BEA MessageQ

MQSeries Connection User's Guide

BEA MessageQ MQSeries Connection V4.0A, V5.0
Document Edition 5.0
February 1999

Copyright

Restricted Rights Legend

Trademarks or Service Marks

BEA MessageQ MQSeries Connection User's Guide

Document Edition	Date	Software Version
5.0	February 1999	BEA MessageQ MQSeries Connection Version 4.0A and Version 5.0

Contents

Preface

1. Introduction to BEA MessageQ MQSeries Connection

2. Developing Message Queuing Applications

3. Configuring BEA MessageQ MQSeries Connection

4. Managing the BEA MessageQ MQSeries Connection Environment

A. Programming Examples

B. Messages

Index

Preface

Purpose of This Document

This document describes Versions 4.0A and 5.0 of the BEA MessageQ MQSeries Connection product and gives instructions for building BEA MessageQ MQSeries Connection applications. Version 4.0A is designed to interoperate with BEA MessageQ Version 4.0A and IBM MQSeries 5.0 applications. Version 5.0 is designed to interoperate with BEA MessageQ Version 5.0 and IBM MQSeries 5.0 applications.

Who Should Read This Document

This document is intended for system administrators, network administrators, and developers who are interested in integrating BEA MessageQ applications with IBM MQSeries applications using the BEA MessageQ MQSeries Connection product.

How This Document Is Organized

The *BEA MessageQ MQSeries Connection User's Guide* is organized as follows:

- Chapter 1, "Introduction to BEA MessageQ MQSeries Connection," defines message queuing, describes the BEA MessageQ and IBM MQSeries products, and describes how the BEA MessageQ MQSeries Connection product works.
- Chapter 2, "Developing Message Queuing Applications," describes how to design applications using the BEA MessageQ MQSeries Connection product. It

also describes how replies and requests are exchanged between BEA MessageQ and MQSeries applications.

- Chapter 3, “Configuring BEA MessageQ MQSeries Connection,” describes the tasks required to define queues and configure your system to support the BEA MessageQ MQSeries Connection product.
- Chapter 4, “Managing the BEA MessageQ MQSeries Connection Environment,” describes how to manage the BEA MessageQ MQSeries Connection environment, including starting the Queue Message Bridge (QMB), using the BEA MessageQ Connection Utility, using the `runmqsc` MQSeries Utility, and troubleshooting.
- Appendix A, “Programming Examples,” describes how to use the programming examples provided with the BEA MessageQ MQSeries Connection product. BEA MessageQ MQSeries Connection client/server programs and programs for testing BEA MessageQ MQSeries Connection installation and configuration are provided.
- Appendix B, “Messages,” describes the control messages that can be sent to the QMB processes using the BEA MessageQ MQSeries Connection utility.

How to Use This Document

This document, *BEA MessageQ MQSeries Connection User’s Guide*, 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 install and access it as an online document via a Web browser.

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

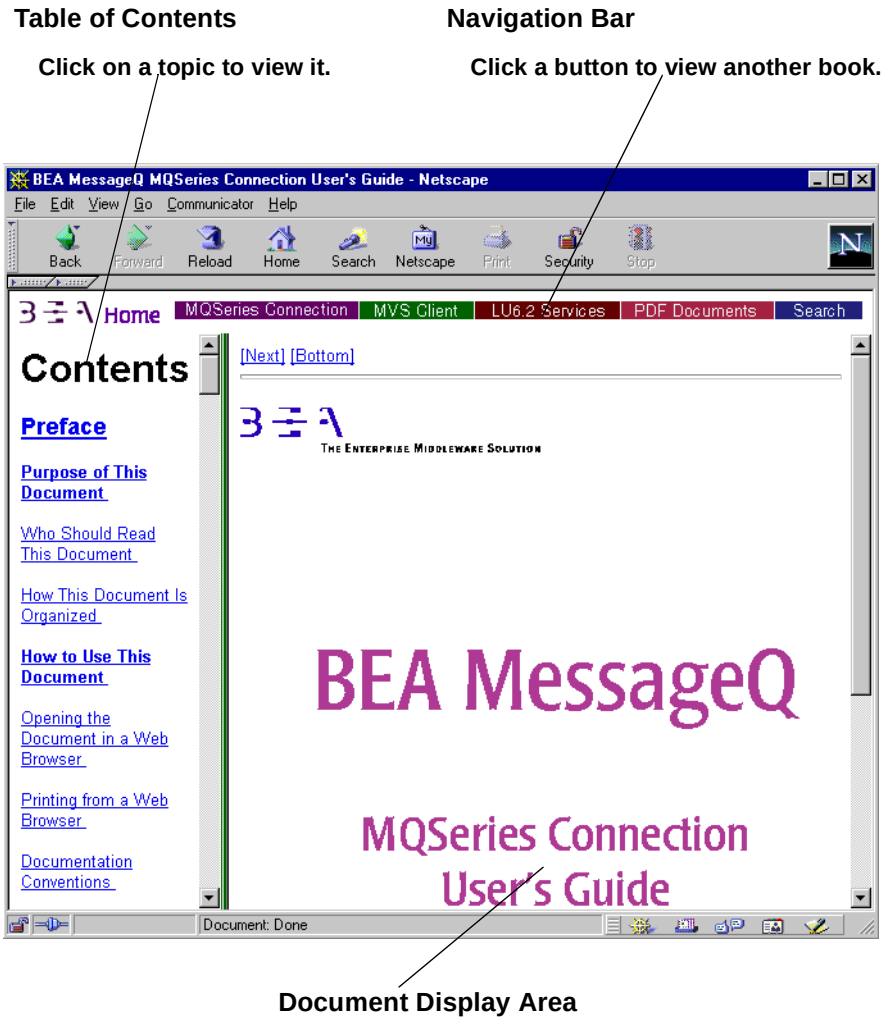
To access the online version of this document, open the following HTML file in a Web browser:

```
/doc/bmq/mqsc/usergde/index.htm
```

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 shows the online document with the clickable navigation bar and table of contents.

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.)

The BEA MessageQ Online Documentation CD and the BEA MessageQ MQSeries Connection product CD also include Adobe Acrobat PDF files of 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 in the BEA MessageQ Introduction to Message Queuing.
Ctrl+Tab	Indicates that you must press two or more keys sequentially.
italics	Indicate emphasis or book titles.

`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 stdio`
	`pams_attach_q`
	`\bmq\lu62_40a\include`
	`.htm`
	`bmq.doc`
	`BITMAP`
	`float`

Convention	Item
`monospace boldface text`	Identifies significant words in code. *Example*: `put_msg(msg_ptr, class, type)`
`monospace italic text`	Identifies variables in code. *Example*: `String expr`
UPPERCASE TEXT	Indicates device names, environment variables, and logical operators. *Example*s: LPT1 PATH 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*: `int32 pams_get_msg (msg_area, priority ... [sel_filter] [psb] [show_buffer]...)`
`|`	Separates mutually exclusive choices in a syntax line. The symbol itself should never be typed.
`...`	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*: `int32 pams_get_msg (msg_area, priority ... [sel_filter] [psb] [show_buffer]...)`
. . .	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 BEA MessageQ MQSeries Connection software.

MessageQ MQSeries Connection Documentation

The BEA MessageQ MQSeries Connection information set consists of the following documents:

BEA MessageQ MQSeries Connection Installation Guide

BEA MessageQ MQSeries Connection User's Guide

BEA MessageQ MQSeries Connection and MVS Client Release Notes, Versions 4.0A and 5.0

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.

BEA Publications

You may find the following documentation helpful when using BEA MessageQ MQSeries Connection. The following manuals describe the BEA MessageQ product:

BEA MessageQ Introduction to Message Queuing

BEA MessageQ Programmer's Guide

BEA MessageQ Installation and Configuration for Windows NT

BEA MessageQ Installation and Configuration for UNIX

BEA MessageQ System Messages

BEA MessageQ FML Programmer's Guide

BEA MessageQ Reference Manual

BEA MessageQ Client for Windows User's Guide

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.

Other Publications

You may also find the IBM MQSeries documentation helpful. For information on IBM MQSeries, see the IBM MQSeries Version 5.0 documentation set.

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 BEA MessageQ MQSeries Connection, or if you have problems installing and running BEA MessageQ MQSeries Connection, 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

1 Introduction to BEA MessageQ MQSeries Connection

To help you understand the BEA MessageQ MQSeries Connection product and the software technology behind it, this chapter discusses the following topics:

- Message Queuing
- Message Queuing Interfaces
- The BEA MessageQ Product
- The IBM MQSeries Product
- How Message Queuing Systems Work
- The Need for Integrated Message Queuing
- How BEA MessageQ MQSeries Connection Works

Message Queuing

Message queuing is a method of information exchange used by two or more cooperating processes, which directs messages to a memory- or disk-based queue as an intermediate storage point. Message queuing applications are generally designed so that messages flow in a request/reply fashion. After the messaging system accepts a

message from the application, the application is free to continue work. It is the responsibility of the messaging system to deliver the message to the target queue, or, if it cannot do so, to take the appropriate action.

Most message queuing applications use asynchronous processing to send and receive messages. The sending and receiving applications are uncoupled or loosely coupled at best. They read and operate on the messages independently. If your application requires a tightly coupled (synchronous) relationship between the sending and receiving applications, you must design and enforce the tight coupling in the application logic using predefined message flow protocols.

BEA MessageQ is the BEA Systems, Inc. implementation of a message queuing system. MQSeries is the IBM implementation of a message queuing system. BEA MessageQ MQSeries Connection provides a set of programming services that allow message exchange between the BEA MessageQ and IBM MQSeries message queuing systems.

Message Queuing Interfaces

Most messaging systems provide a message queuing interface that allows applications to access resources at remote locations. Applications access these resources through common calls that contain no communications protocol-specific variables. The message queuing interface is independent and isolated from any communications protocol implemented by the messaging system.

The communications engine of the messaging system is responsible for all of the physical communications protocols and message delivery. This insulation provides the application with a network-independent topology, and facilitates the development of a truly heterogeneous application environment.

Both the BEA MessageQ and IBM MQSeries messaging systems provide a **common application programming interface** (**API**). The API is a set of basic functions, in the appropriate language, that provide applications access to the messaging system resources. The use and format of the API functions remains constant across all environments supported by the messaging system. Therefore, properly designed applications are platform-independent.

The basic API for most messaging systems consists of functions that:

- Connect, open, and attach to the messaging system
- Dequeue a message from a queue
- Send a message to a queue
- Close or detach from the messaging system

In addition to these basic functions, the messaging system may offer advanced features such as message recovery, selective messaging, message broadcasting, and application development tools.

On BEA MessageQ systems, the API is called the PAMS API. For more information, see the *BEA MessageQ Programmer's Guide*. On IBM MQSeries systems, the API is called the message queuing interface (MQI). For more information, see the *MQSeries Application Programming Reference*.

The BEA MessageQ Product

BEA MessageQ software is a message queuing system for heterogeneous computing environments that eliminates the need to learn system-level communications software for sending and receiving messages. BEA MessageQ software offers a common mechanism for message exchange, called the BEA MessageQ **message queuing bus**, that provides an interprocess communications highway for all applications to send and receive messages.

After the application attaches to the BEA MessageQ message queuing bus, it can send and receive messages from applications running on any supported platform. BEA MessageQ software runs on most UNIX systems, OpenVMS, and Windows NT. All platforms use a common BEA MessageQ API.

BEA MessageQ software uses a client/server architecture to implement distributed communications. A **client** is a software module that requests services of a server. A **server** is a software module that responds to the client's request by providing the specified services.

Each BEA MessageQ environment requires a message server, a message routing system that distributes messages among remote nodes. A BEA MessageQ client can exchange messages with other applications on the message queuing bus (whether local or remote) through a message server.

For a complete explanation on how to install, configure, and use the BEA MessageQ product, see the BEA MessageQ documentation for your BEA MessageQ server product.

The IBM MQSeries Product

IBM MQSeries products enable applications to use message queuing to participate in **message-driven processing**. With message-driven processing, applications can communicate across the same or different platforms using the same kinds of messages; communication protocols are hidden from the applications.

Message-driven processing requires applications to be designed as discrete functional modules. Each module must be an application program with well-defined input and output parameters. An application program's input and output parameters can be shared with other application programs by being included in messages sent to queues.

Using the appropriate IBM MQSeries programming mechanisms, an application program can start executing as a result of one or more messages arriving on a queue. If required, the program can terminate when all messages in a queue have been processed. Message-driven processing allows you to build or modify applications more quickly than you can with other types of application.

IBM MQSeries implements a common application programming interface, called the **message queuing interface** (**MQI**), across all supported platforms.

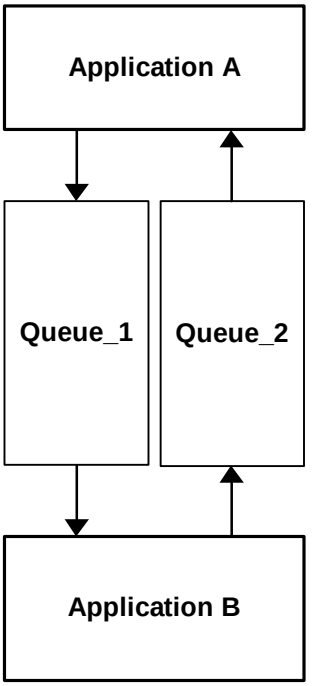
For more information on IBM MQSeries, refer to the IBM MQSeries documentation for the platform that you are using.

