BEA MessageQ

Client for UNIX User's Guide

BEA MessageQ for UNIX Version 5.0
Document Edition 3.0
October 1998

Copyright

Copyright © 1998 BEA Systems, Inc. All Rights Reserved.

Restricted Rights Legend

This software and documentation is subject to and made available only pursuant to the terms of the BEA Systems License Agreement and may be used or copied only in accordance with the terms of that agreement. It is against the law to copy the software except as specifically allowed in the agreement. This document may not, in whole or in part, be copied photocopied, reproduced, translated, or reduced to any electronic medium or machine readable form without prior consent, in writing, from BEA Systems, Inc.

Use, duplication or disclosure by the U.S. Government is subject to restrictions set forth in the BEA Systems License Agreement and in subparagraph (c)(1) of the Commercial Computer Software-Restricted Rights Clause at FAR 52.227-19; subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013, subparagraph (d) of the Commercial Computer Software--Licensing clause at NASA FAR supplement 16-52.227-86; or their equivalent.

Information in this document is subject to change without notice and does not represent a commitment on the part of BEA Systems. THE SOFTWARE AND DOCUMENTATION ARE PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND INCLUDING WITHOUT LIMITATION, ANY WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. FURTHER, BEA Systems DOES NOT WARRANT, GUARANTEE, OR MAKE ANY REPRESENTATIONS REGARDING THE USE, OR THE RESULTS OF THE USE, OF THE SOFTWARE OR WRITTEN MATERIAL IN TERMS OF CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE.

Trademarks or Service Marks

BEA, BEA Builder, BEA Connect, BEA Jolt, BEA Manager, and BEA MessageQ are trademarks of BEA Systems, Inc. BEA ObjectBroker is a registered trademark of BEA Systems, Inc. TUXEDO is a registered trademark in the U.S. and other countries.

All other company names may be trademarks of the respective companies with which they are associated.

SNMP Agent Connection Reference Guide

Document Edition	Date	Software Version
3.0	October 1998	MessageQ Client for UNIX, Verison 5.0

Table of Contents

5. Troubleshooting

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 UNIX User's Guide is organized as follows:

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

- Chapter 2, “Installing the MessageQ Client,” describes how to install the MessageQ Client for UNIX software, including requirements, installation procedure, and postinstallation tasks.
- Chapter 3, “Configuring the MessageQ Client,” describes how to configure the MessageQ Client for UNIX, including information on configuring servers, automatic failover, logging, message recovery services, and tracing.
- Chapter 4, “Using the MessageQ Client for UNIX,” 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

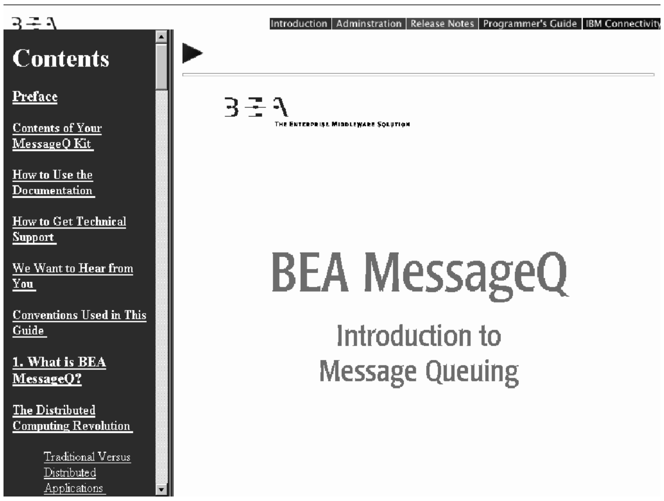
This document is designed primarily as an online, hypertext document. If you are reading this as a paper publication, note that to get full use from this document you should access it as an online document via the BEA MessageQ Online Documentation CD. The following sections explain how to view this document 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 later, or Microsoft Internet Explorer version 3.0 or later are recommended.

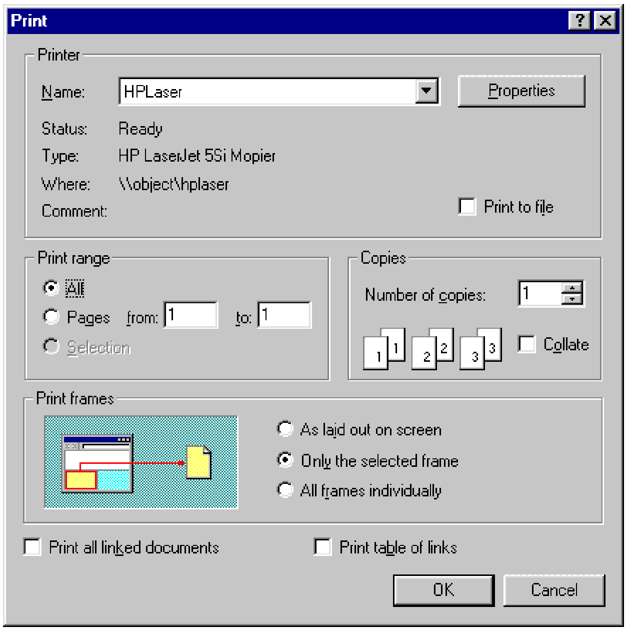
Figure 1 Online Document Displayed in a Netscape Web Browser


Introduction
Adminstration
Release Notes
Programmer's Guide
IBM Connectivity
Contents
Preface
Contents of Your MessageQ Kit
How to Use the Documentation
How to Get Technical Support
We Want to Hear from You
Conventions Used in This Guide
1. What is BEA MessageQ?
The Distributed Computing Revolution
Traditional Versus Distributed Applications
THE ENTERPRISE MIDDLEWARE SOLUTION
BEA MessageQ
Introduction to Message Queuing

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:



Print

Printer

Name: HPLaser

Status: Ready

Type: HP LaserJet 5Si Mopier

Where: \\object\hplaser

Comment:

[ ] Print to file

Print range: [● All] ○ Pages from: 1 to: 1 ○ Selection

Copies: Number of copies: 1 [ ] Collate

Print frames: ○ As laid out on screen [● Only the selected frame] ○ All frames individually

[ ] Print all linked documents

[ ] Print table of links

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.

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. *Examples*: 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 Client for Windows User's Guide

BEA MessageQ FML Programmer's Guide

BEA MessageQ Reference Manual

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?

