 3-4 MessageQ Utilities

Utility	Description
Security	Allows a MessageQ administrator to define a list of MessageQ Client nodes that can attach to CLS. The utility provides a convenient user interface to review, modify, and update the security table used by CLS on a remote server system.
Test	Useful for unit-testing applications under development, the Test Utility allows you to test various MessageQ message delivery options and send messages to any process connected to the MessageQ bus.

Configuring Security Using the CLS Security Utility

The MessageQ CLS Security Utility, `dmqsecu.exe`, allows a MessageQ administrator to define a list of MessageQ client systems that can attach to CLS. The utility provides a convenient user interface to review, modify, and update the security table used by CLS on a remote server system.

A CLS security template file, `dmqclsec.txt`, is installed on the MessageQ server system. This file must be manually edited to specify the client systems that can update the security file with the CLS Security Utility. For more information about the CLS Security template file, see the Client Library Server topics in the *MessageQ Installation and Configuration Guide* for your server platform.

Run the CLS Security Utility to update or modify the CLS security file. Figure 3-7 shows the main window of the MessageQ CLS Security Utility.

The CLS Security Utility uses the MessageQ Client configuration settings to connect to a CLS. The security file is automatically retrieved when the CLS Security Utility is started. You can change the Default Server configuration to use the CLS Security Utility with multiple CLS servers in different MessageQ groups. The current CLS hostname, endpoint, and group ID are displayed in the status bar. If the CLS is not using a security file, the CLS Security Utility will display an "Error getting security file" message.

The CLS security table consists of a list of client nodes that are allowed to connect to CLS and attach to the MessageQ message queuing bus. The security table format allows a MessageQ administrator to restrict access by client nodes to specific CLS endpoints and message queues. The Client Node List is similar to the cross-group connection table in the MessageQ group configuration. Only client nodes with entries in the list are allowed to connect to CLS. All other connection requests are rejected.

The list of client nodes defined in the security table is displayed in a scrollable list on the main form. You can modify individual entries in the list by selecting (highlighting) an entry with one mouse click and selecting the **Edit...** command button, or by double-clicking on the entry. Table 3-5 shows the minimum server versions required for security file support.

Table 3-5 Minimum MessageQ Server Versions

MessageQ	Minimum Version
MessageQ for UNIX	V3.1
MessageQ for Windows NT	V3.2
MessageQ for OpenVMS	V4.0

Figure 3-7 shows a typical CLS Security Utility Main Window.

Figure 3-7 CLS Security Utility Main Window

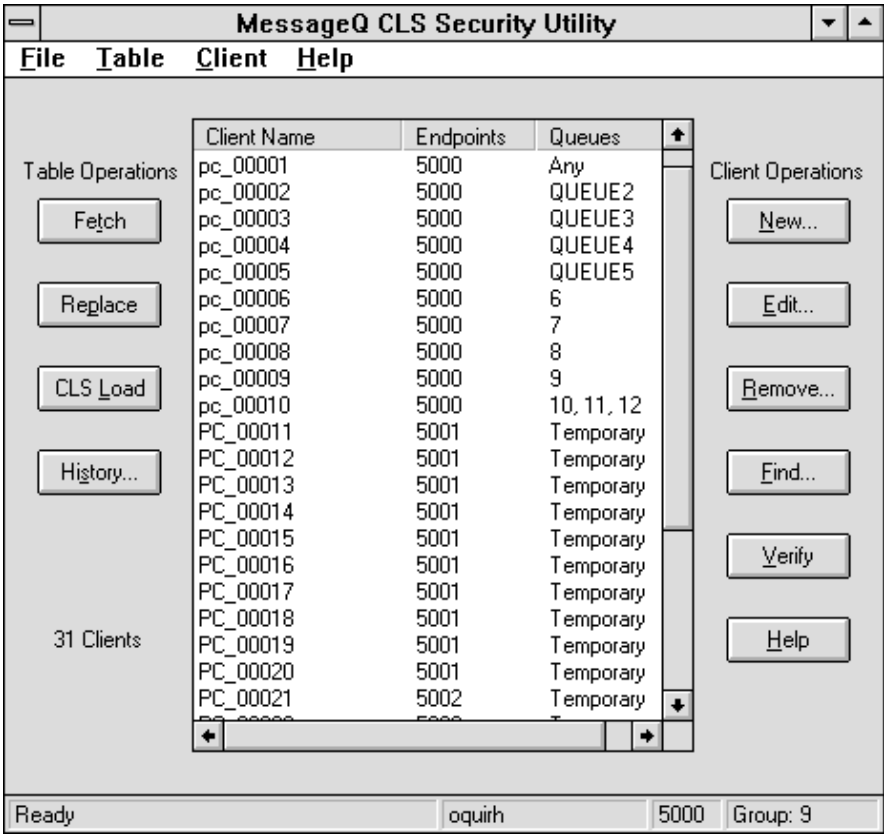


Table 3-6 lists the CLS Security Utility command buttons for the Client Operations.

Table 3-6 CLS Utility Command Buttons for Client

Button	Description
New...	Create a new client entry to add to the security table.
Edit...	Edit the currently selected entry in the Client Node List.
Remove...	Remove the currently selected entry, or multiple selection entries, in the Client Node List.
Find...	Locate an entry in the Client Node List by searching for a string.

Table 3-6 CLS Utility Command Buttons for Client

Button	Description
Verify	Perform a `pams_locate_q` operation on all queue names for all entries in the Client Node List to check that queue names are defined in the MessageQ Queue configuration.
Help	Online help

Table 3-7 lists the CLS Security Utility command buttons for the Table Operations:

Table 3-7 CLS Security Utility Command Buttons for Table Operations

Button	Description
Fetch	Request a new copy of the security table from CLS and replace any current updates to the table.
Replace	Send an updated version of the security table to CLS to save any modifications. Replace is a restricted operation. Updates to the CLS security table do not take effect until CLS is restarted, or until the user requests CLS to dynamically reload the security table.
CLS Load	Request CLS to dynamically reload the security table. CLS Load is a restricted operation. Changes to the security table do not affect clients already connected to CLS and attached to a message queue.
History...	Display the modification history for the security table and provide a form for entering new modification comments.
Help	Online help

Only client nodes that are granted CLS_SECURITY_FILE access in the CLS security table are allowed to modify the contents of the CLS security file using the CLS Security Utility. CLS security file updates are not password-protected. The CLS security table cannot be saved to a local file. However, you can print the contents of the security table by using the File menu Print option. Changes to the security table that have not been saved with the Replace command are not printed.

For more information about the CLS Security File, see the *Installation and Configuration Guide* for your MessageQ server platform.