How Message Queuing Systems Work

Message queuing systems enable applications to communicate without "talking" directly with each other. Messages are placed on message queues for a target application to read. The target application reads the message and responds appropriately.

Figure 1-1 shows how a typical message queuing system, based on a simple request/reply paradigm, sends messages.

Figure 1-1 Typical Message Queuing System



The request/reply interaction shown here works as follows:

- Application A places a message (containing a request for information from Application B) on Queue_1.
- Application B reads the message on Queue_1.
- Application B places a reply on Queue_2. How Application B responds to the message depends on how the message and application are coded.
- Application A reads the message on Queue_2.

The Need for Integrated Message Queuing

Typically, message queuing systems do not allow applications to send messages to queues that are written using a different message queuing system. For example, BEA MessageQ applications cannot place messages on IBM MQSeries queues and vice versa. This lack of interoperability among message queuing systems can be a problem for businesses that merge and want to integrate information systems based on different message queuing systems.

BEA MessageQ MQSeries Connection solves the integration problem by allowing messages to be passed between BEA MessageQ and IBM MQSeries message queuing systems. For example, suppose two banks merge and need to integrate their information systems. Management decides to consolidate all account information on IBM systems that use IBM MQSeries for message exchange. In addition, management wants its customers to continue using their current ATM system. The ATM system receives requests for account information and dispatches requests to the server system using BEA MessageQ messages. Instead of rewriting either application, management decides to use BEA MessageQ MQSeries Connection to forward requests and responses between the two different message queuing systems.

How BEA MessageQ MQSeries Connection Works

BEA MessageQ MQSeries Connection provides a set of programming services that allow message exchange between the BEA MessageQ and IBM MQSeries V5.0 message queuing systems. Table 1-1 shows which versions of these messaging systems are supported by Versions 4.0A and 5.0 of the BEA MessageQ MQSeries Connection product.

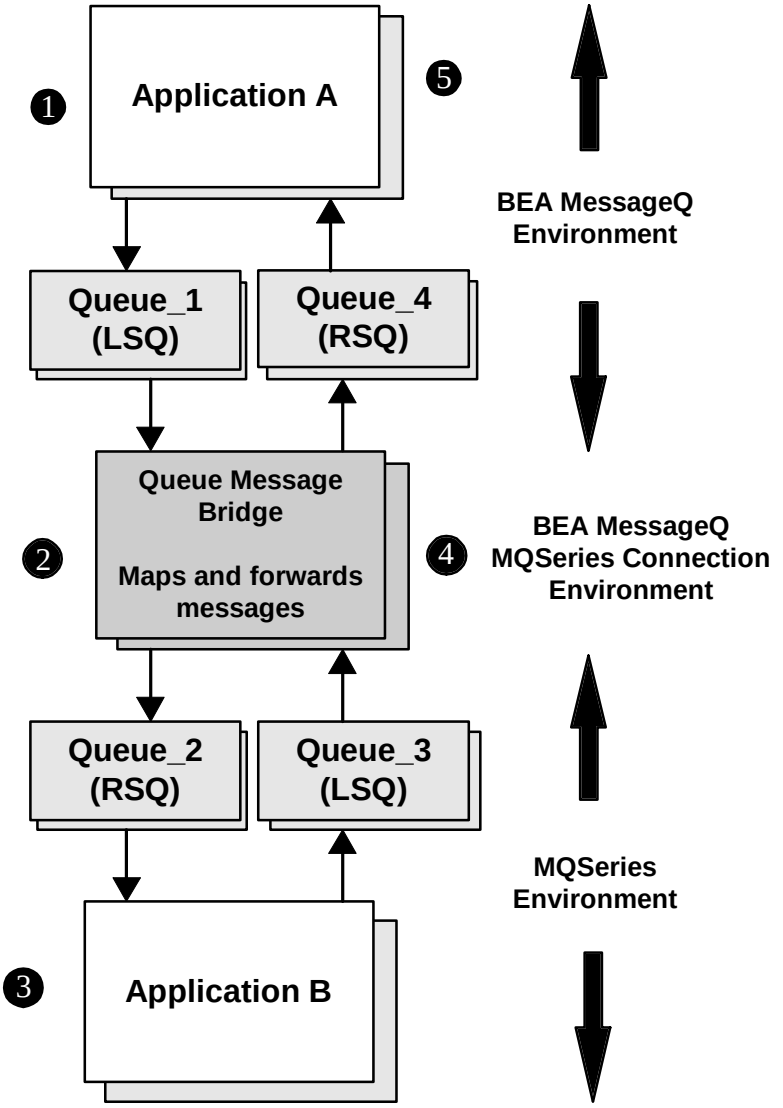
Table 1-1 BEA MessageQ MQSeries Connection Interoperability

BEA MessageQ MQSeries Connections Version	Works with...	
	BEA MessageQ Version	IBM MQSeries Version
4.0A	4.0A	5.0
5.0	5.0	5.0

Application developers can write their applications using either message queuing system. BEA MessageQ MQSeries Connection makes sure that messages are properly passed between the two message queuing systems, and that the message header information is in the proper format for the target application.

For example, suppose Application A is the ATM application that uses BEA MessageQ for message exchange. It is designed to receive requests for account inquiries, send these requests to Application B, and return account information to bank customers. Application B is an account lookup application that uses IBM MQSeries V5.0 for message exchange. Figure 1-2 shows how account inquiries are exchanged between the two messaging systems.

Figure 1-2 BEA MessageQ MQSeries Connection Overview



1. Application A places an account inquiry message on Queue_1, a BEA MessageQ queue.

2. The part of BEA MessageQ MQSeries Connection called the Queue Message Bridge reads the message on Queue_1. It maps the message header data into IBM MQSeries format and forwards the message to Queue_2, an IBM MQSeries queue.
3. Application B reads the message on Queue_2, looks up the requested account information, and places the account information in a reply message. The message is placed on Queue_3, an IBM MQSeries queue.
4. The part of BEA MessageQ MQSeries Connection called the Queue Message Bridge reads the message on Queue_3. It maps the message header data into BEA MessageQ format and forwards the message to Queue_4, a BEA MessageQ queue.
5. Application A reads the message on Queue_4 and displays the account information to the customer.

The LSQ (Local Service Queue) and RSQ (Remote Service Queue) designations are defined in "Communication Services Between BEA MessageQ and IBM MQSeries."

The use of message queuing for application development allows software developers to concentrate on the business needs of application development, without having to worry about the underlying network and communications programming requirements. BEA MessageQ MQSeries Connection eases the integration of mixed environments by allowing applications based on the BEA MessageQ and IBM MQSeries message queuing systems to exchange information without any new code.

Communication Services Between BEA MessageQ and IBM MQSeries

The part of BEA MessageQ MQSeries Connection that provides communication services between BEA MessageQ and IBM MQSeries applications is the Queue Message Bridge (QMB). The QMB provides the following:

- Resource services to connect, open, and attach to the remote messaging system
- Services to close, disconnect, or detach from the remote messaging system
- Queue and message services (such as send and receive functions) that can be used to forward messages between BEA MessageQ and IBM MQSeries queues

As shown earlier in Figure 1-2, the QMB creates a relationship between IBM MQSeries and BEA MessageQ queues in order to be able to forward messages between them. This relationship is based on the use of two QMB-specific entities referred to as the Local Service Queue (LSQ) and Remote Service Queue (RSQ):

- **Local Service Queue** (**LSQ**)—This is an intermediate queue for sending a message. An application always sends to an LSQ, which is defined in the local messaging system. Messages received on an LSQ are read by the QMB and forwarded to the associated RSQ. An LSQ can be either a BEA MessageQ or IBM MQSeries queue.
- **Remote Service Queue** (**RSQ**)—This is the final target queue for sending a message. This queue is defined by the remote messaging system. A QMB sends messages (retrieved from the associated LSQ) to this queue for subsequent processing by the target application. An RSQ can be either a BEA MessageQ or IBM MQSeries queue.

In the banking application described earlier in the "How BEA MessageQ MQSeries Connection Works" section, the LSQs and RSQs are used as follows:

- Queue_1 is a BEA MessageQ LSQ. This queue contains an account inquiry request with its message header data in BEA MessageQ format.
- Queue_2 is the IBM MQSeries RSQ. It contains the account inquiry request with its message header data in IBM MQSeries format.
- Queue_3 is an IBM MQSeries LSQ. It contains the account information reply with its message header data in IBM MQSeries format.
- Queue_4 is the BEA MessageQ RSQ. It contains the account information reply with its message header data in BEA MessageQ format.

The QMB uses the terms *local* and *remote* to reference its own view of the processing environment. The LSQ is owned and maintained by a QMB. The RSQ is owned and maintained by the partner messaging system and application service program. Message delivery between the LSQ and RSQ is asynchronous.

A requesting application, as defined in either BEA MessageQ or IBM MQSeries, formats and sends a message to an LSQ. The LSQ is under the control of a QMB server. The QMB server then maps and forwards the message to the associated RSQ on behalf of the requesting application.

If the local messaging system is BEA MessageQ, the LSQ is a multireader queue and directs messages from a BEA MessageQ application to an IBM MQSeries application. If the local messaging system is IBM MQSeries, the LSQ is a shared queue and directs messages from an IBM MQSeries application to a BEA MessageQ application.

The RSQ may be of any supported queue type, and reside anywhere in the messaging system that is accessible from the BEA MessageQ group or IBM MQSeries Message Queue Manager of the designated QMB process.

The QMB relies on an LSQ to RSQ relationship to know where to forward messages. This relationship is established in the QMB queue configuration file. For more information on this file, see "Configuring the Queue Message Bridge" in Chapter 3, "Configuring BEA MessageQ MQSeries Connection."

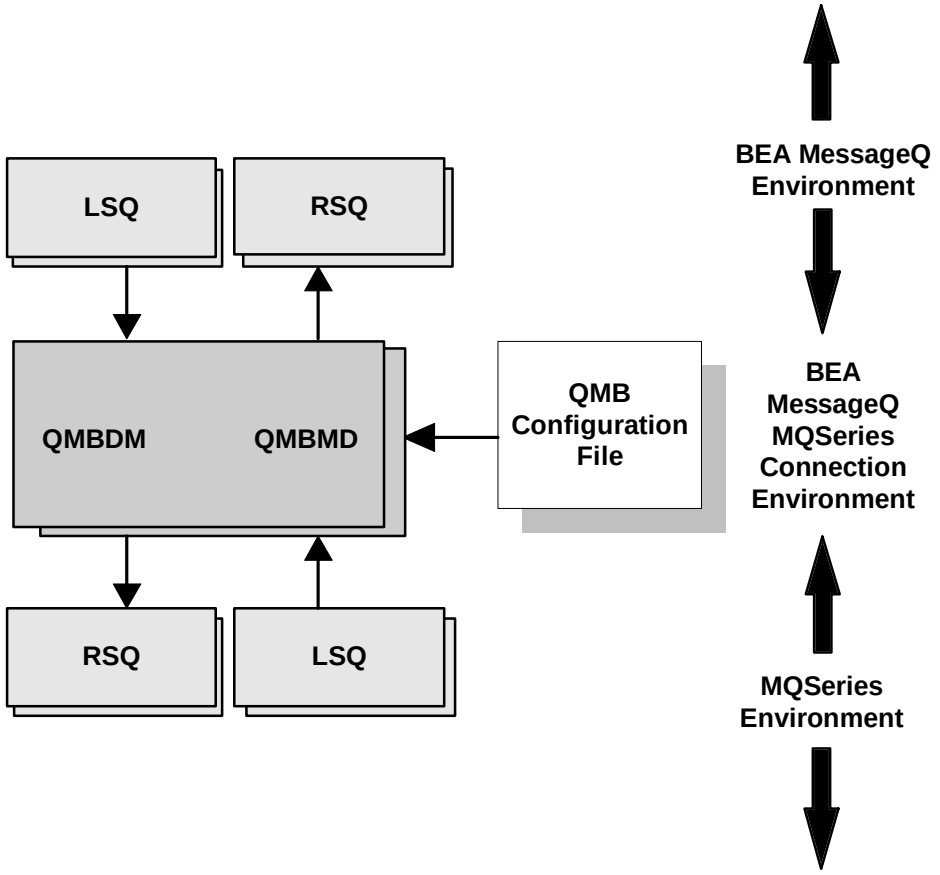
Queue Message Bridge Components

The QMB consists of two processes:

- `QMBDM`—This process is responsible for reading messages from a BEA MessageQ LSQ and forwarding them to the associated IBM MQSeries RSQ.
- `QMBMD`—This process is responsible for reading messages from an IBM MQSeries LSQ and forwarding them to the associated BEA MessageQ RSQ.

Figure 1-3 shows the architecture of the QMB.

Figure 1-3 Queue Message Bridge Architecture



BEA MessageQ to IBM MQSeries messages flow through the QMBDM server and IBM MQSeries to BEA MessageQ messages flow through the QMBMD server.

You may invoke multiple instances of the QMB to allow load balancing and tuning. The actual number of QMB processes required to drive a specific application varies, depending on the application design and overall topology.

Message Flow

A QMB process owns and maintains the LSQ. LSQs can be shared by multiple QMB processes to make BEA MessageQ MQSeries Connection scalable and tunable. An LSQ must exist for each application from which messages will be sent to a remote application. An LSQ must also exist for reply message processing.

Figure 1-4 shows how a QMB forwards a message from a BEA MessageQ LSQ to its associated IBM MQSeries RSQ.