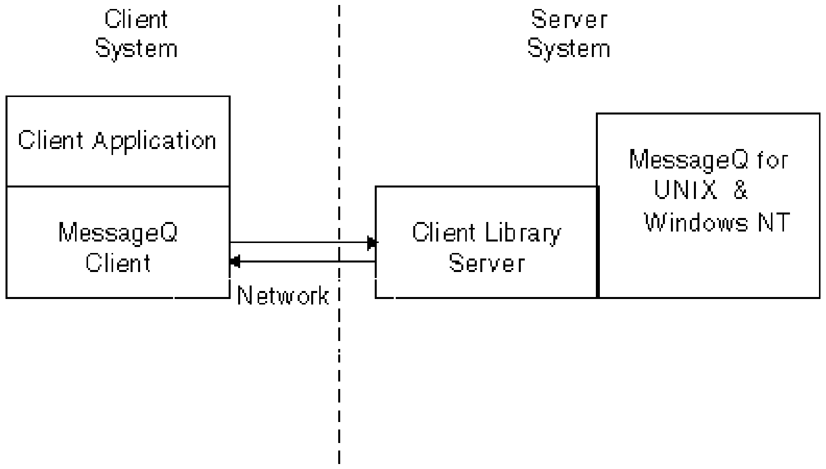
The MessageQ Client is a client implementation of the MessageQ Application Programming Interface (API). It provides message queuing support for distributed network applications using a MessageQ Server to provide reliable message queuing for distributed multi-platform network applications. The MessageQ 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, Windows 3.1, most popular UNIX systems, OpenVMS (MessageQ V4.0A), and MVS (MessageQ V3.2B) 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 re-established, 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, by using the MessageQ Client, a MessageQ Server need only be installed and configured 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.

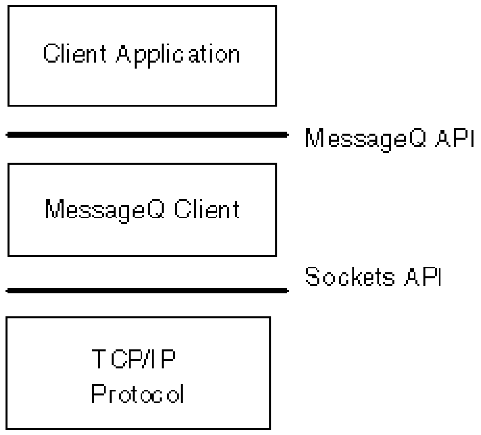
In this example, only one workstation needs to be sized and configured to optimize performance, reducing the burden of system management to a single machine. System management and configuration for the remaining systems is drastically simplified because managing the MessageQ Client consists mainly of 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 UNIX provides an archive library supporting MessageQ enabled applications. The MessageQ Client is available for many platforms including Digital UNIX (Alpha), AIX, HP-UX, and Solaris. Figure 1-2 shows the MessageQ Client for UNIX architecture.

Figure 1-2 MessageQ Client for UNIX 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 is 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 V4.0A), 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.

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 set 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 Registry (on Windows NT and Windows 95) or the `dmq.ini` configuration file (on Windows 3.1, UNIX, and OpenVMS) to identify the location of the CLS. 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.

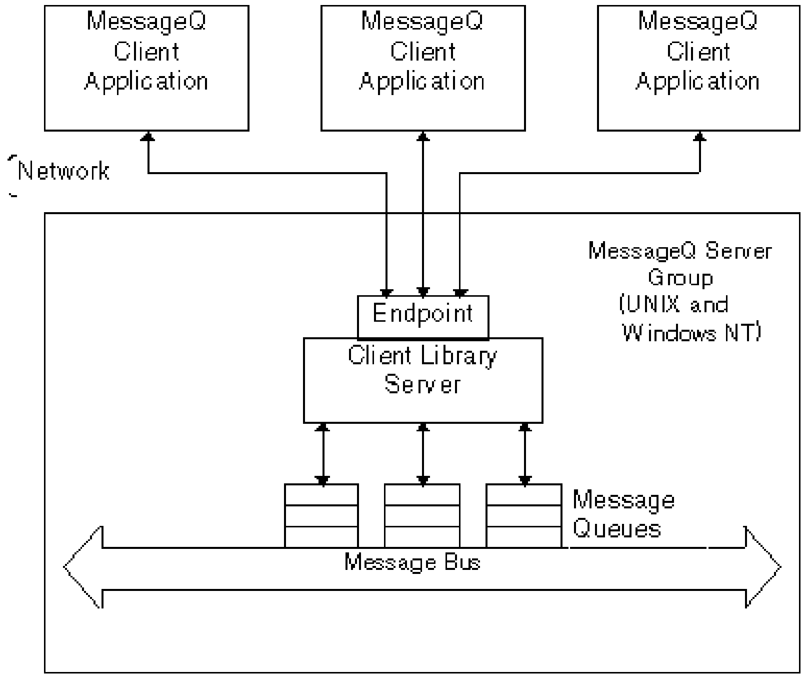
How the MessageQ Client and CLS Work Together

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 is 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.

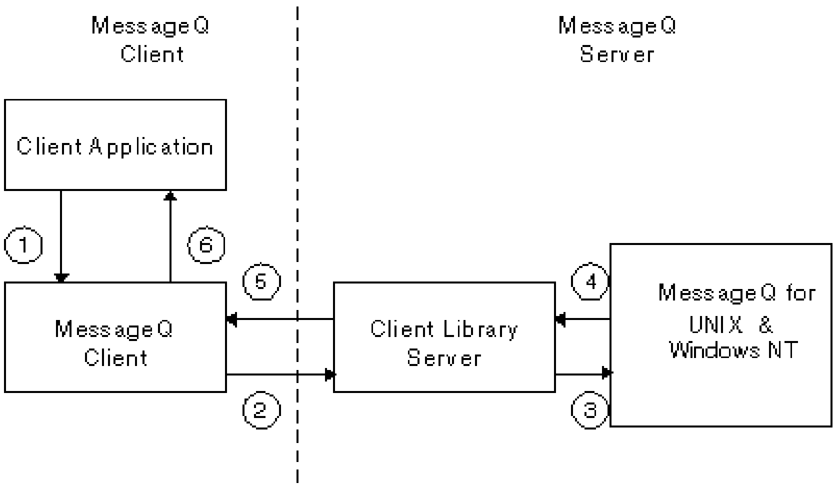
Figure 1-3 MessageQ Client and CLS Architecture


MessageQ Client Application
MessageQ Client Application
MessageQ Client Application
Network
MessageQ Server Group (UNIX and Windows NT)
Endpoint
Client Library Server
Message Queues
Message Bus

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



CHAPTER

2 Installing the MessageQ Client

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

- Preinstallation Requirements
- Installation Procedures for UNIX systems

Preinstallation Requirements

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

- Software
- Disk space
- System disk backup

Refer to the *BEA MessageQ Release Notes* for specific hardware and software environments that are supported by this product.