Adding or Editing a Client Entry

The Security Table Client Entry dialog displays information about the resources the client node can access. Figure 3-8 shows the options for editing and creating client entries.

Figure 3-8 Security Table Client Entry Options

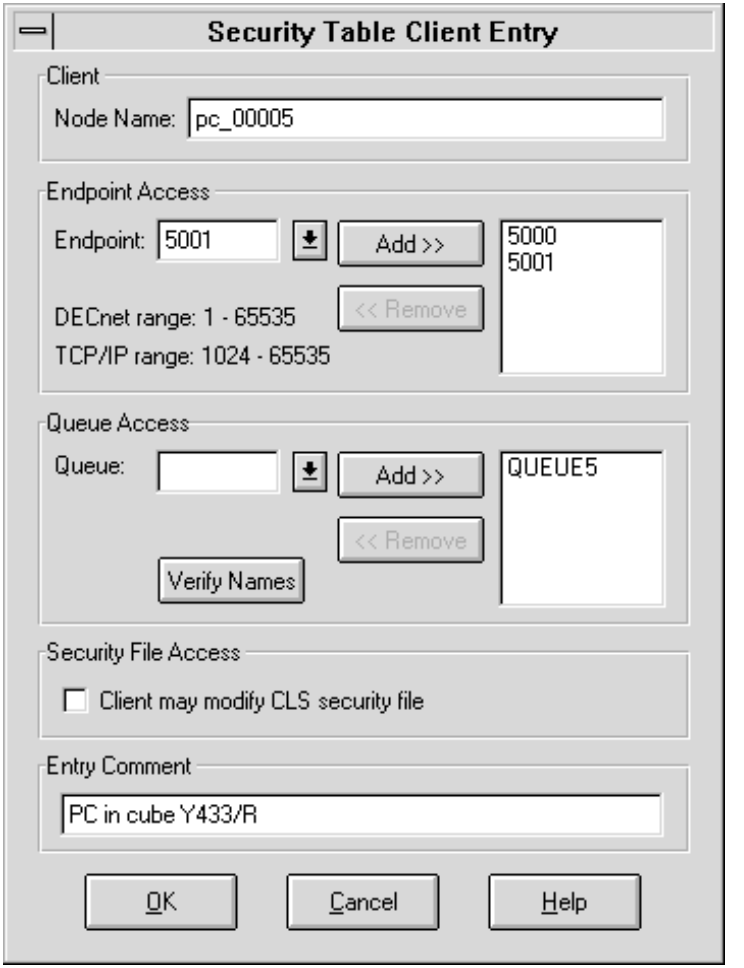


Table 3-8 describes the configuration options for security table client entries.

Table 3-8 Options to Configure Security Table Client Entries

Option	Description
Client Node Name	The node name is usually the TCP/IP hostname or DECnet node name for the client. The node name may also be an IP Address (for example, 1.23.45.156) or an alias name as defined in the TCP/IP hosts table on the CLS server system. MessageQ Version 5.0 only supports TCT/IP as a network transport.
Endpoint Access	The endpoint access is defined by a list of 0 or more endpoints that this client can use. The endpoint list may contain up to 10 different endpoints. An empty endpoint list defaults to allowing access to any endpoint with no restrictions. Endpoints are added or removed from the endpoint list (to the right) by using the corresponding command buttons.
Queue Access	The queue access identifies a list of 0 or more queues that this client can use. The queue list may contain up to 10 different queues. An empty queue list defaults to allowing access to any queue with no restrictions. Queues can be identified by queue names or queue numbers. Clients can also be restricted to allow access only to temporary queues by selecting Temporary from the pull-down list box. Queues are added or removed from the queue list (to the right) by using the corresponding command buttons. The Verify Names button invokes a `pams_locate_q` operation for all the queue names in the queue list to check that they are defined in the MessageQ group configuration.
Security File Access	The security file access option is checked if the client node is granted access to replace and request a CLS reload of the CLS security file. Checking the box grants CLS_SECURITY_FILE access for the client node.
Entry Comment	The entry comment is an optional field to add information about the client node.

The client entry is added to the Client Node List on the main form when the OK button is selected. Changes to the security table do not become permanent until after the table is replaced. Changes to the security table do not become active until after a CLS load request, or the CLS server is restarted.

Maintaining Modification History

Modification history is a useful way to keep track of updates to the CLS security table. You can maintain the modification history for the CLS security table by using the Security Table Modification History dialog, as shown in Figure 3-9.

Figure 3-9 Security Table Modification History

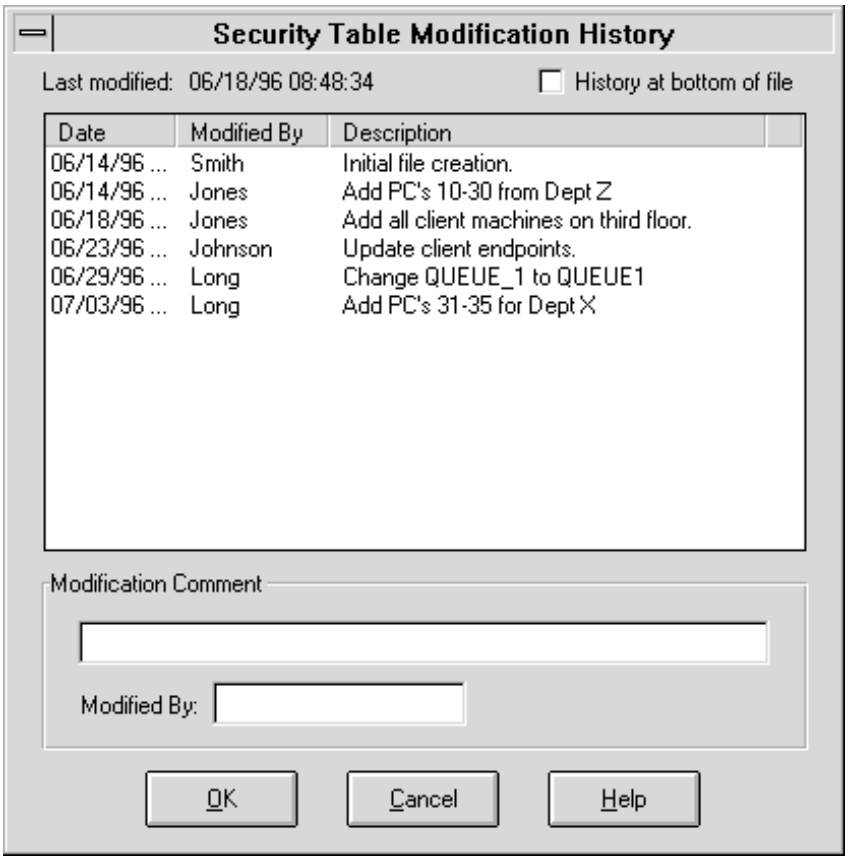

Security Table Modification History
Last modified: 06/18/96 08:48:34
History at bottom of file
Date
Modified By
Description
06/14/96 ... Smith Initial file creation.
06/14/96 ... Jones Add PC's 10-30 from Dept Z
06/18/96 ... Jones Add all client machines on third floor.
06/23/96 ... Johnson Update client endpoints.
06/29/96 ... Long Change QUEUE_1 to QUEUE1
07/03/96 ... Long Add PC's 31-35 for Dept X
Modification Comment
Modified By:
OK
Cancel
Help