Figure 1-4 Forwarding a Message to MQSeries

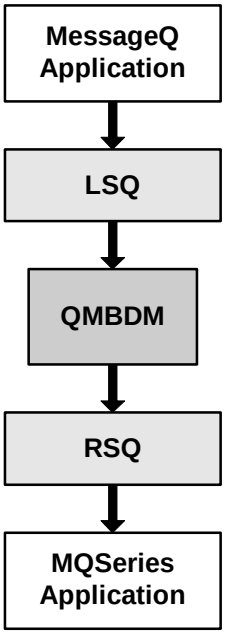
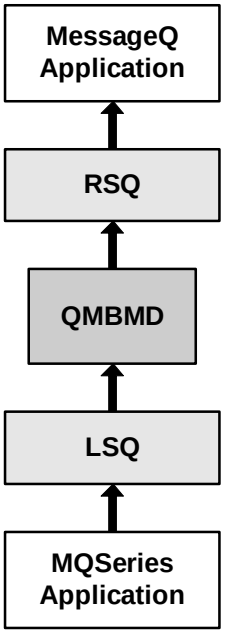


Figure 1-5 shows how a QMB forwards a message from an IBM MQSeries LSQ to its associated BEA MessageQ RSQ.

Figure 1-5 Forwarding a Message to BEA MessageQ



2 Developing Message Queuing Applications

This chapter describes how to develop message queuing applications that use BEA MessageQ MQSeries Connection to exchange messages. It covers the following topics:

- Using Application Programming Interfaces
- Designing Applications to Use BEA MessageQ MQSeries Connection
- Choosing Message Characteristics
- Sending a Request to an IBM MQSeries Server
- Sending a Reply to a BEA MessageQ Client
- Sending a Request to a BEA MessageQ Server
- Sending a Reply to an IBM MQSeries Client
- Restrictions and Limitations

Using Application Programming Interfaces

You should be familiar with the BEA MessageQ and IBM MQSeries application development environments and follow the coding standards required by each. If you are writing the BEA MessageQ half of an application, you must code the BEA

MessageQ application programming interfaces (API) for attaching to the bus, locating target queues, sending and receiving messages, and controlling the application message flow.

Conversely, if you are writing the IBM MQSeries half of the application, you must code to the MQSeries message queuing interface (MQI) for connecting to a Message Queue Manager (MQM), opening and closing queues, sending and receiving messages, and controlling the message flow.

Your application interacts only with the local messaging system (the one to which your application is attached or connected). Forwarding messages to another messaging system is the responsibility of the Queue Message Bridge (QMB) and is not directly controlled by your application.

Designing Applications to Use BEA MessageQ MQSeries Connection

Designing an application that supports the exchange of messages between BEA MessageQ and IBM MQSeries message queueing systems is complex because it requires knowledge of how messages are sent and received by each system. The QMB performs the transformation of the message header data into the format required by the message queuing system running on the receiver platform, but the application developer must understand how the QMB performs the transformation to ensure the correct result. You can use the BEA MessageQ MQSeries Connection to exchange information between both new and legacy applications. Legacy applications may require some alteration unless they use only a simple datagram or request/reply paradigm.

When designing message-driven applications, you must decide how messages are to be sent and received. You may want to design your application so that it sends a request and receives a reply. Or, you may want it to send a datagram message only. To ensure that your application meets your needs, you must identify those needs before starting to design. This section helps you define your requirements by providing information about the following tasks:

- Determining Queues that Your Application Needs

- Defining Queues for BEA MessageQ Clients to IBM MQSeries Servers
- Defining Queues for IBM MQSeries Clients to BEA MessageQ Servers

Determining Queues that Your Application Needs

When designing a message-based application, you must determine what services are needed by your application and how to access these services. Keep in mind that a message-based application accesses a target application by sending a message to a queue that is controlled by the target. This queue must be known and accessible to the application sending the message.

Because the QMB provides communication services to both BEA MessageQ and IBM MQSeries applications, it must present queues to each messaging system in a way that each system recognizes. BEA MessageQ MQSeries Connection does this through the QMB Local Service Queue (LSQ). The receiving application is located on the target messaging system and is listening on a queue known as the Remote Service Queue (RSQ). The QMB manages the LSQ and maintains the LSQ to RSQ relationship.

Before a configured IBM MQSeries or BEA MessageQ queue can be used as a QMB LSQ or RSQ, you must configure it appropriately in the QMB queue configuration file. The entries in this file determine the LSQ to RSQ relationship.

For more information on the QMB configuration file, see the "Configuring the Queue Message Bridge" section in Chapter 3, "Configuring BEA MessageQ MQSeries Connection."

Defining Queues for BEA MessageQ Clients to IBM MQSeries Servers

The QMBDM server reads from the LSQs owned by MessageQ and forwards the received messages to its corresponding RSQ which is a IBM MQSeries queue.

To define queues for applications that send messages from BEA MessageQ clients to IBM MQSeries servers, follow these steps.

1. In the QMB configuration file, define the BEA MessageQ owned LSQ and the IBM MQSeries owned RSQ. In the following example MQS_ECHO is the BEA MessageQ owned LSQ and MQS_ECHO_SERVER is the IBM MQSeries owned RSQ.

   Example:

   ```
   !LSQ       LSQ       RSQ               RSQ
   !Name      Owner     Name              Association
   !
   MQS_ECHO   D         MQS_ECHO_SERVER   S
   ```

2. Define the IBM MQSeries RSQ (MQS_ECHO_SERVER) to the IBM MQSeries Message Queue Manager (MQM) connected to the QMB servers.

   See "Configuring IBM MQSeries" in the next chapter for information on how to define an IBM MQSeries queue.

3. Define the BEA MessageQ LSQ (MQS_ECHO) in the BEA MessageQ group initialization file of the group attached to the QMB. This queue name must be local to the current group and must be a multireader queue.

   (See Listing 3-1 for an example).

For information on defining BEA MessageQ queues, see the BEA MessageQ installation and configuration documentation for your platform.

Defining Queues for IBM MQSeries Clients to BEA MessageQ Servers

The QMBMD server reads from the LSQs owned by IBM MQSeries and forwards the received messages to its corresponding RSQ which is a BEA MessageQ queue.

To define queues for applications that send messages from IBM MQSeries clients to BEA MessageQ servers, follow these steps.

1. In the QMB configuration file, define the IBM MQSeries owned LSQ and the MessageQ owned RSQ. In the following example `DMQ_ECHO` is the IBM MQSeries owned LSQ and `DMQ_ECHO_SERVER` is the MessageQ owned RSQ.

Example:

```
!LSQ       LSQ       RSQ               RSQ
!Name      Owner     Name              Association
!
DMQ_ECHO       M     DMQ_ECHO_SERVER      S
```

2. Define the IBM MQSeries LSQ (`DMQ_ECHO`) to the IBM MQSeries Message Queue Manager (MQM) connected to the QMB servers. This queue must be defined as a QLOCAL shared queue.

   See "Configuring IBM MQSeries" in the next chapter for information on how to define an IBM MQSeries queue.

3. Define the MessageQ RSQ (`DMQ_ECHO_SERVER`) in the MessageQ group initialization file of the group attached to the QMB servers. This queue may be a local queue defined in the %QCT table or a remote queue defined in the %GNT table. Global queue lookups are not supported, therefore, the queue must have a local scope. (See Listing 3-1 for an example.)

4. Define the IBM MQSeries reply queue on which the IBM MQSeries client expects a response. This is not the same reply queue that is used by the QMB servers. This is a reply queue used by the IBM MQSeries client application. This queue needs to be defined to the same MQM to which the IBM MQSeries client application is connected.

For information on defining IBM MQSeries queues, see the *MQSeries Command Reference*.

Choosing Message Characteristics

The characteristics of the message can determine the behavior of the receiving application. Therefore, it is important that when you are designing a message queuing application, you carefully choose the characteristics of the messages that your application will send. The following topics will help you choose the best characteristics for your application's messages:

- Selecting the Type for Message Exchange
- Processing Reply Messages

- Processing Multiple Replies
- Using Message Types and Classes
- Using Recoverable Messaging
- Using Correlation Identifiers
- Using FML Buffers
- Setting Message Priority
- How Message Header Data Is Mapped
- Handling Message Byte Order Differences
- Character Code Conversion
- Guidelines for Choosing Message Characteristics

Selecting the Type for Message Exchange

You can use the QMB to exchange the following types of messages:

- Datagram—When the QMB server receives a datagram message, it forwards the message to the designated RSQ.
- Request—When the QMB server receives a request message, it forwards the request to the designated RSQ. The request is sent with message header information so that the reply can be linked to the originating client. The RSQ program must save and return this message header information in the reply message.
- Reply—When the QMB server receives a reply message, it forwards the reply to the client queue identified in the message header information. See the following section, "Processing Reply Messages," for more information.
- Control—This type of message is a special case message that tells a QMB server to perform a designated internal task such as turning trace logging on/off, opening or closing a log file, and terminating the application.
- Undefined—Messages received on a valid LSQ are checked against known types of messages and processed accordingly. If the type is undefined, the default

characteristics are applied to the message and the QMB forwards the message to the target RSQ. The default characteristics are request (message exchange), no priority, and nonpersistent.

The QMB determines the type of message using the BEA MessageQ message type and class fields or by using the IBM MQSeries `MsgType` header.

Processing Reply Messages

An application program must give special consideration to a request type message exchange. Certain rules must be followed to ensure the successful delivery of the reply to the originating application's reply queue. Applications from which reply messages are going to be sent must adhere to the following rules:

- The reply message must be sent to the sending application's reply queue. Information about the application's reply queue was received with the request and can be determined as follows:
  - Use the `pams_get` source address if the receiving application is written using BEA MessageQ.
  - Use the message descriptors `ReplyToQMgr` and `ReplyToQ` if the receiving application is written using IBM MQSeries.
- Include the Connection Index (CI) that was received with the request in the appropriate message header field of the reply message.

The QMB uses the CI, an internally created entity, to link a reply message to a reply target. This CI is carried in a message header field of the request and returned in the same field of the reply. The data structure used in the message header fields varies, depending on the direction of the request message.

- BEA MessageQ class and type fields are used for message requests flowing from an IBM MQSeries client to a BEA MessageQ server. The class and type fields make up the CI.
- The `ApplIdentityData` field in the IBM MQSeries message descriptor (MQMD) is used for message requests flowing from a BEA MessageQ client to an IBM MQSeries server. The `ApplIdentityData` field contains the ASCII representation of the BEA MessageQ reply address.

For the ApplIdentityData field to pass the CI, first open the IBM MQSeries queue by calling MQOPEN with the MQOO_SET_ALL_CONTEXT option. Then construct all put messages with the MQPMO_SET_ALL_CONTEXT option.

Processing Multiple Replies

BEA MessageQ MQSeries Connection supports multiple replies from a single request. Because of the difference in the way reply messages are processed by BEA MessageQ or IBM MQSeries, different rules apply for sending multiple replies to these two message queueing systems. This section explains:

- How IBM MQSeries Applications Process Multiple Replies
- How BEA MessageQ Applications Process Multiple Replies

How IBM MQSeries Applications Process Multiple Replies

An IBM MQSeries application sending replies formats the data of each reply, includes the ApplIdentityData, and sends the message to the ReplyToQ and ReplyToQMgr descriptors received in the request message. The IBM MQSeries application repeats this for each reply until all replies have been sent.

Listing 2-1 shows a fragment of IBM MQSeries server code that gets messages, processes them, and sends a reply.