You may need to be able to log in as root (superuser) on the system where you are installing MessageQ if your system requires you to mount the cdrom drive as root. Otherwise, unless you need to uninstall a version of MessageQ prior to V5.0, or you want your new MessageQ installation to be owned by root, there is no need to be logged in as root to install MessageQ.

Any installation of BEA MessageQ with a version less than 5.0 must be uninstalled prior to using BEA MessageQ 5.0 or later. Prior to V5.0, only one version of BEA MessageQ was allowed to be installed on any single machine at a time, and the installation procedure symbolically linked files into the `/usr/bin`, `/usr/lib`, `/usr/man`, etc. directories. Starting with V5.0, installation files are no longer placed in any subdirectory of `/usr`, and multiple versions may be installed one a single machine as long as they are all V5.0 or later. If you do not uninstall MessageQ installations with versions less than 5.0 before you attempt to use V5.0 or later installations of MessageQ, you may experience serious operational problems. See the *BEA MessageQ Installation and Configuration Guide for UNIX* for instructions on uninstalling older versions of MessageQ.

Software Requirements

Your environment must meet the following software requirements to run install the MessageQ Client for UNIX software:

1. You must install MessageQ Server software on a UNIX or Windows NT system. In addition, a minimum of one message queuing bus and one message queuing group must be configured. The group must also be configured to run the MessageQ Client Library Server (CLS).

   For information on how to install and configure MessageQ Server software, refer to the installation and configuration guide for the platform that you are using.

2. Network software must be installed and running. TCP/IP networking is supported on all platforms.
3. If you intend to develop client applications, you must have a program development environment that allows you to compile and link your applications.

Disk Space Requirements

MessageQ Client for UNIX systems requires approximately 2 megabytes of free disk space to store the MessageQ Client 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 the system documentation for your server platform.

Choosing Installation Options

The MessageQ for UNIX installation dialogue displays a list of options that let you choose installation options. If you choose a package or option and then decide you don't want to install it, you can cancel your selection and redisplay the list of options. The installation options include:

- installing the MessageQ product or the MessageQ online documentation
- installing with or without BEA TUXEDOor BEA M3
- installing the client and server, or just the client

Table 2-1 describes the installation packages.

Table 2-1 MessageQ Packages

Installation Subset	Description
MessageQ	MessageQ software for UNIX platforms.
MessageQ Online Documentation	MessageQ Online Documentation in HTML format.

Table 2-2 describes the installation options for connecttivity with BEA TUXEDO..

Table 2-2 MessageQ Connectivity Options

Installation Subset	Description
BEA TUXEDO 6.4	Install MessageQ over an existing BEA TUXEDO V6.4 installation.
BEA M3 2.1	Install MessageQ over an existing BEA M3 V2.1 installation

Table 2-2 MessageQ Connectivity Options

Installation Subset	Description
None	Install MessageQ as a standalone product, with no BEA TUXEDO or BEA M3 connectivity.

Table 2-3 describes the installation options.

Table 2-3 MessageQ Installation Options

Installation Subset	Description
Both client and server	Installed on licensed systems used for developing or running MessageQ applications. This option includes the Client Library Server, include files, and examples. The programming examples that illustrate how applications can use interprocess message queuing to exchange information. For more information about the programming examples, refer to the *Programmer's Guide*.
Client Library	Provides remote client applications access to message queuing using MessageQ for UNIX. This option also installs the MessageQ include files.

Installing the MessageQ Client for UNIX Software

The following describes how to install MessageQ Client for UNIX software on UNIX systems from supported vendors. You can stop the installation procedure at any time by using the `-q` option or the terminal interrupt key sequence for your UNIX system (see your UNIX system documentation for a description of the terminal interrupt key sequence). If you stop the installation, files created up to that point are not automatically deleted. You must delete these files manually.

The steps to install the MessageQ Client for UNIX are as follows:

1. Mount the CD-ROM media.

Place the CD-ROM media in the CD-ROM tray and close the door. If your system automatically mounts your CD-ROM media when the door is closed, you may proceed to the next step.

Depending on your system, you may have to manually mount the CD-ROM media. You may have to do this logged in as root (superuser). If you must log in as root to mount the CD-ROM media, do so now.

If you do not have a standard procedure or tool for mounting CD-ROM media, use the following table of platform specific syntax information::

AIX	`mount -v cdrfs -r` *`device directory`*
Digital UNIX	`mount -t cdfs -r` *`device directory`*
HP-UX	`mount -F cdfs -r` *`device directory`*
NCR MP-RAS	`mount -F cdfs -r` *`device directory`*
SCO Unixware	`mount         -r` *`device directory`*
SCO OpenServer	`mount         -r` *`device directory`*
Sequent DYNIX	`mount         -r` *`device directory`*
Solaris	`mount -t hsfs -r` *`device directory`*

Note: When installing the MessageQ Client for UNIX on Solaris systems, ensure that the lsocket and lnsl libraries are present on the system. These libraries are required when compiling.

2. Run the installation script.

   Note that it is not necessary to be logged in as root (superuser) to install MessageQ as long as you have permission to write to the directory where you direct the installation script to install MessageQ.

   Before you run the installation script, move to the directory where the CD-ROM media is mounted. Assuming you mounted the CD-ROM media on directory `/cdrom`, you would issue the following command:

   ```
   %cd /cdrom
   ```

(If your CD-ROM media was automatically mounted, or you used a different tool or process than outlined in the previous step, you may have to move to a subdirectory of the mount point in order to successfully install MessageQ.)