Table 3-9 Options to Configure Security Table Modification History

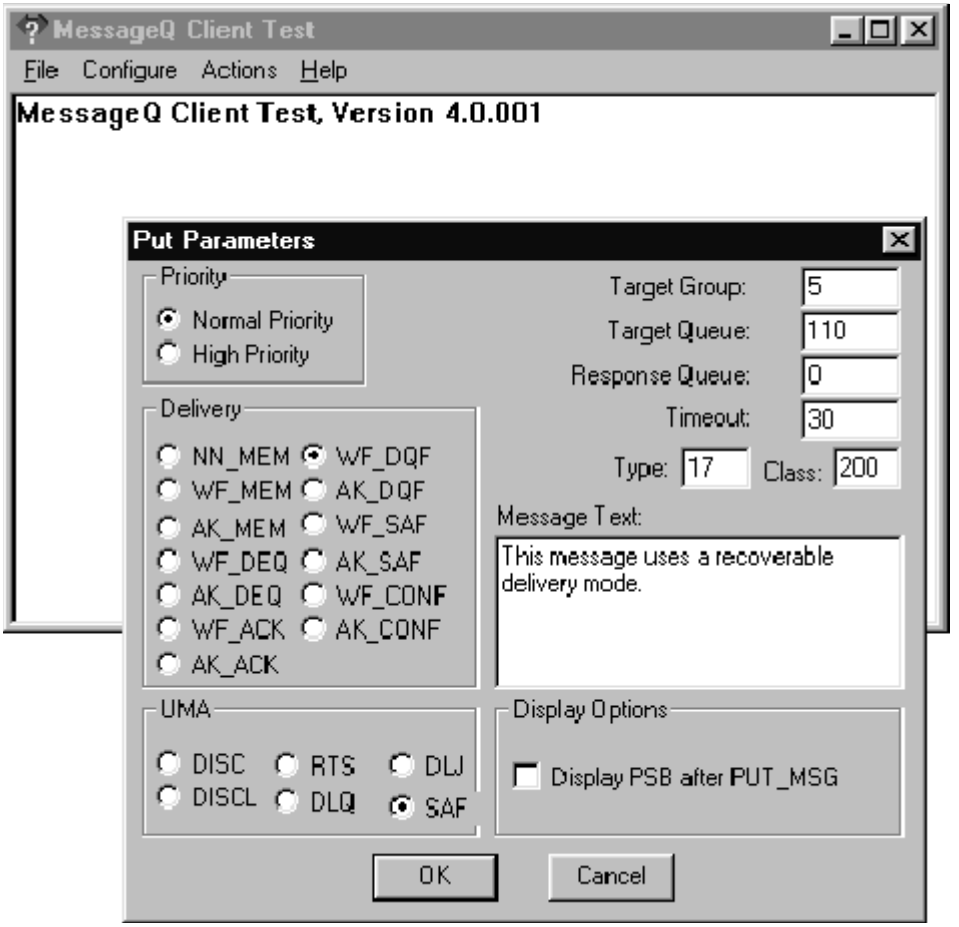
Option	Description
Last modified	Date and time the CLS Security Utility was last used to modify the CLS security file. The last modified time is set automatically when the security table is updated using the Replace option.
History at bottom of file	Check box that indicates how the modification history is written to the CLS security file. When the box is checked, the modification history is written to the file after all entries in the Client Node List. If the box is unchecked, the modification history is written at the beginning of the file before the Client Node List.
Modification Comment	Reasons for changes to the security table. Maximum field size of 80 characters.
Modified By	Author of the entry. Maximum field size of 10 characters.

Testing the Configuration Using the Test Utility

To test your newly configured MessageQ Client, run the MessageQ Client Test Utility for Windows (Figure 3-10).

The Test Utility allows you to interactively select the parameter options for individual calls to MessageQ. The program also allows you to test various MessageQ message delivery options and send messages to any process connected to the MessageQ bus. Use the Test Utility for unit-testing applications under development.

Figure 3-10 MessageQ Test Utility



4 Using the MessageQ Client

This chapter describes how to develop, run, and manage MessageQ Client applications. It contains the following topics:

- Overview of MessageQ Client Utilities
- Developing Your Application
- Building C, C++ and Visual Basic Applications
- Running Your Application
- Managing Your Application

Overview of the MessageQ Client Utilities

The MessageQ Client includes several utility programs for testing client applications and managing the MessageQ Client environment. The default location for the utilities is:

```
c:\Program Files\BEA Systems\BEA MessageQ\bin
```

Table 4-1 describes the MessageQ Client utilities.

Table 4-1 MessageQ Client for Windows Utility Programs

Utility Program	Filename	Description
Configuration Editor	`dmqconf.exe`	Defines the run-time configuration options
Test utility	`dmqtestw.exe`	An interactive application for sending and receiving messages
MRS utility	`dmqmrsu.exe`	Displays the contents of the local store-and-forward (SAF) journal
CLS Security utility	`dmqsecu.exe`	Provides a convenient user interface to review, modify, and update the CLS security file on the server system

Developing Your MessageQ Client Application

This section describes the following special considerations for developing applications to run on the MessageQ Client:

- MessageQ API functions supported by the MessageQ Client
- Limits on API parameter returns imposed by the MessageQ Client
- Contents and location of the MessageQ C/C++ include files
- Considerations for cross-group messaging between systems with different hardware data formats
- How to access the sample programs that come with the MessageQ Client

MessageQ API Support

Table 4-2 shows the API functions supported by the MessageQ Client. A small number of MessageQ API functions are available only for a specific environment and are not supported by the MessageQ Client. For example, the `pams_get_msga` function is available only on OpenVMS systems. To learn about the MessageQ API functions, see the *MessageQ Programmer's Guide*.