Listing 2-1 MQSeries Server Code for Sending a Reply

```
/*********************************************************/
/* MQSeries Server code fragment to Get messages from the */
/* message queue, process the message, and send a reply   */
/*********************************************************/

/***********************************************/
/*   Open the message queue for shared input      */
/***********************************************/

(void)memcpy( (void *)odG.ObjectName,          /* name of input queue   */
              (void *)QName,
              MQ_Q_NAME_LENGTH);
```

```
O_options = MQOO_INPUT_SHARED   /* open queue for shared input    */
          + MQOO_FAIL_IF_QUIESCING,
          + MQOO_SAVE_ALL_CONTEXT;

MQOPEN(Hcon,                    /* connection handle              */
       &odG,                    /* object descriptor for queue    */
       O_options,               /* open options                   */
       &Hobj,                   /* object handle                  */
       &CompCode,               /* MQOPEN completion code         */
       &Reason);                /* reason code                    */

/*************************/
/* stop if it failed      */
/*************************/

if (CompCode != MQCC_OK)
  exit(Reason);

/*************************************************/
/*   Get messages from the message queue         */
/*   Loop until there is a warning or failure    */
/*************************************************/

buflen = sizeof(buffer) - 1;
CompCode = MQCC_OK ;

while (CompCode == MQCC_OK) {
  gmo.Options = MQGMO_ACCEPT_TRUNCATED_MSG
              + MQGMO_CONVERT  /* receive converted messages      */
              + MQGMO_WAIT;    /* wait for new messages           */

  gmo.WaitInterval  = MQWI_UNLIMITED;          /* waiting forever  */
  md.Encoding       = MQENC_NATIVE;
  md.CodedCharSetId = MQCCSI_Q_MGR;

  /*************************************************************/
  /*   In order to read the messages in sequence, MsgId and    */
  /*   CorrelID must have the default value.  MQGET sets them  */
  /*   to the values in for message it returns, so re-initialise */
  /*   them before every call                                  */
  /*************************************************************/

  (void)memcpy((void *)md.MsgId, (void *)MQMI_NONE, sizeof(md.MsgId));
  (void)memcpy((void *)md.CorrelId, (void *)MQCI_NONE,sizeof(md.CorrelId));
  (void)memset((void *)buffer, 0, sizeof(buffer));

  MQGET(Hcon,               /* connection handle                */
        Hobj,               /* object handle                    */
        &md,                /* message descriptor               */
```

```
            &gmo,               /* GET options                        */
            buflen,             /* buffer length                      */
            buffer,             /* message buffer                     */
            &messlen,           /* message length                     */
            &CompCode,          /* completion code                    */
            &Reason);           /* reason code                        */

      /*************************************************/
      /* report reason if any  (loop ends if it failed)   */
      /*************************************************/

      if (Reason != MQRC_NONE)
        (void)printf("MQGET: Report reason code %ld\n", Reason);

      if (CompCode != MQCC_FAILED) {
         buffer[messlen] = '\0';  /* end string ready to use */

         /*****************************************/
         /*    Only process REQUEST messages         */
         /*****************************************/

         if (md.MsgType != MQMT_REQUEST) {
           (void)printf("  -- not a request and discarded\n");
           continue;
         }

         /*****************************************************************/
         /* Set the reply message type                                     */
         /*****************************************************************/

         md.MsgType = MQMT_REPLY;

         /*********************************************************************/
         /* Copy the ReplyToQ and ReplyToQMgr names to the object descriptor */
         /*********************************************************************/

         (void)strncpy(odR.ObjectName, md.ReplyToQ, MQ_Q_NAME_LENGTH);
         (void)strncpy(odR.ObjectQMgrName, md.ReplyToQMgr,MQ_Q_MGR_NAME_LENGTH);

         /*************************************************************/
         /* Insert this processes Qname and Qmgr into the ReplyToQ and */
         /* ReplyToQMgr so receivers can reply back if need be (RTS)   */
         /*************************************************************/

         (void)memset((void *)put_target, 0, MQ_Q_NAME_LENGTH);
         (void)memset((void *)md.ReplyToQMgr, 0, MQ_Q_MGR_NAME_LENGTH);
         (void)memset((void *)md.ReplyToQ, 0, MQ_Q_NAME_LENGTH);

(void)strncpy(put_target, md.ReplyToQ, strlen(put_target));
```

```
        (void)strncpy(md.ReplyToQMgr, QMgrName, MQ_Q_MGR_NAME_LENGTH);
        (void)strncpy(md.ReplyToQ, QName, MQ_Q_NAME_LENGTH);

        /*************************************************************/
        /*   MsgId and CorrelId are currently the values of the      */
        /*   received message.  Reset them if requested, then        */
        /*   stop further reports                                    */
        /*************************************************************/

        if ( !(md.Report & MQRO_PASS_CORREL_ID) ) {
          nLength = strlen((char *)md.MsgId);
          if (nLength > MQ_MSG_ID_LENGTH) nLength = MQ_MSG_ID_LENGTH;
          (void)memcpy((void *)md.CorrelId, (void *)md.MsgId,(size_t)nLength );
          if (nLength < MQ_MSG_ID_LENGTH) md.CorrelId[nLength] = '\0';
}

        if ( !(md.Report & MQRO_PASS_MSG_ID) ) {
          (void)memcpy((void *)md.MsgId, (void *)MQMI_NONE, MQ_MSG_ID_LENGTH);
}

        md.Report   = MQRO_NONE;                /* stop further reports */
        pmo.Options = MQPMO_SET_ALL_CONTEXT +   /* allow pass of ApplIdent */
                      MQPMO_FAIL_IF_QUIESCING;
        pmo.Context = Hobj;


/*************************************************************/
/* Because this code fragment uses the same message descriptor   */
/* for both receiving and sending messages, we do not have to    */
/* save the ApplIdentityData field and copy to an output message */
/* descriptor before we send the reply. The ApplIdentityData    */
/* field carries the MessageQ client address which is the end    */
/* target for the reply data.  If separate input and output      */
/* message descriptors are used, then this information would need*/
/* to be copied from the input md to the output md. */
/*************************************************************/

nSendReplyCount = process_message(buffer);
        while (nSendReplyCount >= 0) {

           nSendReplyCount--;

           MQPUT1(Hcon,           /* connection handle                */
                  &odR,           /* object descriptor                */
                  &md,            /* message descriptor               */
                  &pmo,           /* default options                  */
                  messlen,        /* message length                   */
                  buffer,         /* message buffer                   */
                  &CompCode,      /* completion code                  */
```

```
                &Reason);        /* reason code                    */

          /*************************************************/
          /* report reason if any  (loop ends if it failed) */
          /*************************************************/

          if (CompCode != MQCC_OK)  {
             (void)printf("MQPUT1: Report CompCode code %ld\n", CompCode);
             (void)printf("MQPUT1: Report Reason code %ld\n", Reason);
             if (CompCode == MQCC_FAILED) nSendReplyCount = 0;
          }
       }  /* end while loop */
    }  /* end message for reply */
}  /* end Get message loop */
```

How BEA MessageQ Applications Process Multiple Replies

When a message queuing application responds to a request, the reply may be one in a series of messages, the last in a series, or the only message. A BEA MessageQ application identifies a reply as one of these three types by setting the message header type field in the reply. If the message type field is a positive integer, the reply is the only or last message. If the message type field is negative, the reply is one in a series of messages. All BEA MessageQ replies are mapped to the IBM MQSeries value `MQMT_REPLY`. It then becomes the responsibility of the IBM MQSeries application to handle the first, last, and only replies appropriately.

This classification scheme for replies allows the QMB server to keep the connection active until all replies for a request have been processed. When the last or only message is processed, the CI is removed from the CI table. (To make the message type field negative, the sending process subtracts the value of the message type field from zero (0 - `MSG_TYPE_`*XXX*) and then inserts the remainder into the message type field for the reply message.)

Listing 2-2 shows a code fragment for a BEA MessageQ server to send a reply.

Listing 2-2 BEA MessageQ Server Code for Sending a Reply

```
/*********************************************************/
/* MessageQ Server code fragment to Get messages from the    */
/* RSQ queue, process the message, and send a reply          */
```

```
   /*************************************************************/

   timeout      = 0;      /* wait indefinitely */
   loop         = TRUE;

   while (loop)
   {
      prio = 0;
      /*************************************/
      /* listen for requests and send replys */
      /*************************************/
      nStatus = pams_get_msgw( msg_area,
                               &prio,
                               &source,
                               &class,
                               &type,
                               &dolargemsg,
                               &length_16,
                               &timeout,
                               (int32 *) 0,
                               &lpsb,
                               &show_buf,
                               &show_buf_len,
                               &max_len,
                               &length,
                               (char *) 0 );  /*  Reserved by BEA */

      if (nStatus == PAMS__TIMEOUT)
         continue;                     /* go listen again */


      if ((! (nStatus & 1)) && (nStatus != PAMS__TIMEOUT) )  {
/* log error and exit */
exit(0);
      }

      /*********************************/
      /* Only process REQUEST messages */
      /*********************************/

      if ( (type == MSG_TYPE_DATAGRAM) || (type == MSG_TYPE_RTS_ERROR) )  {
         (void)printf("\n *** Message not a request and discarded\n");
         continue;   /* go read again */
      }

      nSendReplyCount = process_request(msg_area);
      while (nSendReplyCount >= 0) {
         nSendReplyCount--;
```

```
        delivery    = PDEL_MODE_WF_MEM;  /* wait for mem, nonrecover  */
        send_uma    = PDEL_UMA_DISC;     /* If can't deliver it, DISCARD */
        timeout     = 100;               /* Wait 10 seconds */
        prio        = (char) show_buf.priority;

        put_msg_size = length;
        nStatus = pams_put_msg( msg_area,
                &prio,
                &source,  /* passed in */
                &class,   /* foward class on */
                &type,    /* foward type used as index in qmb*/
                &delivery,
                &dolargemsg,
                &timeout,
                (struct psb *) &lpsb,
                &send_uma,
                (q_address *) 0,
                &put_msg_size,
                (char *) 0,
                (char *) 0 );

        if (nStatus != PAMS__SUCCESS)
   nSendReplyCount = 0; /* if put failed then quit */

     } /*end while nSendReplyCount */
  } /* end while loop*/
```

If your BEA MessageQ server application is required to process multiple reply messages, make sure that the CI purge interval is long enough to allow all replies to be sent. To define the CI purge interval, use the `-i` option to `qmbsrv`, the command used to start the QMB. (For more information on running `qmbsrv`, see "Starting the Queue Message Bridge," in Chapter 4, "Managing the BEA MessageQ MQSeries Connection Environment.")

Using Message Types and Classes

Messaging systems use message types to control the logic of a program in a message-driven environment. A **message type** is a specific identifier associated with a message. Types may be assigned by the messaging systems or user application and are contained in the message header portion of the message. Message types are included in all messages sent and received.

In addition to message types, BEA MessageQ provides a message control field called **message class**. In BEA MessageQ message-driven applications, the message class and message type are usually tightly coupled with the program logic. Therefore, when designing and implementing a message-driven interface that has defined message classes and types, you must make sure that all participating programs understand and follow the rules governing message classes and types.

The BEA MessageQ message classes and types defined for use with the QMB allow for maximum flexibility when porting existing BEA MessageQ applications to work with MQSeries Connection. These applications may contain their own message class and type definitions, which must now coexist with the additional QMB definitions.

Whether porting existing applications or writing new ones, you must design the programs to obey the rules defined by the message class and message type values of the QMB, regardless of whether the application is a sending or receiving messages.

The QMB uses individual values and ranges of values (both positive and negative) for message class and message type identifiers.

BEA MessageQ Message Types and Classes

The message dialog between a QMB process and a BEA MessageQ Client or Server is driven by a set of predefined BEA MessageQ class and type identifiers. The values in these message identifiers are determined by the following factors:

- Source of the message (application client, server, or the QMB process)
- State of the dialog (send or receive)
- Type of message exchange (request, reply, datagram, control, undefined)

The class and type fields may be individual values or ranges of values, depending on the combination of the previous factors. Generally, sending applications are required to supply specific class and type values to facilitate the correct message disposition.

Likewise, receiving applications (if sending back a reply) are required to return the received class and type fields with the appropriate reply processing indicator (multi or single element).

See the `qmbuser.h` include file for actual class and type values. The `qmbuser.h` include file is found in */install_dir/*`include` on UNIX systems and in *dev*:*install_dir*\`include` on Windows NT systems. (For BEA MessageQ V5.0, these definitions are also located in another include file, `p_typecl.h`.)

Table 2-1 describes the message classes that are available to a BEA MessageQ client.

Table 2-1 BEA MessageQ Client Message Classes

State	Message Class	Description
Send	`MSG_CLAS_QMB`	Sends a QMB class message
	Other than `MSG_CLAS_QMB`	If your application sends a message with a message class other than `MSG_CLAS_QMB`, the QMB assumes the default characteristics of the request. This scheme allows you to use existing applications without having to change the class.
Receive	`MSG_CLAS_QMB`	Receives a QMB class message

Table 2-2 describes the message classes that are available to a BEA MessageQ server.

Table 2-2 BEA MessageQ Server Message Classes

State	Message Class	Description
Send	QMB range Calculated as an unsigned integer. The range is from 32,768 to 65,535.	Sends a QMB class message. A BEA MessageQ server *must* return the class value received with the request (QMB range) as the reply class.
	`MSG_CLAS_QMB_REPLY_CANCEL`	The server detects that a reply is not forthcoming. This message directs the QMB to release the CI table slot.

Table 2-2 BEA MessageQ Server Message Classes

State	Message Class	Description
Receive	QMB range Calculated as an unsigned integer. The range is from 32,768 to 65,535.	Receives a QMB class message. A BEA MessageQ server must save this value to use in the class field of the reply.
	MSG_CLAS_QMB	Receives a QMB class message

Table 2-3 describes the message types that are available to a BEA MessageQ client.

Table 2-3 BEA MessageQ Client Message Types

State	Message Type	Description
Send	MSG_TYPE_DATAGRAM	Forwards to IBM MQSeries RSQ. No reply.
	MSG_TYPE_REQUEST	Forwards to IBM MQSeries RSQ. Reply pending.
	Other than MSG_TYPE_DATAGRAM or MSG_TYPE_REQUEST	If your application sends a message with a message type other than MSG_TYPE_DATAGRAM or MSG_TYPE_REQUEST, the QMB assumes the default characteristics of the request. This scheme allows you to use existing applications without having to change the message type.
Receive	MSG_TYPE_REPLY	Reply from IBM MQSeries RSQ request
	MSG_TYPE_RTS_ERROR	QMB returns the message. No target RSQ was found.

Table 2-4 describes the message types that are available to a BEA MessageQ server.

Table 2-4 BEA MessageQ Server Message Types

State	Message Type	Description

Table 2-4 BEA MessageQ Server Message Types

Send	Positive range 1 to 1000	Reply to a previously received type message (request). Is an ONLY or LAST message.
	Negative range -1 to -1000	Reply to a previously received type message (request). One in a series of reply message elements.
Receive	Positive range 1 to 1000	Received a message (request) with a reply pending. This field *must* be saved and returned in the type field with the reply.
	`MSG_TYPE_DATAGRAM`	Received message with no reply

IBM MQSeries Message Types

The BEA MessageQ MQSeries Connection supports the IBM MQSeries message types described in Table 2-5. These message types are inserted or received in the MQSeries Message Descriptor (MQMD) by the IBM MQSeries applications. The message types vary, depending on the source and state of the dialog.

Table 2-5 describes the message types that are available to an IBM MQSeries client.