After you have moved to the appropriate directory, determine the case of the installation script name. This depends on what options or tools you used to mount your CD-ROM media. You are looking for a file named install.sh or INSTALL.SH. Issue the following command to determine whether the script name is lower case or upper case:

```
%ls
```

If the script name is in lower case letters, issue the following command:

```
% sh ./install.sh
```

otherwise, issue the command like this:

```
% sh ./INSTALL.SH
```

A list of supported platforms (operating system and machine or processor type) is displayed.

```
01) aix41/rs6000     02) aix42/rs6000     03) aix43/rs6000
04) dux/alpha        05) dynix/i386       06) hpux10/hppa
07) hpux11/hppa      08) mpras/x86        09) sco/x86
10) sco_uw/x86       11) sol251/sparc     12) sol26/sparc
```

The following table provides detailed platform information :

`aix41/rs6000`	IBM AIX 4.1.4, RS/6000 SP2
`aix42/rs6000`	IBM AIX 4.2.1, RS/6000 SP2
`aix43/rs6000`	IBM AIX 4.3, RS/6000 SP2
`dux/alpha`	Digital UNIX 4.0, Alpha
`dynix/i386`	Sequent DYNIX ptx 4.4.1, Intel 386
`hpux10/hppa`	HP HP-UX 10.2, HPPA
`hpux11/hppa`	HP HP-UX 11, HPPA
`mpras/x86`	NCR MP-RAS 3.01 or 3.02, Intel x86
`sco/x86`	SCO OpenServer 5, Intel x86

`sco_uw/x86`	SCO UnixWare 2.1, Intel x86
`sol251/sparc`	Sun Solaris 2.5.1, SPARC
`sol26/sparc`	Sun Solaris 2.6, SPARC

3. Enter the number next to the selected installation platform:

```
Install which platform's files? [01-   12, q to quit, l for list]: 4

** You have chosen to install from dux/alpha **
```

4. Confirm your choice of platform:

```
BEA MessageQ 5.0

This directory contains the BEA MessageQ Core System for
Digital Unix 4.0 on DEC Alpha.

Is this correct? [y,n,q]: y
```

The following MessageQ packages are listed.

```
  1    msgq           BEA MessageQ
  2     msgqdoc         BEA MessageQ Online Documentation
```

5. Select the packages you wish to install:

```
Select the package(s) you wish to install (or 'all' to install
all packages) (default: all) [?,??,q]: 1
```

Copyright and trademark information is displayed:

```
BEA MessageQ
(alpha) Release 5.0
Copyright (c) 1998 BEA Systems, Inc.
Portions * Copyright 1986-1998 RSA Data Security, Inc.
All Rights Reserved.
Distributed under license by BEA Systems, Inc.
MessageQ is a registered trademark of BEA Systems, Inc.
TUXEDO is a registered trademark.
```

6. Select one of the following connectivity options in respect to BEA TUXEDO:

```
The following connectivity options are available:
  1    tux64          Install On Top Of BEA TUXEDO v6.4
  2    tux65          Install On Top Of BEA TUXEDO v6.5
  3     none            Install Without BEA TUXEDO

Select an option (default: none) [?,??,q]: 3
```

7. Select one of the following connectivity options in respect to other BEA Systems products:

```
The following connectivity options are available:

  1     tux64           Install On Top Of BEA TUXEDO v6.4
  2     m3_21           Install On Top Of BEA M3 v2.1
  3     none            Install Without BEA TUXEDO

Select an option (default: none) [?,??,q]: 2
```

Note: The BEA M3 V2.1 installation option is only available on platforms where BEA M3 V2.1 is supported.

8. Select one of the following installation options with respect to client and server environments:

```
The following installation options are available:

  1     both            Server and client
  2     client          Client only

Select an option (default: both) [?,??,q]: 2
```

9. Specify the directory where MessageQ files are to be installed:

```
Directory where MessageQ files are to be installed [?,q]:
/opt/messageq
Creating /opt/messageq
```

The system determines if sufficient space is available in the installation directory, unloads the files for the selected installation options, and sets file permissions:

```
Determining if sufficient space is available ...
2000 blocks are required
1361008 blocks are available to /opt/messageq

Using /opt/messageq as the MessageQ base directory
```

```
.
.
.
Changing file permissions...
.. finished

Installation of BEA MessageQ was successful

Please don't forget to fill out and send in your registration card
```

Before you unmount your CD-ROM media, don't forget to move out of the /cdrom directory or you will get a message that the /cdrom device is busy:

```
% cd /
% umount /cdrom
```

Once MessageQ software is installed, see the Postinstallation Tasks topic in the *MessageQ for UNIX Installation and Configuration Guide*.

Recovering from Errors During the Installation

If errors occur during the MessageQ Client installation procedure, recheck your preinstallation steps to ensure that the correct versions of prerequisite software have been installed. Errors can occur during the installation if the following conditions exist:

- ♦ The operating system version is not supported by MessageQ Client.
- ♦ Sufficient disk space is not available.
- ♦ TCP/IP or networking software is not installed or configured (if you are installing over the network).
- ♦ You do not have write privileges to the directory into which you attempted to install MessageQ

For descriptions of the error messages generated by these conditions, see the system management documentation for the UNIX system that you are using. If an error occurs while installing MessageQ and you believe the error is caused by a problem with the MessageQ Client for UNIX software, call BEA Technical Support at the number provided in the Preface.

Adding the Initialization File Directory to Your PATH

The `dmq.ini` template file is installed in the `/templates` subdirectory.

Copy the template to a local directory or a directory shared by multiple users, then add that directory to your PATH. Refer to Chapter 3, “Configuring the MessageQ Client” for instructions on using the configuration utility to specify `dmq.ini` options.

CHAPTER

3 Configuring the MessageQ Client

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

Table 3-1 MessageQ Client for UNIX Configuration Options

Option	Description	Required?
Server	Default Server Network transport, server host name, and endpoint definition	Yes
Failover	Automatic Failover Server Network transport, 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
Tracing	Settings to enable runtime trace information about the API calls and Client library activity	No

To configure the MessageQ Client for UNIX, use the Client for UNIX Configuration Utility, `dmqclconf`. The configuration utility is started from the command line using the following command line format:

```
dmqclconf [-f file] [-l] [-v] [-h]
```

Refer to Table 3-2 for the command-line parameters.

Table 3-2 MessageQ UNIX Command Line Parameters

Option	Description
`-f` *`file`*	Specifies the `dmq.ini` file path. The default file is `./dmq.ini`. If this option is not used, dmqclconf searches the directories specified by the *PATH* environment variable, and opens the first dmq.ini file that it finds.
`-l`	Lists the current configuration settings (if used with the `-f` *`file`* options), or the default configuration settings
`-v`	Displays the MessageQ Client for UNIX Configuration Utility version number
`-h`	Displays a brief help message that describes the options for this command

See Listing 3-1 for the MessageQ Client for UNIX Configuration Utility Main Menu.

Listing 3-1 MessageQ Client for UNIX Configuration Utility Main Menu

```
Main Menu            (file: /usr/jones/messageq/dmq.ini)

  1  Open
  2  Configure
  3  List
  4  Save
  5  Exit

Enter Selection:
```

Refer to Table 3-3 for a description of the Main Menu options.