Table 4-2 MessageQ Client API Functions

API Function	Description
`pams_attach_q`	Connects a program to the MessageQ bus by attaching it to a message queue
`pams_bind_q`	Binds a temporary queue with a specified queue name
`pams_cancel_select`	Cancels selection of messages using a selection mask
`pams_cancel_timer`	Deletes the specified MessageQ timer
`pams_confirm_msg`	Confirms receipt of a recoverable message
`pams_detach_q`	Detaches a selected message queue, or all attached queues, from the bus
`pams_exit`	Terminates all attachments between the application and the MessageQ bus
`pams_get_msg`	Retrieves the next available message from a selected queue
`pams_get_msgw`	Waits until a message arrives in the selected queue, then retrieves the message
`pams_locate_q`	Requests the queue address for a specified queue name
`pams_put_msg`	Sends a message to a target queue
`pams_set_select`	Defines a message selection mask
`pams_set_timer`	Creates a timer that sends a message to the application when the timer expires

Table 4-2 MessageQ Client API Functions

API Function	Description
pams_status_text	Receives the severity level and text description of a user-supplied PAMS API return code
putil_show_pending	Requests the number of pending messages for a list of selected queues

MessageQ Client Function Parameter Limits

The MessageQ Client sets specific limits on function parameter values that allow very large arguments on MessageQ Server systems. The limits for the function parameters reduce the size of network messages exchanged between the MessageQ Client and the remote Client Library Server. Table 4-3 lists the functions, parameters, and their maximum values on the MessageQ Client.

Table 4-3 API Function Parameter Maximum Values

API Function	Parameter	Maximum Value
pams_attach_q pams_locate_q	q_name_len	32
pams_attach_q pams_locate_q	name_space_list_len	100
pams_put_msg pams_get_msg	msg_area_size	32,700 (see Note)
pams_detach_q	detach_q options	32
pams_set_select	num_masks	20
putil_show_pending	count	100

Note: Messages larger than 32,700 bytes can be sent or received by using the semantics for FML-based messages (PSYM_MSG_FML) or large messages (PSYM_MSG_LARGE). Refer to the *MessageQ Programmer's Guide* for information on how to send these kinds of messages.

Include Files for C and C++

The MessageQ Client provides include files for C and C++ language programs. The include files contain the MessageQ API function prototype declarations, return status codes, symbolic constants used for API parameters, and other declarations for using MessageQ message-based services. Table 4-4 lists the standard MessageQ include files, which are described in the *MessageQ Programmer's Guide*. The default location for these include files is:

```
c:\...\BEA Systems\MessageQ\include
```

Table 4-4 C Language Include Files

Include File	Contents
`p_entry.h`	Function prototypes and type declarations for the MessageQ API
`p_group.h`	Constant definitions for MessageQ message-based services
`p_msg.h`	Contains definitions for message-based services
`p_proces.h`	Constant definitions for MessageQ (for OpenVMS) processes
`p_return.h`	Return status values
`p_symbol.h`	Symbolic constants used for function parameters
`p_typecl.h`	Constant definitions of MessageQ message type and class for message-based services

MessageQ Client Return Codes

All MessageQ return codes are defined in the include file, `p_return.h`. Some of the return codes are specific to the MessageQ Client and are not returned to server-based applications. Table 4-5 lists the return codes specific to the MessageQ Client.

Table 4-5 MessageQ Client Return Codes

Return Code	Description
`PAMS__JOURNAL_FAIL`	The MRS service could not add messages to the local journal because of an operating system I/O error.
`PAMS__JOURNAL_FULL`	The MRS service could not add messages to the local journal because it is full.
`PAMS__JOURNAL_ON`	The link to the CLS is broken and the MRS service reports that journaling has begun.
`PAMS__LINK_UP`	The link to the CLS has been reestablished.
`PAMS__NETERROR`	The network connection to the CLS is broken.
`PAMS__NETNOLINK`	The network connection to the CLS is not available.
`PAMS__PREVCALLBUSY`	A previous MessageQ function call is still in progress.

There are platform-specific differences in the numeric values for the `p_return.h` return codes. The OpenVMS version of `p_return.h` contains numeric values different from those used on Windows NT, Windows 95, or UNIX. Client applications do not need to be concerned with these differences because the MessageQ Client returns status codes as they are defined on the client system, regardless of the system where the CLS is running.

It is recommended that programs use the symbolic value when testing the return status codes, rather than a numeric value. For example, use

```
if ( status == PAMS__NETNOLINK )
```

instead of,

```
if ( status == -278 ).
```

This improves code portability because of the platform-specific differences in the numeric values listed in `p_return.h`. It also makes code maintenance easier in the event that any status code numeric value is changed.

Byte Order Considerations for Application Developers

MessageQ provides the capability to send and receive messages between many different types of operating systems and CPU architectures. The byte order used by different CPU architectures is referred to as either little endian (or right-to-left order) or big endian (left-to-right order). Application designers must take into account the differences in byte ordering when designing a distributed application with MessageQ.

The byte order used on the MessageQ Client system and the CLS platform may be different. For example, a Windows PC with an Intel x86 CPU is a little endian machine and an HP PA-RISC system is a big endian machine. This means that integer values sent in the message area from the client are represented differently when received by the application server on the host.

The MessageQ Client and CLS handle differences in byte ordering by using network byte order when the Client and Server system are based on different representations (network byte order is big endian.). This ensures that the arguments to the MessageQ API functions called on the client are passed correctly to CLS platform to initiate the messaging operation.

Note: The MessageQ Client **does not** perform byte-swapping on the user data passed in the message area for `pams_put_msg` or `pams_get_msg` calls. Only MessageQ self-describing messages perform data marshaling between systems with unlike endian formats. Refer to the *MessageQ Programmer's Guide* for more information about how to use self-describing messages.

There are various techniques for handling the byte order differences in the client or server application components:

- One approach is to send user data messages containing only character string data. Integer values are converted to the corresponding character representation before they are sent in the message.
- Another approach is to design an application-specific interface for sending and receiving messages that implements marshaling routines for the user data contained in each message.

The data marshaling routines can be implemented as a set of library routines designed specifically to support data format conversion. These routines are typically written so that each marshal routine performs one specific record conversion. Standard socket routines are available to support byte-order conversion. These routines are `htonl`, `htons`, `ntohl`, and `ntohs`. For example, `htonl` means host-to-network long (32-bit) conversion.

The WinSock DLL is already included as a run-time component of Windows applications using the MessageQ Client. The WinSock DLL exports the standard byte-order conversion routines. Marshaling routines can call these functions by including the file `winsock.h`, and linking with the IMPORT library, `wsock32.lib`.

Sample Programs

The MessageQ Client is distributed with a number of sample application programs that demonstrate many features of the MessageQ API. If the sample programs were selected during installation, they will be located in the MessageQ installation directory tree. The default location is:

```
c:\...\BEA Systems\BEA MessageQ\examples\x
```

The sample programs consist of C language source modules. The sample programs are identical to the sample programs distributed with the MessageQ Server products, which demonstrates the portability of the MessageQ API across all supported platforms.