Table 2-5 MQSeries Client Message Types

State	Message Type	Description
Send	`MQMT_DATAGRAM`	Forwards to the BEA MessageQ RSQ. No reply.
	`MQMT_REQUEST`	Forwards to the BEA MessageQ RSQ. Reply pending.
Receive	`MQMT_REPLY`	Reply from BEA MessageQ RSQ from previous request
	`MQMT_RTS`	Return to sender. This is from the QMBMD. No remote service queue or client table slots were available. Implicit error reply.

In Table 2-5, `MQMT_RTS` is a user-defined IBM MQSeries message type. It is defined as follows:

```
# define MQMT_RTSMQMT_APPL_FIRST + 1
```

IBM MQSeries client applications must be prepared to receive this message type as a valid response to a `MQMT_REQUEST`.

Table 2-6 describes the message types that are available to an IBM MQSeries server.

Table 2-6 MQSeries Server Message Types

State	Message Type	Description
Send	`MQMT_REPLY`	Reply to a BEA MessageQ client from a previous request
Receive	`MQMT_DATAGRAM`	Message received. No reply.
	`MQMT_REQUEST`	Request received. Reply to `MQS_REPLYQ` and return `ApplIdentityData` (CI).

Using Recoverable Messaging

To recover messages after a system mishap, your application must have message persistence defined. **Message persistence** describes messages written to nonvolatile storage; persistent messages can survive a system restart.

Messages received with IBM MQSeries Persistence or BEA MessageQ Message Recovery Services are forwarded by the QMB with the corresponding persistence or MRS mode.

BEA MessageQ MQSeries Connection locksteps the IBM MQSeries persistence mode with the BEA MessageQ Message Recovery Services and delays the confirmation of a message until the message has been forwarded and safely stored. Persistence mode processing may affect the performance of your application. You may want to have separate persistence and nonpersistence services.

Note: IBM MQSeries dynamic temporary queues do not support message persistence. BEA MessageQ temporary queues do not support Message Recovery Services.

Using Correlation Identifiers

With BEA MessageQ Version 5.0, you can include an optional 32-byte **correlation identifier** in a message header. The correlation identifier allows a developer to associate a user defined identifier with each message. Applications receiving the message can tag any response to the message with the same identifier. This feature is useful for asynchronous client/server applications because it allows responses to be matched with associated requests. This feature is available only in BEA MessageQ Version 5.0 and BEA MessageQ MQSeries Connection Version 5.0.

IBM MQSeries provides an optional 24-byte correlation identifier in the message header. The BEA MessageQ MQSeries Connection product handles the exchange of correlation identifiers as follows:

- **From IBM MQSeries to BEA MessageQ**—All 24 bytes of the IBM MQSeries correlation identifier are placed in the BEA MessageQ message header. The remaining 8 bytes are padded with space characters.
- **From BEA MessageQ to IBM MQSeries**—Only the first 24 bytes of the BEA MessageQ correlation identifier are placed in the message header.

If messages containing a correlation identifier are going to be exchanged between BEA MessageQ and IBM MQSeries, the correlation identifiers used should have no more than 24 bytes of significant data. Using more than 24 bytes of significant data can lead to unexpected application results or an inability to properly match requests and replies. The QMB Server will truncate correlation IDs that have more than 24 bytes of significant data. If a BEA MessageQ application sends a correlation ID where the last 8 bytes are not either all zeros or blank characters, then a message will be logged to the QMB log file indicating that the correlation ID was truncated.

Using FML Buffers

BEA MessageQ Version 5.0 supports Field Manipulation Language (FML), specifically the 32-bit FML32 format. FML32 enables applications to encode messages with tags and values that describe the content of the message. This capability makes it unnecessary to code the receiver program in such a way that it will recognize the exact data structure of the message. Instead, the receiver program can simply

decode the contents of a message using the tag associated with each value. In addition, FML32 performs data marshaling for applications that include information exchanges between systems that use different hardware data formats.

In some cases, an IBM MQSeries application can be used as a transport between two BEA MessageQ applications. BEA MessageQ MQSeries Connection does not recognize an FML32 buffer and no data marshaling is performed. However, FML32 data is preserved as a binary object and can be used and interpreted as FML32 data on machines with similar architectures.

Setting Message Priority

Message priority is the priority the message assumes for delivery. The BEA MessageQ Version 4.0a product has a priority range of 0-1. For the BEA MessageQ Version 5.0 product, the priority range has been expanded to 0-99. IBM MQSeries Version 5.0 accepts priorities in the range of 0-9.

Message priority is maintained between the BEA MessageQ Version 4.0a and IBM MQSeries Version 5.0 messaging systems as follows:

- Messages received with a BEA MessageQ priority of 0 are sent to IBM MQSeries with MQSeries DefPriority.
- Messages received with an IBM MQSeries priority of 0 are sent to BEA MessageQ with priority 0.
- Messages received with a BEA MessageQ priority of 1 are sent to IBM MQSeries with priority 1.
- Messages received with an IBM MQSeries priority not equal to 0 are sent to BEA MessageQ with a priority of 1.

IBM MQSeries queues defined with a Message Delivery Sequence as first-in/first-out (FIFO) order ignore priority.

BEA MessageQ MQSeries Connection Version 5.0 allows you to map BEA MessageQ Version 5.0 and IBM MQSeries priority ranges using the QMB queue configuration file. You can configure BEA MessageQ MQSeries Connection to use a new default priority mapping or to use the previous 0-1 priority mapping. For more information on mapping priority ranges, see “Configuring the Queue Message Bridge,” in Chapter 3, “Configuring BEA MessageQ MQSeries Connection.”

How Message Header Data Is Mapped

Message header data is defined as the IBM MQSeries message descriptor (MQMD) and the BEA MessageQ message attributes. Many fields used by the two systems contain values with the same meaning but quite different formats. Because of these differences in message header formats, a limited amount of mapping of fields is performed by the QMB applications. To allow application-specific information exchange (if your application requires it), include a user application level header as part of the message body.

In addition, the QMB server maintains (in a limited manner) a loose coupling between a subset of IBM MQSeries and BEA MessageQ message header fields.

Table 2-7 describes this coupling of message header fields.

Table 2-7 Coupling of MQSeries and BEA MessageQ Message Header Fields

MQSeries Message Header Field	BEA MessageQ Message Header Field	Description
`MQMD -> Priority`	`PUT/GET -> Priority`	Message priority
`MQMD -> Persistence`	`PSB -> ConfirmRequest`	Reliable message delivery
`MQMD -> MsgType`	`PUT/GET -> Type`	Message type indicator
`MQMD -> CorrelId`	`SHOWBUFFER->correlation_id`	Correlation identifier
`MQMD -> ApplIdentityData`	`PUT->source q_address`	The BEA MessageQ original source address is stored in the first 10 bytes of the `ApplIdentityData` field in ASCII format: `GGGGG.QQQQ`

Handling Message Byte Order Differences

The content of the message buffer is forwarded as received. It is the responsibility of the receiving application to interpret the byte order and normalize it, if required. The QMB forwards byte order information as follows:

- For messages flowing from BEA MessageQ to IBM MQSeries, the QMB tests the `show_buffer` endian field of the received message and loads the result into

the MQSeries MQMD `ApplOriginData` field as `BEND` or `LEND` text. This field is passed with the message to the IBM MQSeries RSQ.

- For messages flowing from IBM MQSeries to BEA MessageQ, the `show_buffer` endian field for the message received at the BEA MessageQ RSQ contains the endian of the QMB node.

We recommend that you use string data (if possible) or user-defined fields in the message body to carry application-specific information about the message.

Character Code Conversion

Character code conversion between the QMB and MVS based IBM MQSeries applications is supported as follows:

- ASCII-to-EBCDIC data conversion for message flow from the QMB to MVS is supported at the IBM MQSeries channel convert level using the IBM MQSeries built-in-string format `MQFMT_STRING`. If the `CONVERT(YES)` parameter for the channel is coded, all messages flowing over that channel are converted.
- EBCDIC-to-ASCII data conversion for message flow from MVS to the QMB is supported at the IBM MQSeries `MQGET` (`Get Message Option MQGMO_CONVERT`). The IBM MQSeries built-in string format `MQMFT_STRING` must be set by the sending application.
- Custom, user-written data conversion exits are supported at the Local Service Queue level. See the *MQSeries Application Programming Guide* for information on writing user data conversion exits.
- See the *BEA MessageQ MQSeries Connection and MVS Client Release Notes, Version 4.0A and 5.0* for more information on user exits.

Guidelines for Choosing Message Characteristics

When designing message queuing applications, you must choose the characteristics of the messages to be sent by your applications. These characteristics, which indicate to the QMB how the message is to be processed, include the following:

- Message flow

- Message type, which corresponds to the type of message exchange
- Message class

When choosing message characteristics, keep the following guidelines in mind:

- Use the appropriate message class and types (BEA MessageQ and IBM MQSeries) to indicate the kind of message being sent and to ensure that the message is properly processed. Using default message-handling rules may not produce the desired results. (For more information, see "Using Message Types and Classes" in this chapter.)
- When processing request type messages, the receiving application must save and return (in the reply message) the appropriate message header data.
- All messages must contain no more than 4,194,304 bytes.
- A request must be a single message.
- A reply consists of either a single message or multiple reply messages. Each message must contain the appropriate message header information that was received with the original request, and is routed according to that information. The rules for determining the last message in a multireply set must be obeyed.
- Request messages that do not return a reply message may cause processing overhead due to internal routing table maintenance; they should be avoided.

Sending a Request to an IBM MQSeries Server

Suppose you want your BEA MessageQ client to send a request to an IBM MQSeries server and receive a reply. In order for the message to be sent, BEA MessageQ, IBM MQSeries, and the QMB must be properly configured and running.

The following example provides details about the BEA MessageQ client and IBM MQSeries server applications:

- The BEA MessageQ client characteristics are as follows:

```
Name = QMB_DMQCLIENT
rspq = DMQCAdd
Message = Request Data
Class = MSG_CLAS_QMB
Type = MSG_TYPE_REQUEST
Target queue = MQS_ECHO
```

- The IBM MQSeries server characteristics are as follows:

```
Name = QMB_MQSECHO
Message = Request Data
ReplyToQ = MQS_REPLYQ
Type = MQMT_REQUEST
ApplIdentityData = CI
```

Table 2-8 describes the queue definitions required to send a message to an IBM MQSeries server for the previous example.

Table 2-8 Required Queue Definitions for an MQSeries Server

Messaging System	Queue Name/Queue Type	Description
BEA MessageQ	MQS_ECHO / Multireader	MQS_ECHO is the BEA MessageQ LSQ that receives messages to be forwarded to the IBM MQSeries RSQ named MQS_ECHO_SERVER. MQS_ECHO is a multireader queue maintained by a QMBDM process.
IBM MQSeries	MQS_ECHO_SERVER/ Any supported IBM MQSeries queue type	MQS_ECHO_SERVER is the IBM MQSeries queue that the IBM MQSeries server is listening on for requests.
	MQS_REPLYQ / Shared Permanent	MQS_REPLYQ is a required IBM MQSeries queue that receives all IBM MQSeries reply messages from IBM MQSeries server programs. A QMBMD process is listening on this shared queue for replies.

The BEA MessageQ LSQ named MQS_ECHO is associated to the IBM MQSeries RSQ named MQS_ECHO_SERVER in a QMB configuration file. This file stores the LSQ-to-RSQ relationship that the QMB uses when forwarding messages.

Listing 2-3 shows the QMB configuration file for the IBM MQSeries server.

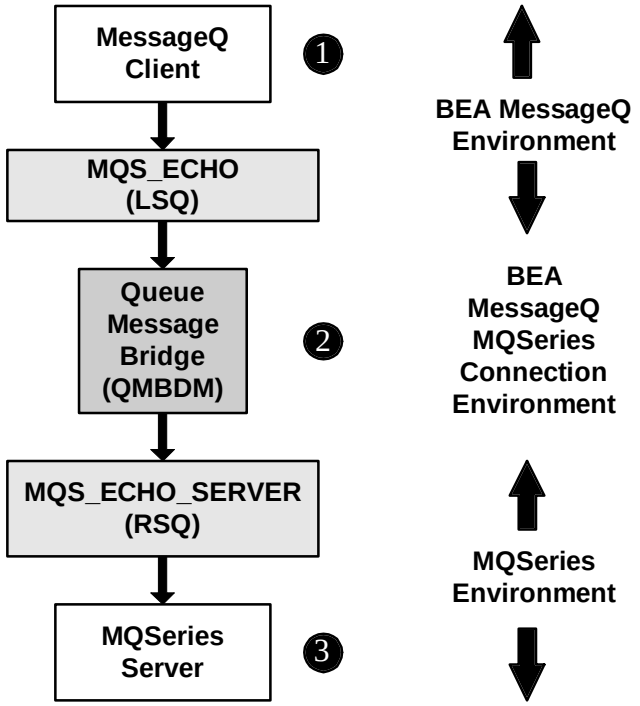
Listing 2-3 MQSeries Server Queue Message Bridge Configuration File

```
!LSQ        LSQ     RSQ                RSQ
!Name       Owner   Name               Association
!
MQS_ECHO     D       MQS_ECHO_SERVER    S
```

For detailed information on fields in the QMB configuration file, see "Configuring the Queue Message Bridge" in Chapter 3, "Configuring BEA MessageQ MQSeries Connection."

Figure 2-1 shows how a request is sent to an IBM MQSeries server.