Table 3-3 MessageQ Client for UNIX Main Menu Options

Option	Description
Open	Opens a specific `dmq.ini` file
Configure	Configure the MessageQ Client for UNIX

Table 3-3 MessageQ Client for UNIX Main Menu Options

Option	Description
List	Lists the current (or the default) settings
Save	Saves the configuration changes or updates to the `dmq.ini` file
Exit	Exits the MessageQ Client for UNIX Configuration Utility

The Configuration Utility updates an initialization file, `dmq.ini`, that is used at run time by the MessageQ Client for UNIX library. The `dmq.ini` file can be shared by multiple MessageQ-enabled applications using the same general configuration. Individual copies of `dmq.ini` can be used to tailor the configuration for individual applications. The `dmq.ini` file can be stored in any of the following directories:

- ♦ Default working directory where the application is running
- ♦ Directory identified by the PATH environment variable

The location of the `dmq.ini` file determines whether the same configuration is shared by multiple applications. When the application attempts to attach to the MessageQ message queuing bus, the client library searches the directories in the order listed for a copy of the `dmq.ini` file. The `dmq.ini` file can be modified using any text editor; however, we recommend using the MessageQ Client for UNIX Configuration Utility.

To begin configuring the MessageQ Client for UNIX, select item 2, Configure, from the Main Menu. See Listing 3-2 for the MessageQ Client for UNIX Configure Menu.

Listing 3-2 MessageQ Client for UNIX Configure Menu Options

```
Configure Menu    (file: /usr/jones/messageq/dmq.ini)

  1  Server
  2  Failover
  3  Logging
  4  MRS
  5  Tracing
  6  Previous Menu

Enter Selection:
```

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 server is used for all connections to the message queuing bus. If automatic reconnection is enabled, 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 (optional)

  If the primary default server is not available, the MessageQ Client for UNIX 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 are attempting to connect 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 for UNIX 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 SAF journal are sent to the CLS before new operations can be performed. For example, when a `pams_get_msg` triggers the reconnect threshold and a successful automatic reconnect and attach operation completes, the SAF journal is completely drained before the `pams_get_msg` function call returns. See Listing 3-3 for the default server configuration options.

Listing 3-3 Default Server Options

```
Server Configuration
  Network Transport Type (TCP/IP) [TCP/IP]:
  Server Hostname [arches]: dmqsrv
  Server Endpoint [5000]:
  Reconnect Interval (# of messages) [0]:
```

Refer to Table 3-4 for the default server configuration options.

Table 3-4 Configuring the Default Server

Option	Description
Network Transport Type	The network-level transport used to send messages to the MessageQ CLS. MessageQ supports TCP/IP as a network transport.
Server Hostname	The name of the host running the MessageQ CLS. The *hostname* must have a corresponding entry in the local `hosts` file. Refer to your network documentation for additional information on the location of these files.

Table 3-4 Configuring the Default Server

Option	Description
Server Endpoint	The endpoint used by the MessageQ CLS. For more information about specifying the endpoint, see the CLS section of the Installation and Configuration Guide for your server platform.
Reconnect Interval	The number of message operations that occur before the server attempts to reconnect. If set to 0, automatic reconnect is not enabled. If set to greater than 0, automatic reconnect is enabled. The MessageQ Client for UNIX attempts to reconnect to the server using the Reconnect Interval option as the threshold for making a new connection attempt. Any messaging operation call increments the count used to determine when to attempt another reconnect. When the number of operations attempted reaches the Reconnect Interval threshold, a reconnect attempt is made. Applications can choose to use a higher reconnect value to store messages in the local journal for forwarding at a later time.

Automatic Failover Server

If the primary (default) server is not available and automatic failover is enabled, the MessageQ Client for UNIX provides the option of connecting to a failover CLS. The failover options are ignored if automatic reconnect to the default server is enabled.

By enabling automatic failover, a MessageQ Client for UNIX 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 made only during a call to `pams_attach_q`.

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 for UNIX 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 for UNIX 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 MessageQ Clients for UNIXs attach to a temporary queue and use 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 for UNIX 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.

Automatic failover is not appropriate for all applications. 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 for UNIX.

On the other hand, the previous scenario could use failover by making the MessageQ server group (and all disk-based queuing resources) also fail over to another system so that messages previously sent to the MessageQ Client for UNIX are received after a failover transition. See Listing 3-4 for the automatic failover server configuration options.

Listing 3-4 Automatic Failover Server Options

```
Failover Configuration
Enable Automatic Failover (yes/no) [no]: y
  Network Transport Type (TCP/IP) [TCP/IP]:
  Server Hostname [oquirh]: dmqbck
  Server Endpoint [5000]:
```

Refer to Table 3-5 for a description of the automatic failover server configuration options.

Table 3-5 Configuring 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 Reconnect Message Interval option (see Table 3-3) must be greater than 0.
Network Transport Type	The network-level transport used to send messages to the failover server. MessageQ supports TCP/IP as a network transport.
Server Hostname	The name of the host running the MessageQ CLS. The *hostname* must have a corresponding entry in the local `hosts` file.
Server Endpoint	The endpoint used by the MessageQ CLS. For more information, see the startup information for the CLS in the *MessageQ Installation and Configuration Guide* for your server platform.

Configuring Logging

The MessageQ Client for UNIX allows you to log error events and messages a log file, as well as to trace messages and write the output to a trace file. All log files are located in the current working directory for the application.

Message logging allows you to obtain a complete history of the messaging activity of your application. Tracing messages to a file is an effective way to monitor the run time behavior of the application. See Listing 3-5 for the logging configuration options.

Listing 3-5 Logging Options

```
Logging Configuration
```

```
Log Error Events (yes/no) [yes]: y
Log Messages Sent to Trace File (yes/no) [no]:
Log Messages Received to Trace File (yes/no) [no]:
```

Refer to Table 3-6 for the message logging and message tracing configuration options. Note that you must perform a `pams_attach_q` operation for any of the "Log Messages" options to take effect.

Table 3-6 Configuring Message Logging

Option	Description
Log Error Events	If set to yes, logs error events to the file `dmqerror.log`. The default behavior is to log error events. When error event logging is enabled, connection errors to the CLS also log the full file path of the configuration file used at the time of the connection attempt. This can help identify problems due to multiple copies of the configuration file.
Log Messages Sent To Trace File	If set to yes, sends a copy of MessageQ messages sent by the application to the `dmqtrace.log` message log file
Log Messages Received To Trace File	If set to yes, sends a copy of MessageQ messages received by the application to the `dmqtrace.log` message log file

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 re-established, 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. See Listing 3-6 for the MRS configuration options.