The sample programs can be built using an integrated development environment that provides a default Windows program shell, such as the Borland IDE, or Microsoft Visual C++ QuickWin. A make file is provided in the same directory for building the sample programs. Read the make file for details about the command line options.

To run any of the sample programs, make sure that the directories containing the MessageQ Client DLL and Windows Socket DLL files are identified by the PATH environment variable. The MessageQ Client Configuration for the Server host name and endpoint must be set for your environment. The Client Library Server (CLS) must be running on a MessageQ server system for the sample program to work.

On Windows 95 and Windows NT systems the sample programs are built as console applications.

Before building the sample programs, you may need to set the INCLUDE and LIB environment variables to point to the required MessageQ directories. Make sure that LIB points to `\...\BEA Systems\MessageQ\lib` and INCLUDE points to `\...\BEA Systems\MessageQ\include`. Refer to the makefile for the description of command line parameters.

It is also possible to build the samples using Visual C++ as follows:

1. Create a directory (e.g. `c:\x_example`) and copy the `x_*.c` programs from the `\...\BEA Systems\MessageQ\examples` directory to this new directory.
2. From Visual C++ create a new project. The New Project dialog asks for the project name and type. Enter the sample program name (e.g. `x_attnam`), and select a console application (for Visual C++ Version 2.x and later) as the project type.
3. Visual C++ then prompts for a list of files in the project. Enter the name of the desired sample program (e.g. `x_attnam.c`).
4. The project options need to be updated to link with the appropriate MessageQ Client import library. Use `dmqcl32.lib` for Windows NT and Windows 95. Set the project options as follows:

   Add `dmqcl32.lib` to the linker library list.
5. Build the application. This creates an executable program called *`project`*`.exe`. For example, `x_attnam.exe`.
6. Make sure the MessageQ group and CLS are running on your server platform. Use the Configuration Editor to identify the server host name, network transport, and endpoint number.
7. Run the application.

Building C and C++ Applications

This topic describes how to build your application linking C/C++ applications. Table 4-6 shows which MessageQ Client IMPORT library to use when linking a C/C++ application. The IMPORT library defines the MessageQ API functions exported by the DLL.

Table 4-6 MessageQ IMPORT Library

Environment	IMPORT Library	DLL
Windows NT or Windows 95	`dmqcl32.lib`	`dmqcl32.dll`

Include the IMPORT library to the link files used in an `nmake` makefile to resolve the external functions for the MessageQ Client. Add the IMPORT library to your project file when using an integrated development environment, such as Borland C/C++ or Microsoft Visual C++.

Building Visual Basic Applications

Visual Basic applications can use the MessageQ Client in two ways:

- By directly calling the MessageQ API functions in `dmqcl32.dll`, or
- By using the MessageQ Client Custom Controls. Refer to the MessageQ Client Custom Controls online help file for more information.

A complete definition of the MessageQ API is defined for Visual Basic development by the file `dmqapi.bas`. If Visual Basic Support is selected during the MessageQ Client for Windows installation, `dmqapi.bas` is installed along with the MessageQ Client Custom Controls (`dmqclcc.ocx`). The `dmqapi.bas` file defines the symbols, return status codes, and function declarations to call the MessageQ API functions in the DLL from Visual Basic. Add `dmqapi.bas` to your Visual Basic project.

Visual Basic Version 4.0 users should also read the file `vb4dll.txt` for information about using Visual Basic with DLLs. The `vb4dll.txt` file is installed in the Visual Basic Version 4.0 directory.

The MessageQ support for Visual Basic consists of the following:

- The MessageQ Queue Controls define the properties, events, and actions associated with a message queue.
- The MessageQ Msg Controls define the properties, events, and actions associated with sending or receiving MessageQ messages.
- A Visual Basic definition module, `dmqapi.bas`, declares the external functions and symbol definitions for the MessageQ Client DLL.
- Example programs in Visual Basic (either Version 3.0 or Version 4.0) use the MessageQ Custom Controls and call the MessageQ Client Library functions directly.

The MessageQ Custom Controls use the MessageQ API functions provided by the MessageQ Client DLL. The MessageQ Custom Controls also use the utilities provided with the MessageQ Client, including the MessageQ Configuration Editor and MessageQ Message Recovery Services tools.

Note: Visual Basic allows application developers to run the application interactively and pause or stop the application at runtime to switch back to design mode. Applications that initiate a MessageQ attach operation at runtime (typically during a Form Load event) will leave a TCP/IP connection open to the CLS when the developer uses the Visual Basic stop button to switch back to design mode. The CLS is unaware that the Visual Basic program has been stopped, so any existing queue attachments remain. To work around this condition, it is recommended that Visual Basic programs perform a `pams_exit` prior to the first `pams_attach_q`. It is convenient to do this in the Form Load event.

Visual Basic Sample Programs

Several Visual Basic sample programs are provided to demonstrate techniques for building MessageQ Client applications. The default location for the sample programs is:

```
c:\...\BEA Systems\MessageQ\example\vb
```

Table 4-7 lists the available sample programs.

Table 4-7 Visual Basic Sample Programs

Program	Description
`dmqfrmvb`	Calls the MessageQ Client API functions from Visual Basic
`dmqccsam`	Uses the MessageQ Client Custom Controls to send and receive messages
`confirm`	Uses the MessageQ Client Custom Controls to send, receive, and confirm recoverable messages
`sdmdemo`	Demonstrates the use of self-describing messages

It is recommended that developers copy the desired sample programs from the installation directory to a personal development directory before making any modifications. This makes it possible for the kit uninstall program to cleanly remove files from the installation directory.

Version 4.0 (32 bit) User-Defined Types

Visual Basic Version 4.0 (32-bit) programs may experience problems sending or receiving messages which consist of user-defined types because 32-bit Visual Basic Version 4.0 uses DWORD alignment (4 byte) for user-defined types, adding padding if needed. Note that this UDT alignment issue applies only to the 32-bit Visual Basic Version 4.0. All 16-bit versions of Visual Basic use byte-alignment for UDTs; therefore, padding is not added.

The only data type guaranteed to be instantiated and passed without additional padding is a byte array. However, Visual Basic does not provide a built-in mechanism for copying data between byte arrays and user-defined types.

A work-around to the problem of a copying between byte arrays and user-defined types (UDTs) can be implemented using a combination of temporary data types, `LSet`, and explicit byte-by-byte copies. The sample code in Listing 4-1 shows how this can be done for a UDT consisting of an SBS registration message. If you have problems with padding in user-defined types, contact Microsoft Visual Basic Technical Support at (206) 646-5105.