Figure 2-1 Sending a Request to an MQSeries Server



The sending process shown in Figure 2-1 works as follows:

1. The BEA MessageQ client (named `QMB_DMQCLIENT`) must either know the BEA MessageQ queue address of the LSQ `MQS_ECHO` or use the BEA MessageQ `LocateQ` message-based call to obtain it. Once an address is known, the client sends the request message to the BEA MessageQ LSQ named `MQS_ECHO`.
2. The QMBDM has a read posted against the `MQS_ECHO`. The QMB creates a Connection Index (CI) based on the message source. It inserts the original source `q_address` into the `ApplIdentityData` field and sets the following characteristics:

   - `ReplyToQ = MQS_REPLYQ`
   - `MsgType = MQMT_REQUEST`

   The QMB maps and forwards the message to the IBM MQSeries RSQ named `MQS_ECHO_SERVER`.
3. The IBM MQSeries server (named `QMB_MQSECHO`) reads the message and responds appropriately.

Sending a Reply to a BEA MessageQ Client

Consider the following application that sends a reply from an IBM MQSeries server to a BEA MessageQ client:

- The IBM MQSeries server characteristics are as follows:

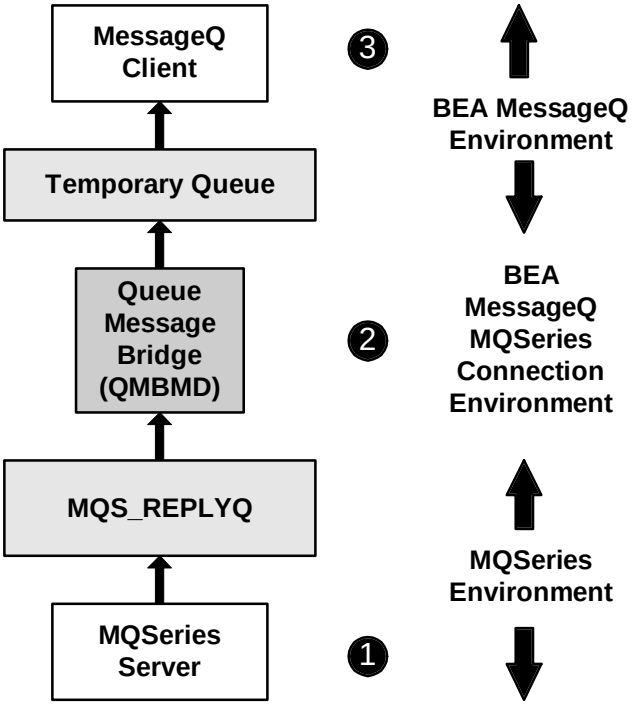
```
Name = QMB_MQSECHO
Message = Reply Data
ReplyToQ = MQS_REPLYQ
Type = MQMT_REPLY
ApplIdentityData = CI
Target = ReplyToQ and ReplyToQMgr
```

- The BEA MessageQ client characteristics are as follows:

```
Name = QMB_DMQCLIENT
Message = Reply Data
Class = MSG_CLAS_QMB
Type = MSG_TYPE_REPLY
```

Figure 2-2 shows how a reply is sent to a BEA MessageQ client.

Figure 2-2 Sending a Reply to a BEA MessageQ Client



The sending process shown in Figure 2-2 works as follows:

1. The IBM MQSeries server (`QMB_MQSECHO`) reads the message, processes it, and sends back a reply (which *must* include the `ApplIdentityData`) to the queue designated by the `ReplyToQ` field `MQS_REPLYQ`, and `ReplyToQMgr`.
2. The `QMBMD` has a read posted against `MQS_REPLYQ`. When the reply message arrives in the `MQS_REPLYQ`, the `QMBMD` program performs the following tasks:
   - Converts the `ApplIdentityData` to the actual BEA MessageQ client target address
   - Maps and forwards the message to the BEA MessageQ client queue named `QMB_DMQCLIENT`
3. The BEA MessageQ client (named `QMB_DMQCLIENT`) reads the message.

When writing message-based applications, use BEA MessageQ and IBM MQSeries system functions and services, where applicable. These functions can help determine the status and state of the bridge, associated application programs, network services, and messaging systems.

Sending a Request to a BEA MessageQ Server

Suppose that you want to send a request to a BEA MessageQ server. In order for this to work, you must have BEA MessageQ, IBM MQSeries, and the QMB properly configured and running.

Consider the following application that sends an IBM MQSeries message to a BEA MessageQ server:

- The IBM MQSeries client characteristics are as follows:

```
Name = QMB_MQSCLIENT
Message = Request Data
ReplyToQ = CLI_REPLYQ
Type = MQMT_REQUEST
Target queue = DMQ_ECHO
```

- The BEA MessageQ server characteristics are as follows:

```
Name = QMB_DMQECHO
Src = DMQ_REPLYQ
Message = Request Data
Class = QMB range
Type = CI
```

Table 2-9 provides an example of the required queue definitions.

Table 2-9 Required Queue Definitions for a BEA MessageQ Server

Messaging System	Queue Name / Queue Type	Description
IBM MQSeries	DMQ_ECHO / Shared permanent	DMQ_ECHO is an IBM MQSeries LSQ that receives messages to be forwarded to the BEA MessageQ RSQ named DMQ_ECHO_SERVER. This is an IBM MQSeries shared queue maintained by a QMBMD process.
	CLI_REPLYQ / Shared permanent	This is the IBM MQSeries queue on which the IBM MQSeries Client listens for replies.
BEA MessageQ	DMQ_ECHO_SERVER / Any supported queue type	This is the BEA MessageQ queue on which the BEA MessageQ server is listening for requests.
	DMQ_REPLYQ / Multireader	DMQ_REPLY is a required BEA MessageQ queue that receives all BEA MessageQ reply messages from the BEA MessageQ server. A QMBDM process listens on this queue for reply messages.

The IBM MQSeries LSQ (named DMQ_ECHO) is associated with the BEA MessageQ RSQ (DMQ_ECHO_SERVER) in a QMB configuration file entry, as shown in Listing 2-4.

Listing 2-4 DMQ_ECHO Queue Message Bridge Configuration File

```
!LSQ        LSQ     RSQ               RSQ
!Name       Owner   Name              Association
!
DMQ_ECHO    M       DMQ_ECHO_SERVER    S
```

For detailed information on fields in the QMB configuration file, see "Configuring the Queue Message Bridge" in Chapter 3, "Configuring BEA MessageQ MQSeries Connection."

Figure 2-3 shows how an IBM MQSeries client sends a request to the BEA MessageQ server.

Figure 2-3 Sending a Request to a BEA MessageQ Server

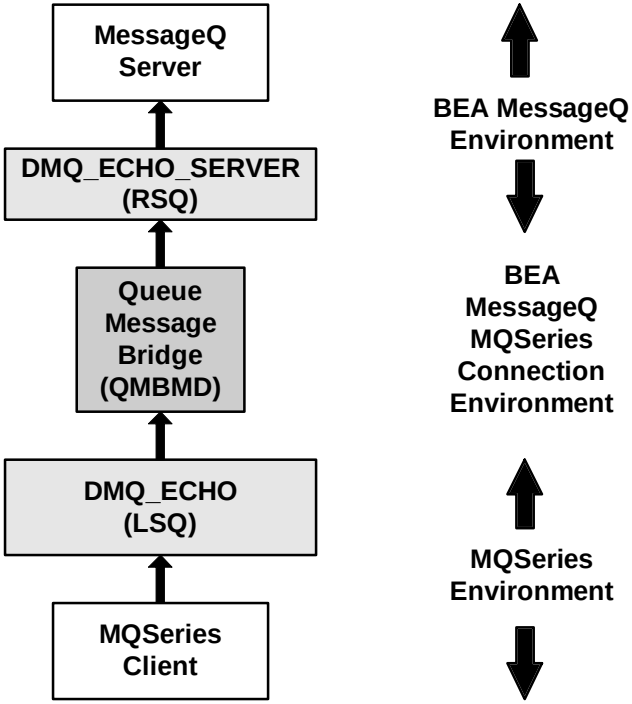


Figure 2-3 shows how an IBM MQSeries client (`QMB_MQSCLIENT`) sends a request message to the IBM MQSeries LSQ named `DMQ_ECHO`. The QMB performs the following functions:

1. Creates a CI based on the `ReplyToQ` name and sets the `respq` field to `DMQ_REPLYQ`.
2. Maps and forwards the message to the BEA MessageQ RSQ named `DMQ_ECHO_SERVER`.

Sending a Reply to an IBM MQSeries Client

Consider the following application that sends a reply from a BEA MessageQ server to an IBM MQSeries client:

- The BEA MessageQ server characteristics are as follows:

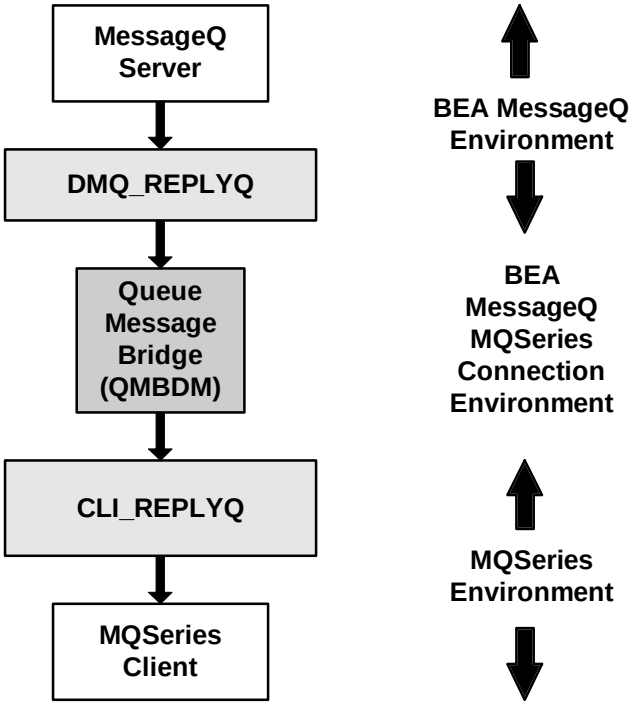
```
Name = QMB_DMQECHO
Src = DMQ_REPLYQ
Message = Reply Data
Class = QMB range
Type = CI
Target queue = DMQ_REPLYQ (same as SRC)
```

- The IBM MQSeries client characteristics are as follows:

```
Name = QMB_MQSCLIENT
Message = Reply Data
ReplyToQ = CLI_REPLYQ
Type = MQMT_REPLY
```

In Figure 2-4, the BEA MessageQ server (`QMB_DMQECHO`) reads the message, processes it, and sends back a reply to the queue (`DMQ_REPLYQ`) designated in the `respq` field. The reply must include the request class and type.

Figure 2-4 Sending a Reply to an MQSeries Client



When the reply message arrives in the BEA MessageQ to IBM MQSeries (`QMBDM`) `DMQ_REPLYQ`, the `QMBDM` program performs the following functions:

1. `QMBDM` extracts the required information from the Connection Index (CI) for the reply message.
2. `QMBDM` maps and forwards the message to the IBM MQSeries client.

Restrictions and Limitations

BEA MessageQ MQSeries Connection has the following restrictions and limitations:

- Each IBM MQSeries message received is processed as a single unit of work.

- The Connection Index (CI) must be returned in the designated message header fields by the service application. For example:
  - For the IBM MQSeries service, the CI is returned in the `MQMD->ApplIdentityData` field.
  - For the BEA MessageQ service, the CI is returned in the message class and type fields
- The `MQMD->ReplyToQ` and `MQMD->ReplyToQMgr` fields must be used by the IBM MQSeries server that received the request as the target when sending back a reply.
- A BEA MessageQ server application must use the source address from the get call as the target queue address to which to send a reply.
- Message header information is not mapped from one system to another. The designated message header fields are exchanged between the QMB and the RSQ application for the purpose of reply processing. For field definitions, see "Processing Reply Messages" earlier in this chapter.
- We recommend that you use multiple instances of the QMB for customizing and scaling your applications. Each instance of the QMB can use its own QMB configuration file or share a common configuration file. A maximum of 49 IBM MQSeries LSQs per `QMBMD` instance is supported. Although there is no specific limit for the number of BEA MessageQ LSQs per `QMBMD` instance, limits imposed by memory resources and by the number of permanent multireader queues allowed by a BEA MessageQ group still apply.
- Performance is affected for messages sent in delivery modes that require waiting for confirmation when the message is delivered to the target.
- The maximum message size is 4,194,304 bytes.
- If messages containing correlation identifiers are going to be exchanged between BEA MessageQ and IBM MQSeries, the correlation identifiers used should have no more than 24 bytes of significant data. Correlation identifiers are supported only by the BEA MessageQ MQSeries Connection Version 5.0 product.
- BEA MessageQ MQSeries Connection does not recognize FML32 buffers; no data marshaling is performed. However, FML32 data is preserved as a binary object and can be used and interpreted as FML32 data on machines with similar architectures. The use of FML32 buffers is supported only by the BEA MessageQ Version 5.0 product.

- The maximum number of IBM MQSeries clients with replies pending is 1000 per QMB set.
- Message chaining (assembly/disassembly) is not supported.
- Because of the loosely coupled aspect of BEA MessageQ to BEA MessageQ MQSeries Connection, most message-based services terminate at the boundary function. For example, using the BEA MessageQ avail/unavail services, a BEA MessageQ client can register interest in the QMB that owned the desired LSQ and be informed of any state changes of that QMB. However, your application is unaware of the status of the RSQ application program itself.
- A properly designed BEA MessageQ server must reply to a request even if there is no application-level response. This allows the QMB to release internal routing resources in a normal and timely manner. Use the BEA MessageQ message class identifier `MSG_CLAS_QMB_REPLY_CANCEL` to cancel a request or to indicate that a reply is not forthcoming.
- The only way you can register an RSQ is by using dynamic queue association from a BEA MessageQ program. You may register both the BEA MessageQ and IBM MQSeries RSQs so that they are dynamically associated at run time.