Listing 3-6 MRS Configuration Options

```
MRS Configuration
  MRS Enabled (yes/no) [yes]:
  Journal File Path [./]:
  Journal File Size (bytes) [48000]:
  Cycle Journal File Blocks (yes/no) [yes]: n
  Fixed Size Journal File (yes/no) [yes]:
  Preallocate Journal File (yes/no) [yes]:
```

Refer to Table 3-7 for the MRS configuration options.

Table 3-7 MRS Configuration Options

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 set to yes, the journal *cycles* (reuses) disk blocks when full and overwrites previous messages. The Cycle Journal Blocks file 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.

Table 3-7 MRS Configuration Options

Option	Description
Fixed Size Journal File	Determines if the journal size is fixed or allowed to grow. If Cycle Journal Blocks is set to yes, 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 set to yes, 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. When calculating this value, add 80 bytes to the largest user message (because an 80-byte MRS message header is written to the journal for each user message).

Configuring Tracing

Tracing can be a useful debugging tool, because it allows you to enable and disable MessageQ Client for UNIX processing activity trace output. The trace output may create large output files on your system, and should be used only to monitor specific application behavior. The trace output log files are located in the default working directory for the application. See Listing 3-7 for the tracing configuration options.

Listing 3-7 Tracing Configuration Options

```
Tracing Configuration
  Trace PAMS API Calls (yes/no) [no]: y
  Trace Client Library Activity (yes/no) [yes]: y
```

Refer to Table 3-8 for the tracing configuration options.

Table 3-8 Tracing Configuration Options

Option	Description
Trace PAMS API calls	If set to yes, logs API call activity to the file `dmqcldll.log`. The default is no tracing.
Trace Client library activity	If set to yes, traces the internal client library activity to the file `dmqcldll.log`. The default is no tracing.

Testing the Configuration Using the Test Utility

To test your newly configured MessageQ Client for UNIX, run the MessageQ Test Utility `dmqcltest`. The Test Utility is started from the command line using the following command-line format:

```
dmqcltest
```

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.

To use the Test Utility, you first set the parameters associated with an action, then you perform the action. For example, to attach to a queue, you set the desired attach parameters, then execute the attach action.

See Listing 3-8 for the Test Utility main menu options.

Listing 3-8 Test Utility Main Menu

```
Wed > dmqcltest
Main Menu
```

```
1 Parameters
2 Actions
3 Exit
Enter Menu Selection >> 1
```

Refer to Table 3-9 for the Test Utility Parameters and Actions menu Options.

Table 3-9 Test Utility Parameters and Actions Menu Options

Parameters Menu Options	Actions Menu Options
Attach Parameters	Attach Queue
Detach Parameters	Detach Queue
Locate Parameters	Locate Queue
Put Parameters	Put Message
Get Parameters	Get Message
Set Timer Parameters	Set Timer
Cancel Timer Parameters	Cancel Timer
View Current Parameters	View Current Parameters
Previous Menu	Previous Menu

The examples in the following figures show how to use the Test Utility to attach to a temporary queue and send a message to another queue. The steps shown by the examples are as follows:

Set the Attach parameters to specify a temporary primary queue (Listing 3-9)

Set the Put parameters for the message (Listing 3-10)

Attach to queue 206 in group 9 (Listing 3-11)

Put the message to queue 1 in group 9 (Listing 3-13)

Detach from the temporary queue (Listing 3-14)

Exit from the Test Utility (Listing 3-15)

Listing 3-9 Specify a Temporary Queue

```
Wed > dmqcltest
Main Menu
1 Parameters
2 Actions
3 Exit
Enter Menu Selection >> 1
Parameters Menu
 1 Attach Parameters
 2 Bind Parameters
 3 Detach Parameters
 4 Locate Parameters
 5 Put Parameters
 6 Get Parameters
 7 Set Timer Parameters
 8 Cancel Timer Parameters
 9 View Current Parameters
10 Previous Menu
Enter Menu Selection >> 1
SELECT ATTACH TYPE
       1) Attach Primary
       2) Attach Secondary
Select attach type [1] ?
SELECT ATTACH_MODE
       1) Attach by name
       2) Attach by number
       3) Attach temporary
Select attach mode [3] ?
```

Listing 3-10 Set the Put Parameters

```
Parameters Menu
 1 Attach Parameters
 2 Bind Parameters
 3 Detach Parameters
 4 Locate Parameters
 5 Put Parameters
 6 Get Parameters
 7 Set Timer Parameters
 8 Cancel Timer Parameters
 9 View Current Parameters
10 Previous Menu
Enter Menu Selection >> 5
SELECT PRIORITY
       1) Standard Priority
       2) High Priority
Select priority [1] ?
SELECT DELIVERY MODE
       1) PDEL_MODE_AK_xxx
       2) PDEL_MODE_NN_xxx
       3) PDEL_MODE_WF_xxx
Select deliver mode [2] ? 3
SELECT DELIVERY MODE
       1) PDEL_MODE_xx_ACK
       2) PDEL_MODE_xx_CONF
       3) PDEL_MODE_xx_DEQ
       4) PDEL_MODE_xx_DQF
       5) PDEL_MODE_xx_MEM
       6) PDEL_MODE_xx_SAF
Select delivery mode [5] ? 5
SELECT UMA
       1) PDEL_UMA_DISC
       2) PDEL_UMA_RTS
       3) PDEL_UMA_SAF
       4) PDEL_UMA_DLQ
       5) PDEL_UMA_DLJ
Select UMA [1] ? 1
Enter target group [9] ? 9
Enter target queue [205] ?
Enter response queue [0] ?
Enter timeout in seconds [30] ?
Enter message class [1] ? 12
Enter message type [-100] ? 34
Enter message text ? This is a test message.
```

Listing 3-11 Attach to Queue 206 in Group 9

```
Parameters Menu
 1 Attach Parameters
 2 Bind Parameters
 3 Detach Parameters
 4 Locate Parameters
 5 Put Parameters
 6 Get Parameters
 7 Set Timer Parameters
 8 Cancel Timer Parameters
 9 View Current Parameters
10 Previous Menu
 Enter Menu Selection >> 10
Main Menu
1 Parameters
2 Actions
3 Exit
Enter Menu Selection >> 2
Actions Menu
 1 Attach Queue
 2 Bind Queue
 3 Detach Queue
 4 Locate Queue
 5 Put Message
 6 Get Message
 7 Set Timer
 8 Cancel Timer
 9 View Current Parameters
10 Previous Menu
Enter Menu Selection >> 1
attached to queue 9.206
```

If the MessageQ Client is properly configured to communicate with the Client Library Server running on a MessageQ Server, the Test utility returns a success message indicating that the attached operation was successful. However, if a problem occurs when the MessageQ Client attempts to attach to the message queuing bus, an error message is displayed indicating the source of the problem as shown in Listing 3-12.