Listing 4-1 Temporary Data Type Code

```
‘ Visual Basic Version 4.0, 32-bit example to show the copy of a user defined
‘ type (UDT) to and from a byte array, omitting alignment padding bytes
‘ present in the UDT. Note that this is not necessary and should not be done
‘ in Visual Basic Version 4.0, 16-bit or Visual Basic Version 3.0.
‘ SBS registration message
Type SBS_Reg_Long_Form
       Version As Integer
       MOT_Addr As Integer
       Distribution_Q As Long
       Offset As Integer
       Operator As Byte
       Length As Integer
       Operand As Long
       Req_Ack As Byte
       Reg_SeqGap As Byte
       Reg_AutoDereg As Byte
End Type
#If Win32 Then
‘ SBS registration message sized temporary byte array (INCLUDING padding)
Type SBS_TMp
       byt (23) As Byte
End Type
‘ SBSRegToByteArray ()
‘
‘      Copy SBS registration message to a byte array
‘
Sub SBSRegToByteArray (sbs As SBS_Reg_Long_Form, byt () As Byte)
       Dim tmp As SBS_Tmp
‘ Use LSet to copy the SBS registration message to the temporary byte array
       LSet tmp = sbs
‘ Copy the temporary byte array to the return byte array, skipping pad bytes
       byt (0)  = tmp.byt (0)‘ Version
       byt (1)  = tmp.byt (1)‘      “
       byt (2)  = tmp.byt (2)‘ MOT_Addr
       byt (3)  = tmp.byt (3) ‘“
       byt (4)  = tmp.byt (4)‘ Distribution_Q
       byt (5)  = tmp.byt (5)‘“
       byt (6)  = tmp.byt (6)‘“
       byt (7)  = tmp.byt (7)‘“
```

```
       byt (8)  = tmp.byt (8)‘ Offset
       byt (9)  = tmp.byt (9) ‘ Operator
       byt (10) = tmp.byt (10)
              ‘ pad
       byt (11) = tmp.byt (12)‘ Length
       byt (12) = tmp.byt (13)‘“
              ‘ pad
              ‘ pad
       byt (13) = tmp.byt (16)‘ Operand
byt (14) = tmp.byt (17)‘“
       byt (15) = tmp.byt (18) ‘“
       byt (16) = tmp.byt (19)‘“
byt (17) = tmp.byt (20)‘ Req_Ack
       byt (18) = tmp.byt (21)‘ Req_SeqGap
       byt (19) = tmp.byt (22)‘ Req_AutoDereg
End Sub
‘ ByteArrayToSBSReg ()
‘
‘      Copy a byte array to a SBS registration message
‘
Sub ByteArrayToSBSReg (byt() As Byte, sbs As SBS_Reg_Long_Form)
       Dim tmp As SBS_Tmp
 ‘ Copy the input byte array to the temporary byte array, skipping pad bytes.
       tmp.byt(0)  = byt(0)  ‘ Version
       tmp.byt(1)  = byt(1)  ‘     “
       tmp.byt(2)  = byt(2)  ‘ MOT_Addr
       tmp.byt(3)  = byt(3)  ‘      “
       tmp.byt(4)  = byt(4)  ‘ Distribution_Q
       tmp.byt(5)  = byt(5)  ‘“
       tmp.byt(6)  = byt(6)  ‘“
       tmp.byt(7)  = byt(7)  ‘“
       tmp.byt(8)  = byt(8)  ‘ Offset
       tmp.byt(9)  = byt(9)  ‘    “
       tmp.byt(10) = byt(10) ‘ Operator
                  ‘ pad
       tmp.byt(12) = byt(11) ‘ Length
       tmp.byt(13) = byt(12) ‘    “
                  ‘ pad
                  ‘ pad
       tmp.byt(16) = byt(13) ‘ Operand
       tmp.byt(17) = byt(14) ‘     “
```

```
        tmp.byt(18) = byt(15) ‘     “
        tmp.byt(19) = byt(16) ‘     “
        tmp.byt(20) = byt(17) ‘ Req_Ack
        tmp.byt(21) = byt(18) ‘ Req_SeqGap
        tmp.byt(22) = byt(19) ‘ Req_AutoDereg
                ‘ Use LSet to copy the temporary byte array to the return SBS
                ‘ registration message.
                LSet sbs = tmp
End Sub
#End If
```

Running Your Application

This topic explains how to run your application with MessageQ Client. Before attempting to run a MessageQ Client application, verify that the TCP/IP connection between the MessageQ Client and Server are properly configured. Use the `ping` utility to check the TCP/IP connection (see the documentation for TCP/IP networking for your system for more information).

To use the MessageQ Client, your run-time environment must meet the following software requirements:

1. The MessageQ for UNIX or Windows NT product must be installed on a server system. A message queuing group must be configured to support the requirements of your messaging application environment. The MessageQ Client applications use messaging resources, including message queues, message buffers, and system resources on the server system. See the *Installation and Configuration Guide* for your MessageQ Server system and review the system resource requirements for using MessageQ in your environment.

2. When using the TCP/IP transport, the host names for the client and server systems must be properly identified in the host files on both the MessageQ Client and Server. Refer to your TCP/IP vendor documentation for the location of the host files.

Run-time Files

To execute a MessageQ Client application, the following files are used at runtime for Windows 95 and Windows NT.

- `dmqcl32.dll`
- TCP/IP
- `dmq.ini` (Optional. Configuration settings may also be stored in the Registry.)

These files are distributed along with the other DLLs, executable files, configuration files, and data files required to run your application. They must be installed in a directory identified by the PATH environment variable to enable Windows to load the DLLs correctly at runtime.

Using the MessageQ Client on Novell Networks

The MessageQ Client has been successfully tested with Novell LAN Workplace Version 4.2. The following configuration tips may be helpful for Novell users.

When installing LAN Workplace, select the following options:

- Any valid driver in the list. Do not select "None of these," otherwise TCP/IP is not installed and the network setup is not performed. If your driver is not listed, select one of the valid drivers (preferably an ETHERNET_II device). It will then be necessary to install your driver after LAN Workplace has been installed.
- Select ETHERNET_II for the frame type.

After installing LAN Workplace:

- Install the required driver, if necessary.
- LAN Workplace supplies `winsock.dll` and `tcpip.exe`, and places them in `\net\bin`. Make sure `\net\bin` is in your PATH.
- Copy the hosts file from `\net\sample\hosts` to `\net\hosts`. Edit the file to include both the server and client IP addresses and names.
- Check `\net\nwclient\net.cfg` to make sure all parameters are correct. Protocol and Frame should both be ETHERNET_II.