3 Configuring BEA MessageQ MQSeries Connection

This chapter describes the tasks required to define queues and configure your system to support BEA MessageQ MQSeries Connection, including:

- Overview of Configuration Tasks
- Configuring BEA MessageQ
- Configuring IBM MQSeries
- Configuring the Queue Message Bridge

Overview of Configuration Tasks

Before you configure BEA MessageQ MQSeries Connection, familiarize yourself with the BEA MessageQ and IBM MQSeries applications and queues that your application will access. Specifically, you must know the following:

- Types of messages to be passed (datagram, request, reply, control, undefined)
- Direction of the message flow between applications
- The name of the LSQ, which serves as the intermediate queue for messages

- The name of the RSQ, which serves as the final target queue for messages
- The message queuing system used for each application and queue

After you know these elements of your application, configure each message queuing system according to the instructions provided with the message queuing system, as follows:

- To configure BEA MessageQ, set up the BEA MessageQ group initialization file.
- To configure IBM MQSeries, modify the IBM MQSeries queue definitions with the appropriate LSQ and RSQ information.

After you complete the configuration for each message queuing system, configure BEA MessageQ MQSeries Connection using the QMB configuration file.

The BEA MessageQ MQSeries Connection media kit contains programming examples of an IBM MQSeries-to-BEA MessageQ application. It also contains example configuration and initialization files which support the programming examples. These files are as follows:

- `dmc500_config.dat`—the QMB configuration file in which the LSQ to RSQ relationship is defined.
- `dmc500_dmq_group.init`—group initialization file that contains all the QCT entries needed to execute the QMB and programming examples.

The programming examples and supporting files are located in:

- */install_dir/*`examples/mqsc` on UNIX systems
- *dev*`:\`*install_dir*`\examples\mqsc` on Windows NT systems

Use programming examples and supporting files to help you understand how to configure the BEA MessageQ MQSeries Connection software. For more information on the programming examples, see Appendix A, “Programming Examples.”

Configuring BEA MessageQ

For each BEA MessageQ message queuing group that communicates with an IBM MQSeries application, you must set up a BEA MessageQ group initialization file (`group.init`).

The BEA MessageQ group initialization file that supports a QMB set must include the appropriate queue definitions required for the QMB servers and all available BEA MessageQ local services.

A QMB set requires the following queue entries in the QCT section of the group initialization file:

- A permanent primary queue named `QMBDM`. This is the queue to which the BEA MessageQ to IBM MQSeries QMB server (`QMBDM`) attaches.
- A permanent primary queue named `QMBMD`. This is the queue to which the IBM MQSeries to BEA MessageQ QMB server (`QMBMD`) attaches.
- A permanent multireader queue. This is the BEA MessageQ reply queue. The name of this queue must be `DMQ_`*r*, where *r* is the value passed in the `-r` *qname* parameter of the QMB start command. For example, if you specify `-r REPLYQ`, then the actual BEA MessageQ queue name is `DMQ_REPLYQ`. For more information on the `-r` parameter, see "Starting the Queue Message Bridge" in Chapter 4, "Managing the BEA MessageQ MQSeries Connection Environment."

The group initialization file must also include a QCT entry for each LSQ "owned" by BEA MessageQ. All LSQ queues must be defined as permanent multireader queues.

Listing 3-1 provides an example of a BEA MessageQ initialization file queue entry section. In this example, the `QMBDM`, `QMBMD`, and `DMQ_REPLYQ` are required queues for the QMB server. The queues `LSQ_TEST` and `MQS_ECHO` are LSQs and `DMQ_ECHO_SERVER` is an RSQ.

Listing 3-1 BEA MessageQ Configuration File Queue Entry Section

```
!QUEUE CONFIGURATION SECTION
!
% QCT

!Queue          Queue  Byte   Msg   Quota  UCB  Queue Owner Conf  Perm   Name Security
```

```
!Name           Number Quota  Quota Enable Send Type  Queue Style Active Scope
!
QMBDM             8    64000  100    None   .    P     0     .     N      L    N
QMBMD             9    64000  100    None   .    P     0     .     N      L    N
DMQ_DMC_REPLYQ    7    64000  100    None   .    M     0     .     Y      L    N
LSQ_TEST         11    32000   10    None   .    M     0     .     Y      L    N
MQS_ECHO         12    64000  100    None   .    M     0     .     Y      L    N
DMQ_ECHO_SERVER  15    64000  100    None   .    M     0     .     N      L    N
```

Note that the actual values of `Queue Number`, `Byte Quota`, and `Msg Quota` for each entry are determined by other existing queue configuration entries, message sizes, and arrival rate calculations for the QMB application.

If your application requires multiple QMB servers, the additional servers attach to BEA MessageQ Temporary Primary Queues (TPQs) and associate themselves with the appropriate permanent QMB server. No BEA MessageQ configuration entries are required when using TPQs with additional instances of a QMB server.

See the BEA MessageQ installation and configuration documentation for your platform for more information on configuring the BEA MessageQ product.

Group Name Table

Any BEA MessageQ service that is not local to the group with the QMB must have an entry in the group name table. This table allows BEA MessageQ MQSeries Connection to locate the actual queue address when it processes the RSQ entries in the QMB configuration file.

Note: The QMB supports only local name scope lookups. Global name scope lookups are not supported.

For example, suppose `DMQ_ECHO_SERVER`, as defined in the QMB configuration file, does not reside in the BEA MessageQ group defined previously. In this case, BEA MessageQ MQSeries Connection requires a group name table entry to allow address resolution. If the `DMQ_ECHO_SERVER` program is on a remote group such as group 5/queue 24, the entry is as follows:

```
%GNT

!Name              Address      Scope
```

```
DMQ_ECHO_SERVER     5.24            L
```

The appropriate BEA MessageQ cross-group configuration entries must be in place to allow access to remote services, which are offered in other BEA MessageQ groups.

See the BEA MessageQ installation and configuration documentation for your platform for more information on configuring the BEA MessageQ product.

Configuring IBM MQSeries

This topic describes how to configure IBM MQSeries to use BEA MessageQ MQSeries Connection. It includes the following tasks:

- Configuring the Required IBM MQSeries Queues
- Defining IBM MQSeries Queues

Configuring the Required IBM MQSeries Queues

A QMB set requires the configuration of an IBM MQSeries permanent shared queue for an IBM MQSeries reply queue and all offered LSQs. These queues must be configured in the Message Queue Manager (MQM) to which the QMB connects. They must have the same characteristics as the IBM MQSeries queue model defined in Listing 3-2.

The reply queue name must be `MQS_`*r*, where *r* is the value passed in the `-r` parameter of the QMB startup command line. The QMB prefixes the constant `MQS_` to the `-r` parameter for a complete name. For more information on the `-r` parameter, see "Starting the Queue Message Bridge" in Chapter 4, "Managing the BEA MessageQ MQSeries Connection Environment."

Each LSQ that the QMB services must also be defined in the MQM.

Listing 3-2 shows a sample definition of the `QMODEL` for an IBM MQSeries LSQ. The source for this model is located in */install_dir/*`templates` on UNIX systems and in *dev*:*install_dir*\`templates\mqsc` on Windows NT systems. This model may be copied to the MQM configuration file or added manually. To add it manually, use the `define` and `alter` commands from the `runmqsc` IBM MQSeries utility.

Listing 3-2 MQSeries Queue Definition

```
DEFINE QMODEL('QMBLSQ')                              +
       DESCR('QMB Local Service Queue Model')        +
       PUT(ENABLED)                                  +
       DEFPRTY(0)                                    +
       DEFPSIST(NO)                                  +
       GET(ENABLED)                                  +
       MAXDEPTH(5000)                                +
       MAXMSGL(4194304)                  +
       SHARE                             +
       DEFSOPT(SHARED)                   +
       MSGDLVSQ(PRIORITY)                +
       USAGE(NORMAL)                     +
       NOTRIGGER                         +
       RETINTVL(999999999)               +
       BOTHRESH(0)                       +
       BONAME(' ')                       +
       SCOPE(QMGR)                       +
       QDEPTHHI(80)                      +
       QDEPTHLO(20)                      +
       QDPMAXEV(ENABLED)                 +
       QDPHIEV(DISABLED)                 +
       QDPLOEV(DISABLED)                 +
       QSVCINT(999999999)                +
       QSVCIEV(NONE)
```

Note: The plus sign in Listing 3-2 is an IBM line continuation character.

Defining IBM MQSeries Queues

You can modify the Message Queue Manager configuration by either executing the `runmqsc` utility interactively or editing and loading the command file associated with the MQM. See *MQSeries Command Reference* and *MQSeries System Administration* for more information.

You can add the `QMODEL` defined in Listing 3-2 to the command file, and then use the `QMODEL` in defining the QMB required queues with the `runmqsc` utility.

- To define the IBM MQSeries reply queue, enter the following command:

  ```
  DEFINE QLOCAL(MQS_REPLYQ) LIKE(QMBLSQ)
  ```

- To define an IBM MQSeries LSQ for the echo program, enter the following command:

  ```
  DEFINE QLOCAL(DMQ_ECHO) LIKE(QMBLSQ)
  ```

Any remote service offered on the local MQM must be defined as `QLOCAL`. Any remote service that is not on the local MQM must be defined as `QREMOTE` to allow the messages be forwarded to it. Also, all Distributed Queue Manager (DQM) definitions (channels, `xmitq`, and others) must exist and be active to allow IBM MQSeries distributed processing to occur.

A template file defining all the IBM MQSeries queues required to run the programming examples is located in */install_dir/*`templates/dmc500_mqmdef.cfg` on UNIX systems and in *dev*:*install_dir*`\templates\dmc500_mqmdef.cfg` on Windows NT systems.

To load the MQM with the required commands, so that the example QMB queue definitions are included, enter the following command:

```
runmqsc YOURMQMNAME < dmc500_mqmdef.cfg > mqmdef.out
```

The `mqmdef.out` file contains the results of the command executions.

Tips for Configuring IBM MQSeries

Note the following tips when configuring an IBM MQSeries queue:

- An IBM MQSeries transmit queue must have a depth greater than the maximum message batch that can ever be received. If it does not, the queue reaches `MAXDEPTH` and the channel shuts down.
- IBM MQSeries queues are created in the following directories:

  `/var/mqm/qmgrs/QNAME/queues` on UNIX systems

  `dev``:\mqm\qmgrs\QNAME\QUEUES` on Windows NT systems

  A queue can grow to the maximum size of (`MAXDEPTH *` *`largest_message_received`*). IBM MQSeries requires that the file system that it is installed on be able to support the expected message volume.
- An IBM MQSeries channel must be started manually or triggered by placing a message in the channels transmit queue (assuming the channel is configured to allow triggering). If a channel is triggered, the channel initialization program must be active and properly configured.

See the *MQSeries Command Reference* for detailed information on how to configure IBM MQSeries software.

Configuring the Queue Message Bridge

Use the QMB configuration file to define the services offered by one messaging system to the other. The QMB processes read the configuration file and set up an association between an LSQ and an RSQ. This association can be established statically or dynamically. An LSQ is a queue that is local to the messaging system of the sending application. An RSQ is a queue that is remote to the messaging system of the sending application and is the final target of all messages received on the associated LSQ. Configuration files for MQSeries Connection Version 5.0 also include priority mapping information.

The QMB configuration file is an ASCII text file that may be located in any directory. Parameters within the file are separated by white space. An example configuration file, named `dmc500_config.dat`, is located in */`*`install_dir`*`/templates` on UNIX systems and in *`dev`*`:\`*`install_dir`*`\templates` on Windows NT systems.

A configuration file for MQSeries Connection Version 4.0A includes only the parameters required to associate an LSQ to an RSQ. A configuration file for MQSeries Connection 5.0 includes a section containing the parameters required to associate an LSQ to an RSQ (`%QUEUE`) and a section containing the parameters for defining priority mapping (`%PRIORITY MAPPING`). The version of the configuration file is shown in the `%VERSION` section at the head of the file.

BEA MessageQ MQSeries Connection Versions 4.0A and 5.0 both support the same QMB configuration file format, except that the Version 4.0A product ignores the `%PRIORITY MAPPING` section. In addition, both versions support the configuration file format used in BEA MessageQ MQSeries Connection Version 3.2B.

A QMB configuration file includes the following parameters: the LSQ Name, LSQ Owner, RSQ Name, RSQ Association, and Format Name. These parameters must be specified on the same line in the order in which they are listed in Table 3-1. Configuration files for MQSeries Connection Version 5.0 contain an additional section in which priority mapping is defined. Table 3-1 describes all the QMB configuration file parameters.

Table 3-1 Parameters in the Queue Message Bridge Configuration File

Field	Description
LSQ Name	Name of the Local Service Queue defined in either BEA MessageQ or IBM MQSeries. This queue receives messages to be forwarded to the associated RSQ. BEA MessageQ LSQs must be multireader queues in the group to which the QMB attaches. IBM MQSeries LSQs must be shared queues that are defined in the MQM. The QMB connects to the queue specified by this parameter.
LSQ Owner	Indicates the messaging system that owns the LSQ as follows: ♦ A `D` indicates BEA MessageQ. It is used for messages flowing from BEA MessageQ to IBM MQSeries. ♦ An `M` indicates IBM MQSeries. It is used for messages flowing from IBM MQSeries to BEA MessageQ.