The PAMS_NETNOLINK return value is a common error condition that occurs when network communication between the MessageQ Client and the CLS has not been established.

Listing 3-12 PAMS_NETNOLINK Error

```
Actions Menu
 1 Attach Queue
 2 Bind Queue
 3 Detach Queue
 4 Locate Queue
 5 Put Message
 6 Get Message
 7 Set Timer
 8 Cancel Timer
 9 View Current Parameters
10 Previous Menu
Enter Menu Selection >> 1
PAMS_NETNOLINK, Communications link could not be established.
```

The PAMS_NETNOLINK error can be caused by a variety of conditions. Table 3-10 describes the potential causes of this problem and their resolution.

Table 3-10 PAMS_NETNOLINK Error

Condition	Resolution
The client configuration is incorrect.	Check the server configuration information in the dmq.ini file to make sure that: ♦ The host name for the CLS is spelled correctly. ♦ The network endpoint specified is correct. ♦ The network transport type is correct.

Table 3-10 PAMS_NETNOLINK Error

Condition	Resolution
The Client cannot determine the network address for the CLS.	Check the local hosts file or name server to make sure that the network address specified for the host system running the CLS is correct.
The CLS may not be running.	Check the MessageQ Server system that is running the CLS to be sure that: ♦ A MessageQ group is running. ♦ The CLS is running. ♦ The CLS that is running uses the network transport and endpoint specified in the server configuration file on the client.

The PAMS_NETNOLINK error is only one of the error conditions that can arise when using the Test utility to test your MessageQ Client configuration. Refer to Chapter 5, "Troubleshooting" for more troubleshooting information.

Listing 3-13 Put the Message to Queue 1 in Group 9

```
Actions Menu
 1 Attach Queue
 2 Bind Queue
 3 Detach Queue
 4 Locate Queue
 5 Put Message
 6 Get Message
 7 Set Timer
 8 Cancel Timer
 9 View Current Parameters
10 Previous Menu
Enter Menu Selection >> 5
put message to queue 9.205
Actions Menu
 1 Attach Queue
 2 Bind Queue
 3 Detach Queue
 4 Locate Queue
 5 Put Message
 6 Get Message
 7 Set Timer
 8 Cancel Timer
 9 View Current Parameters
10 Previous Menu
Enter Menu Selection >> 10
```

Listing 3-14 Detach from the Temporary Queue

```
Main Menu
1 Parameters
2 Actions
3 Exit
Enter Menu Selection >> 2
Actions Menu
 1 Attach Queue
 2 Bind Queue
 3 Detach Queue
 4 Locate Queue
 5 Put Message
 6 Get Message
 7 Set Timer
 8 Cancel Timer
 9 View Current Parameters
10 Previous Menu
Enter Menu Selection >> 3
detached from queue 9.205
```

Listing 3-15 Exit from the Test Utility

```
Actions Menu
 1 Attach Queue
 2 Bind Queue
 3 Detach Queue
 4 Locate Queue
 5 Put Message
 6 Get Message
 7 Set Timer
 8 Cancel Timer
 9 View Current Parameters
10 Previous Menu
Enter Menu Selection >> 10
Main Menu
1 Parameters
2 Actions
3 Exit
Enter Menu Selection >> 3
```

CHAPTER

4 Using the MessageQ Client for UNIX

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 and C++ 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 is in `/bin` in the MessageQ installation directory.

Refer to Table 4-1 for a list of the MessageQ Client Utilities.

Table 4-1 MessageQ Client for UNIX Utility Programs

Utility Program	Filename	Description
Configuration Editor	`dmqclconf.exe`	Defines the run-time configuration options
Test utility	`dmqcltest.exe`	An interactive application for sending and receiving messages
MRS utility	`dmqclmrsu.exe`	Displays the contents of the local store-and-forward (SAF) journal

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. Refer to the *MessageQ Programmer's Guide* for complete information on how to use each API function.

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 in which it can receive messages
`pams_bind_q`	Binds a queue name to a queue address at runtime
`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 message that requires explicit confirmation
`pams_detach_q`	Detaches a selected message queue, or all attached queues, from the message queuing bus
`pams_exit`	Terminates all attachments between the application and the MessageQ message queuing 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 reduces 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:

```
/usr/kits/DMx410/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, Windows 3.1, 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,

```
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.

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 in `/examples` in the MessageQ installation directory.

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 with the `make` file provided in `/examples` subdirectory. Copy the sample programs to a personal development directory before modifying any of the files.

The `makefile` defines a LIBS macro to specify whether the sample programs are built with the MessageQ server library or the MessageQ Client library. To build with the MessageQ Client, uncomment the line containing `-ldmqcl`.

For example,

```
#LIBS=-ldmq

LIBS=-ldmqcl
```

The `libdmqcl.a` provides support for TCP/IP networks. After editing the `makefile`, use the `make` command to build the sample programs.

The MessageQ Client provides the `libdmqcl.a` archive library for application development. Client applications must link with the library, as shown in Table 4-6.

Table 4-6 MessageQ Client for UNIX Link Library

Library	Link Option	Network Support
`libdmqcl.a`	`-ldmqcl`	Supports TCP/IP only

Note: When building MessageQ client applications on Digital UNIX systems, you must link against the library `libots.a` in addition to the MessageQ Client library as shown below:

```
#  cc myapp.c -ldmqcl -lots -o myapp
```

Running Your Application

This topic explains how to run your application with MessageQ Client. Before attempting to run a MessageQ Client application, verify the TCP/IP connection between the MessageQ Client and Server is 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 MessageQ for 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. If you are planning to use the TCP/IP transport, the host names for the client and server systems must be properly identified in the hosts files on both the MessageQ Client and Server. Table 4-7 shows the location of host files on all MessageQ Servers.

Table 4-7 Hosts File Location

Systems	Hosts File Location
UNIX	`/etc/hosts`
Windows 95	See your TCP/IP vendor documentation.
Windows NT	`c:\winnt\system32\drivers\etc\hosts`

For a complete description of MessageQ Server and TCP/IP transports supported by the MessageQ Client, see the Read Me First letter supplied as part of your media kit.