- If the driver was not provided by LAN Workplace, edit `\net\bin\lanwp.bat`. Add a command to start the alternate driver. For example:

  ```
  c:\net\nwclient\ewrk3.com
  ```

- Check the network connection with the IP Resolver in the LWP program group. Ping the server using both IP addresses and host name.

Managing Your Application

This topic describes the utilities, listed in Table 4-8, that are used to manage MessageQ Client applications.

Table 4-8 Utilities Used to Manage Client Applications

Utility	Filename	Description
MRS Utility	`dmqmrsu.exe`	Message Recovery Services (MRS) utility that allows you to view the contents of the store-and-forward (SAF) journals

MRS Utility

The MessageQ Client MRS utility lets you view the contents of local SAF journals. When a sender program running on the MessageQ Client sends a message marked as recoverable, it is written to the SAF journal on the client system. In the event that the recoverable message cannot be delivered to the CLS on the MessageQ Server and stored by the MessageQ message recovery system, it can be resent at a later time from the SAF journal on the MessageQ Client using this utility.

Figure 4-1 shows the MessageQ Client MRS utility.

Figure 4-1 MessageQ Client MRS Utility

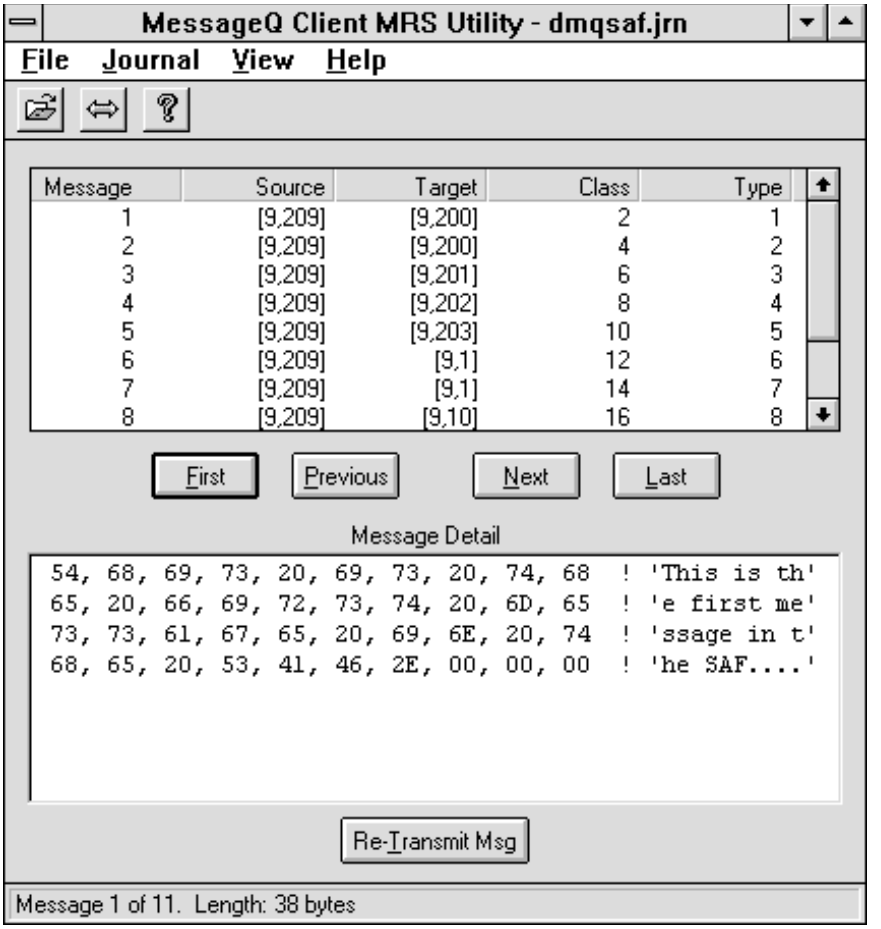


Table 4-9 describes the command buttons for the MRS utility.

Table 4-9 MRS Utility Command Buttons

Command Button	Function
First	View the first message
Previous	View previous message
Next	View next message

Table 4-9 MRS Utility Command Buttons

Command Button	Function
Last	View last message
Re-Transmit Msg	Messages can be retransmitted from the local journal by clicking on the Re-Transmit Msg button. You can retransmit multiple messages by selecting a group of messages from the message list. Retransmitting messages does not remove them from the local journal.

Messages contained in a local journal file are viewed by opening a `dmqsaf.jrn` file using the Open option on the File Menu (or the Open File toolbar icon). Opening the journal displays a list of messages currently stored in the file beginning with the first (oldest) message in the file. You can scroll though the messages in the file and display the contents.

Message header information, including message type, class, source, and target address is displayed for each message in the list. The data contents of the message are displayed in the Message Detail area using a hexadecimal byte array, with 10 bytes per line. The Message Detail display area also shows the ASCII printable characters. Selecting messages from the list displays the contents in the Message Detail area. You can navigate through the messages by scrolling or by using the command buttons.

5 Troubleshooting

This chapter describes how to identify and correct problems while running your application. It includes the following topics:

- Identifying Run-Time Errors
- Logging an Error Event
- Failing to Connect to the CLS
- Identifying Network Errors
- Tracing PAMS API Activity
- Tracing Client DLL Activity
- Tracing Message Activity
- Recovering from Client Crashes
- Loading Incorrect WinSock DLL at Runtime
- EAFNOSUPPORT Error

Identifying Run-time Errors

Problems at runtime can arise from a variety of error conditions. To identify and solve problems with the MessageQ Client, you can use several tools to track down the source of the problem. The following list provides some ideas to help you troubleshoot problems:

- Check the contents of the `dmqerror.log` file to get more information about the problem. Network errors are identified in the error log file.
- Use the trace output capability on the MessageQ Client and the Client Library Server (CLS) to get a detailed flow of activity that leads up to the problem. For shorter log files, use the PAMS_TRACE environment variable on the MessageQ Client.
- Restart Windows 95 or log out of Windows NT and then log back in to reload any DLLs that need to be reset.
- Use the `netstat` TCP/IP Utility on the PC at the MS-DOS prompt to monitor the network connections on the client. Also, use `netstat` on the server system to monitor the TCP/IP connections on the host system where the CLS is running.
- Try to repeat the error using the Test Utility included with the MessageQ Client. Reproducing problems with the Test Utility is an effective way to isolate application programming errors and provide a convenient way to test problems.

Logging an Error Event

Run-time errors detected by the Client Dynamic Link Library (DLL) are written to the file, `dmqerror.log`, in the default directory for the application. The errors indicate a run-time problem due to either a configuration error, an application error, network problem, or unexpected server response. Additional information can be found using the DMQCL_TRACE environment variable described in the Tracing Client DLL Activity topic.