Run-time Files

The run-time configuration file, `dmq.ini`, is required to run a client application. This file can be located in the application working directory, or in one of the directories specified by the PATH environment variable.

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.

The MRS utility is started with the following command:
`dmqclmrsu [-h | -v | -d | -l | -n` *`message`* `| -t` *`message`*`] [-f` *`saf_path`*`]`

Table 4-9 MRS Utility Command Line Parameters

Command Switch	Description
`-h`	Displays a brief help message
`-v`	Display MRS utility version number
`-d`	Display journal file header details
`-l`	Brief display of all messages in the journal
`-n message`	Display detail for the specified message
`-t message`	Transmit the specified message
`-f saf_path`	Specifies the full file path to the desired journal file. The default is `./dmqsaf.jrn`

Listing 4-1 shows the MessageQ Client MRS utility using the `-l` and the `-n` options.

Listing 4-1 MessageQ Client MRS Utility

```
/usr/users/smith/dmq > dmqclmrsu -l

SAF journal: dmqsaf.jrn
 Msg     Source       Target      Class   Type   Pri   Size     Data...
-----  -----------  -----------  ------  ------  ---  ------  ------------
    1      0.0          5.300        66      99   0       13  ’first mess’
```

```
   2      0.0          5.300        66      99   0       14  'second mes'
   3      0.0          5.300        66      99   0       13  'third mess'
   4      0.0          5.300        66      99   0       14  'fourth mes'
   5      0.0          5.300        66      99   0       13  'fifth mess'
   6      0.0          5.300        66      99   0       13  'sixth mess'
   7      0.0          5.300        66      99   0       15  'seventh me'
   8      0.0          5.300        66      99   0       14  'eighth mes'
   9      0.0          5.300        66      99   0       13  'ninth mess'
  10      0.0          5.300        66      99   0       13  'tenth mess'

/usr/users/smith/dmq > dmqclmrsu -n 7

SAF journal: dmqsaf.jrn
  Detail of message: 7

  Source:      0.0      Priority: 0       Size: 15      Large_Size: 0
  Target:      5.300    Class:    66      Type: 99
  Resp Q:      0.0      Delivery: 29      UMA:  5       Timeout: 100

  Contents of message buffer:
   XB 73, 65, 76, 65, 6E, 74, 68, 20, 6D, 65   ! 'seventh me'
   XB 73, 73, 61, 67, 65                       ! 'ssage'
```

CHAPTER

5 Troubleshooting

This chapter describes how to identify and correct problems while running your MessageQ client applications. Troubleshooting includes the following topics:

- Determining the Version Number of the Client
- Identifying Run-time Errors
- Logging an Error Event
- Failing to Connect to the CLS
- Identifying Network Errors
- Tracing PAMS API Activity
- Tracing Client Library Activity
- Recovering from Client Crashes

Determining the Version Number of the Client

To obtain technical support, you must know the version number of the MessageQ Client software the you are running. To determine the version of the MessageQ Client for UNIX library, enable tracing of Client Library activity, run your application, and check the trace file `dmqcldll.log` for the version number.

Identifying Run-time Errors

Problems at run time can arise from a variety of error conditions. To identify and solve problems with the MessageQ Client for UNIX, you can use a variety of tools to track down the source of the problem. The following list provides some ideas to help you to help you troubleshoot the source of application 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 for UNIX and the Client Library Server (CLS) to get a detailed flow of the activity that leads up to the problem. For shorter log files, use the *PAMS_TRACE* environment variable on the MessageQ Client for UNIX.
- Use the TELNET Utility to log in to the remote system and run the MessageQ Monitor Utility. On UNIX systems, use the character-cell program, `dmqmonc`, to monitor MessageQ groups remotely.
- Use the `netstat` TCP/IP Utility 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 for UNIX. Reproducing problems with the Test Utility is an effective way to isolate application programming errors and provide a convenient way to test problems.

This chapter summarizes how to find and resolve application problems.

Logging an Error Event

Run time errors detected by the Client library are written to the `dmqerror.log` file 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.

Error event logging can be either enabled or disabled by changing the MessageQ Client for UNIX Configuration Logging option. When error event logging is disabled, the `dmqerror.log` file is not used and no information on error conditions is available. Refer to Configuring Logging in Chapter 3 for more information about trace file settings.

Failing to Connect to the CLS

The MessageQ Client for UNIX attempts to establish a connection to the CLS in response to a call to `pams_attach_q`.

When the connection attempt fails, `pams_attach_q` returns the following error status:

```
PAMS__NETNOLINK-278
```

Check the file `dmqerror.log` for the full path of the configuration file (`dmq.ini`) used, the host name, and the endpoint of the server system with which the MessageQ Client for UNIX attempted to connect. Refer to for additional information about the `PAMS_NETNOLINK` error.

Identifying Network Errors

Network errors result from the Client Library receiving an error when attempting to read or write on the network link. Occasional network connection problems can occur due to the state of the 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:

```
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 return code 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 application. In some cases, restarting the application or simply retrying the attach operation succeeds.
3. 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 Utility.
2. Choose Configure from the Main menu, then choose Tracing from the Configure menu.
3. Set the Trace PAMS API Calls option to yes.

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

Tracing Client Library Activity

To obtain detailed, time-stamped traces of the Client Library activity, follow these steps:

1. Invoke the Configuration Utility.
2. Choose Configure from the Main menu, then choose Tracing from the Configure menu.

3. Set the Trace Client Library Activity option to yes.

The information from the library trace might be useful to debug connection problems between client library applications and the CLS. The library trace output is written to the log file, `dmqcldll.log`, in the default directory for the application. Be aware that the output from tracing option 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, refer to the *Installation and Configuration Guide* for your MessageQ server platform.

Recovering from Client Crashes

Occasionally, applications crash (particularly during development) and do not have an opportunity to close or return resources in use before terminating. Applications using the MessageQ Client for UNIX 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, such as sockets
- Network resources on the server system
- Message queue resources used 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 keep-alive mechanism does not notify the server that the client has gone away for a lengthy time period. Typically, you can reboot the client system and the server still functions as though it has a connection open to the client.

Restarting the client application usually establishes a new connection to CLS. If network connect errors occur, follow the troubleshooting procedure described in the Identifying 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 host 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 instances of the CLS remains active. For information about terminating CLS servers, see the CLS topic in the *Installation and Configuration Guide* for your MessageQ server platform.