You can enable or disable error event logging by changing the MessageQ Client Configuration Logging option. When error event logging is disabled, the `dmqerror.log` file is not used.

Failing to Connect to the CLS

The MessageQ Client attempts to establish a connection to the CLS in response to a call to `pams_attach_q`. The attempt to establish a connection fails immediately when:

- There is no CLS running on the system identified in the MessageQ Client default server configuration.
- The CLS is listening for connections on a different endpoint than the endpoint defined in the MessageQ Client default server configuration.

When the connection attempt fails, `pams_attach_q` returns the `PAMS__NETNOLINK -278` error status.

Check the file `dmqerror.log` for the host name and the endpoint of the server system with which the MessageQ Client attempted to connect. Either start the CLS on the server system or reconfigure the MessageQ Client default server.

Identifying Network Errors

Network errors result when the DLL receives an error when attempting to read or write on the network link. Occasional network connection problems can occur due to the state of the Windows TCP/IP protocol stack or the network connection to the host system. Network errors are identified by the return status from the `pams_attach_q` function, such as the following status value:

```
PAMS__NETNOLINK -278
```

Network connection errors might also occur when attempting to execute any of the MessageQ API functions. For example, the `pams_put_msg` and `pams_get_msg` functions return the following status value when the connection to the server is broken and MRS is not enabled:

```
PAMS__NETERROR -276
```

The specific steps for clearing the network error depend on how the problem developed. The following actions will generally clear the problem:

1. Check the error event log file, `dmqerror.log`, for a description of the error event.
2. Stop and restart the Windows application. In some cases, restarting the application or simply retrying the attach operation succeeds.
3. Exit the Windows Program Manager to shut down Windows and restart Windows (not a complete reboot) to unload and reload any DLLs that need to be reset.
4. Reboot the PC to completely reset the network drivers.
5. Stop and restart the CLS.

Tracing PAMS API Activity

To obtain a time-stamped output file showing the sequence of MessageQ function calls and return status codes, follow these steps:

1. Invoke the Configuration Editor.
2. Click on the **Tracing** button.
3. Check the Trace PAMS API calls option.

The information from the MessageQ API function call trace is written to the log file `dmqcldll.log` in the default directory for the application. The PAMS tracing option can be useful for observing the sequence of message function calls to determine the run-time behavior of the application.

Tracing Client DLL Activity

To obtain detailed, time-stamped traces of the MessageQ Client DLL execution events, follow these steps:

1. Invoke the Configuration Editor.
2. Click on the Tracing button.
3. Check the Trace client DLL activity option.

The information from the DLL trace might be useful to debug connection problems between client library applications and the CLS. The DLL trace output is written to the log file, `dmqcldll.log`, in the default directory for the application. Be aware that the tracing output can become very large over a long period of time.

A CLS server trace might be useful to get a detailed time-stamped activity of the client requests and MessageQ message operations performed by the CLS. For more information about trace output from the CLS, see the *Installation and Configuration Guide* for your MessageQ server platform.

Tracing Message Activity

Messages transmitted or received by the DLL can be logged to a file or the Event Watcher. The information from a message trace can be used to verify the quantity and content of messaging traffic. For more information about enabling message tracing, see the Configuring Logging topic in Chapter 3.

Recovering from Client Crashes

Applications occasionally crash (particularly during development) and do not have an opportunity to close or return resources in use prior to terminating. Applications using the MessageQ Client that are attached to the message queuing bus and then crash (or terminate) without calling `pams_exit` or `pams_detach_q` leave many resources allocated but not available for reuse. Resources that are in use after a client application crash include:

- Global memory allocated on behalf of the client application
- Network protocol resources; for example, sockets

- Network resources on the server system
- Message queue resources in use by the CLS on behalf of the client

After the client crashes, the server system still has an open connection to the client and the CLS remains attached to the primary queue used by the client. The network protocol keepalive mechanism does not notify the server that the client has gone away for a lengthy time period. Typically, you can reboot the client PC and the server still functions as though it has a connection open to the client.

If the Client for Windows application attaches to temporary queues, the CLS processes must be stopped manually. If the MessageQ Client application uses permanent queues, they will be able to reconnect and the previous CLS process will terminate.

Restarting the client application usually establishes a new connection to CLS. If network connect errors occur, follow the troubleshooting procedure described in the Network Errors topic. The procedure releases and frees all resources used by the client.

If the client application calls `pams_attach_q` using either ATTACH_BY_NAME or ATTACH_BY_NUMBER to attach to a specific primary queue, the CLS detects a client reconnect attempt and automatically terminates the CLS instance (server process or thread) attached to the same message queue. Reconnecting to the same queue is only accepted if the client application is attempting to reconnect from the same PC node as the previous connection.

If the client application calls `pams_attach_q` using the ATTACH_TEMPORARY attach mode, a new instance of the CLS is started to support the client reconnect. The previous instance of the CLS remains active. For information about terminating CLS servers, see the CLS topic in the *Installation and Configuration Guide* for your server platform.

Loading Incorrect WinSock DLL at Runtime

The WinSock DLL is loaded at runtime by the Windows Operating System. Under some circumstances the “wrong” WinSock can be loaded. This usually results in an error when a WinSock function is used.

Use the following procedure to verify which WinSock DLL is being loaded:

1. Using the MessageQ Client Config utility, enable the "Trace Client DLL activity". This can also be done by setting the DMQCL_TRACE environment variable.
2. Run your application and attempt a `pams_attach_q`. This traces client DLL activity to `dmqcldll.log`.
3. Open dmqcldll.log and search for the WSA Socket information. These lines contain a brief description of the WinSock DLL which is loaded at runtime.

When Windows loads the WinSock DLL, it searches the following directories using the following sequence:

1. The directory from which the application loaded
2. The current directory
3. Windows 95: The Windows system directory
   Windows NT: The 32-bit Windows System directory
4. The Windows directory
5. The directories listed in the PATH environment

If the incorrect WinSock DLL is being loaded, you need to check for a WinSock DLL in each of the previous locations. To resolve a conflict, it may be necessary to do one or more of the following steps:

- Copy the desired WinSock DLL to a directory earlier in the search sequence.
- Rename a conflicting WinSock DLL.
- Change the order of directories in the PATH.

Check with your WinSock DLL vendor for their recommendation on resolving these conflicts.

EAFNOSUPPORT Errors

An EAFNOSUPPORT error indicated that the WinSock DLL does not support the requested action. For example, this would occur if DECnet is selected as the transport but WinSock DLL does not support DECnet. This probably indicates that the wrong WinSock DLL is being loaded at runtime. Refer to the topic Loading Incorrect WinSock DLL at Runtime for instructions on correcting this condition.

Index

T

U

V

W