Table 3-1 Parameters in the Queue Message Bridge Configuration File

Field	Description
RSQ Name	Name of the Remote Service Queue associated with the LSQ. The RSQ is the target queue when a message is sent. If a name is specified, the target queue is static. If a period (.) is specified, it serves as a placeholder for dynamic queue association. This is the final target for all messages sent. The RSQ may be located anywhere in the target messaging system that is accessible to the local BEA MessageQ group or IBM MQSeries MQM. A BEA MessageQ RSQ requires an entry in the Group Name Table if it is not defined in the local group. An IBM MQSeries RSQ requires a `QREMOTE` definition if it is not defined in the local MQM.
RSQ Association	Method of Remote Service Queue registration as follows: ♦ An `S` indicates static association of the RSQ name to the LSQ. ♦ A `D` indicates dynamic queue association of the RSQ name or address received in a registration request to the LSQ. See "Registering Remote Service Queues" later in this chapter for more information on static and dynamic queue association.
Format Name	If specified, this name becomes the message descriptor Format parameter. If the IBM MQSeries channel definition associated with the RSQ allows character conversion, the named conversion exit will be invoked.
MessageQ Priority Range	The BEA MessageQ priority range. Each segment of the MessageQ priority range (for example 0-9, 10-19, and so on) must be mapped to a single IBM MQSeries priority and must be listed in ascending order. The priority range segments must be contiguous and all MessageQ priorities (0-99) must be included in the list of ranges. Priority values must not overlap. **Note:** This field is recognized only by BEA MessageQ MQSeries Connection Version 5.0 and is ignored by the Version 4.0A product.
IBM MQSeries Priority	The IBM MQSeries priority value in the range of 0-9. All ten priority values must be listed in ascending sequential order and may be listed no more than once. **Note:** This field is recognized only by BEA MessageQ MQSeries Connection Version 5.0 and is ignored by the Version 4.0A product.

Table 3-1 Parameters in the Queue Message Bridge Configuration File

Field	Description
MessageQ Priority	The BEA MessageQ priority value to which an IBM MQSeries priority value is mapped. **Note:** This field is recognized only by BEA MessageQ MQSeries Connection Version 5.0 and is ignored by the Version 4.0A product.

A QMB server process reads the configuration file passed with the command line, resolves all name-to-address issues, and builds an internal table to allow LSQ to RSQ mapping.

In the following example, BEA MessageQ clients have access to IBM MQSeries servers through the following BEA MessageQ LSQs:

- `MQS_ECHO` (uses static association)
- `MQSTEST` (uses dynamic queue association)

Likewise, IBM MQSeries clients have access to BEA MessageQ servers through the following IBM MQSeries LSQs:

- `DMQ_ECHO` (uses static queue association)
- `DMQTEST` (uses dynamic queue association)

Listing 3-3 shows the QMB configuration file that supports this case.

Listing 3-3 Queues Defined in a QMB Configuration File

```
! QMB Config File to Support ECHO Servers
!
!LSQ       LSQ     RSQ              RSQ
!Name      Owner   Name             Association
!
MQS_ECHO   D       MQS_ECHO_SERVER  S
MQSTEST    D       .                D
!
DMQ_ECHO   M       DMQ_ECHO_SERVER  S
DMQTEST    M       .                D
```

An LSQ with a dynamic RSQ association must have a program registered to it before it can be used. Any messages received before registration are returned to the sender with the appropriate Return To Sender (RTS) message type.

Reply queues are built using the `-r` parameter on the `qmbsrv` command line and are not required to be defined in the QMB configuration file.

When a QMB server process reads an MQSeries Connection Version 5.0 configuration file, it also sets up an internal table to allow priority mapping. Listing 3-4 shows a sample priority mapping section:

Listing 3-4 Priority Mapping Defined in a QMB Configuration File

```
* MessageQ Priority Range   MQSeries Priority   MessageQ Priority
*-------------------------------------------------------------
          0-09                      0                    0
         10-19                      1                   11
         20-29                      2                   22
         30-39                      3                   33
         40-49                      4                   44
         50-59                      5                   55
         60-69                      6                   66
         70-79                      7                   77
         80-89                      8                   88
         90-99                      9                   99
*

%EOS
```

Given a BEA MessageQ priority range of 30-39, an IBM MQSeries priority of 3, and a BEA MessageQ priority of 33, the following priority conversions occur:

- A priority of 3 in an IBM MQSeries message is converted to a BEA MessageQ priority of 33.
- A priority between 30 and 39 (inclusive) in a BEA MessageQ message is converted to an IBM MQSeries priority of 3.

Registering Remote Service Queues

An RSQ must be registered to an LSQ before any message may be forwarded to it. The RSQ registration may be statically defined in the QMB configuration file or dynamically associated at run time. There are two things that you must do to register an RSQ dynamically:

- You must build and fill in a registration message data structure with the information needed to allow the association of the dynamic RSQ to the LSQ. The same structure is used to register BEA MessageQ or IBM MQSeries dynamic RSQs. If registering a BEA MessageQ RSQ, your application must supply the address of the RSQ. If registering an IBM MQSeries RSQ, your application must supply the name of the RSQ.
- You must set the message type and class for RSQ registration. Code your application to send a specially formatted message with the BEA MessageQ message type of `MSG_TYPE_RSQ_REGISTER` and message class of `MSG_CLAS_QMB` to the permanent `QMBDM` primary queue (PQ). The `QMBDM` processes and forwards the registration request to all other associated QMB servers in the set.

In the QMB configuration file, static or dynamic association of an RSQ to an LSQ is determined by the RSQ association parameter of the LSQ entry. The RSQ association parameters are as follows:

- [S] Static—The QMB uses the queue name defined in the RSQ name parameter as the RSQ to be associated with the LSQ. Static associations are determined and created at QMB initialization.
- [D] Dynamic—The QMB uses the queue name or BEA MessageQ address received in an RSQ registration control message as the RSQ to be associated with the LSQ.

You can also register an RSQ using the BEA MessageQ MQSeries Connection utility. For more information on this utility, see "Using the BEA MessageQ MQSeries Connection Utility" in Chapter 3, "Configuring BEA MessageQ MQSeries Connection."

All messages received on the defined LSQ are forwarded to the RSQ name or address. This scheme allows flexibility: the server application can be moved among various BEA MessageQ groups or IBM MQSeries MQMs. Messages received on an LSQ that does not have a registered RSQ are returned to the sender.

Listing 3-5 shows a registration message data structure.

Listing 3-5 Registration Message Data Structure

```
typedef struct
  {
   char lsq[49];        /* (MAX_MQS_LEN = 49)              */
   char lowner;         /* LSQ Owner [D]MQ or [M]QS        */
   char rsq[49];        /* RSQ name of MQS Appl Queue      */
   char rfu;            /* Reserved for Future Use         */
   q_address rsq_add;   /* DMQ Address of DMQ RSQ          */
  }
   rsq_reg_struct;
```

Data structures and BEA MessageQ class and type definitions are supplied in the `qmbuser.h` file, which is located in */install_dir/*`include` on UNIX systems and in *dev*:*install_dir*\`include` on Windows NT systems. For BEA MessageQ Version 5.0, class and type definitions are also supplied in the `p_typecl.h` include file.

CHAPTER

4 Managing the BEA MessageQ MQSeries Connection Environment

This chapter describes how to manage the BEA MessageQ MQSeries Connection environment. It covers the following tasks:

- Starting the Queue Message Bridge
- Stopping the Queue Message Bridge
- Using the BEA MessageQ Monitor Utility
- Using the runmqsc MQSeries Utility
- Troubleshooting BEA MessageQ MQSeries Connection Problems
- Using IBM MQSeries Log Files
- Using the BEA MessageQ MQSeries Connection Utility

Starting the Queue Message Bridge

Note: Before you can start the QMB, you must set the BEA MessageQ environment variables for the BEA MessageQ bus and group to which the QMB will attach.

To start the QMB processes, you must run the `qmbsrv` command with the required command parameters. You supply the required parameters in a command file or on the command line during an interactive session. Remember that for the bridge to function correctly, you must have at least one QMBDM and one QMBMD.

The command syntax for the QMB program is as follows:

```
qmbsrv -d msg-direction -s server-type -n mgr-name -r reply-qname
     -c pathname -l pathname -a number -i interval
     [-e] [-t] [-p] [-v]
```

Table 4-1 describes the QMB command parameters.

Table 4-1 Queue Message Bridge Command Parameters

Parameter	Description
`-d msg-direction`	Specifies the direction in which messages flow. You must specify one of the following parameters: ♦ `QMBDM` for messages flowing from BEA MessageQ to IBM MQSeries ♦ `QMBMD` for messages flowing from IBM MQSeries to BEA MessageQ This parameter is passed as an ASCII string.
`-s server-type`	Defines the QMB server type, which can be either `PERMANENT` or `TEMPORARY`. However, there must be at least one `PERMANENT QMBDM` and `QMBMD` in a QMB server set.
`-n mgr-name`	Specifies the name of the IBM MQSeries queue manager to which the QMB connects. The *`mgr-name`* specified must be the same for all QMB instances in the set. This parameter is passed as an ASCII string.

Table 4-1 Queue Message Bridge Command Parameters

Parameter	Description
`-r` *`reply-name`*	Specifies the name of the reply queue. The QMB prefixes the name with either `DMQ_` or `MQS_` to create a full name. This queue must be defined as a permanent queue on the messaging system. The *`reply_name`* must be the same for all QMB instances in the set. This parameter is passed as an ASCII string.
`-c` *`pathname`*	Specifies the full pathname of the QMB configuration file
`-l` *`pathname`*	Specifies the full pathname of the QMB log file
`-a` *`number`*	Specifies the maximum number of active (reply pending from BEA MessageQ servers) IBM MQSeries clients. The *`number`* specified must be the same for all QMB instances in the set. The maximum value is 1000.
`-i` *`interval`*	Specifies the connection index (CI) purge interval (in seconds) for pending replies being sent from a BEA MessageQ application to an IBM MQSeries client. The *`interval`* specified must be the same for all QMB instances in the set.
`-e`	Specifies event logging. This parameter is optional.
`-t`	Specifies trace logging. This parameter is optional.
`-p`	Enables default priority mapping if a valid `%PRIORITY MAPPING` section is not found in the QMB configuration file. This parameter is optional and is ignored on BEA MessageQ MQSeries Connection Version 4.0A.
`-v`	Enables Verbose trace logging. This parameter is optional and should only be used for troubleshooting when the `-t` parameter is not sufficient.

For MQSeries Connection Versions 4.0A and 5.0, the `-m` *`memory-name`* parameter is obsolete. In earlier versions, this parameter specified the shared memory name assigned to the shared memory and semaphore required by the QMB. The name had either `_SEM` or `_SHM` appended to it. For the current version, the parameter is ignored and no error is generated if the parameter is used.

Note that the -e, -t, and -v parameters generate a fair amount of information in your log files. These parameters should not be left on for extended periods.

To start the permanent QMB process, enter the following commands:

```
qmbsrv -d QMBDM -s PERMANENT -n MQMNAME -r DMC_REPLYQ
    -c /usr/qmb/qmbprod1.cfg -l /usr/qmb/qmbdm.log -a 60 -i 100

qmbsrv -d QMBMD -s PERMANENT -n MQMNAME -r DMC_REPLYQ
    -c /usr/qmb/qmbprod1.cfg -l /usr/qmb/qmbmd.log -a 60 -i 100
```

Note: Permanent QMBDM and QMBMD processes must be running before any message exchange can occur. These processes register interest in each other and allow message exchange only if *both* are active. For QMB processes to register interest in each other, the BEA MessageQ SBS Server must be enabled so that AVAIL Services are active.

To add a QMBMD server (in which messages flow from an IBM MQSeries application to a BEA MessageQ application), enter the following command:

```
qmbsrv -d QMBMD -s TEMPORARY -n MQMNAME -r DMC_REPLYQ
    -c /usr/qmb/qmbprod2.cfg -l /usr/qmb/qmbmd.log -a 60 -i 100
```

You can use a different configuration file (qmbprod2.cfg) to allow specific queues to be serviced by this copy of the QMB. Using a different configuration file gives you an additional level of customization and tuning.

Performance Considerations

You can maximize the performance of the IBM MQSeries to BEA MessageQ connection. To do so, add a temporary QMBMD process with a configuration file designed to service a specific IBM MQSeries LSQ. The required permanent QMBMD process polls (in a round-robin fashion) all IBM MQSeries LSQs for input. You can force a temporary QMBMD process to service a subset or single LSQ or REPLYQ by specifying a tailored configuration file with the -c parameter at startup. The rules that determine how a temporary QMBMD process services its LSQs are as follows:

- If the QMB configuration file does not contain any LSQ entries, *only* the REPLYQ (specified with the -r parameter) is serviced.
- If the QMB configuration file contains a single LSQ entry, *only* that LSQ is serviced.

- If the QMB configuration file contains more than one LSQ entry, the round-robin polling algorithm for all LSQs and the `REPLYQ` is invoked.

Stopping the Queue Message Bridge

You can stop QMB processes in the following ways:

- Send a terminate message type to the primary `QMBDM` process. This message type terminates all the QMB processes in a set. See "Using the BEA MessageQ MQSeries Connection Utility" in this chapter for more information.
- On UNIX systems, use the `kill` command. The `kill` command terminates processes one at a time providing the *`user_ID`* has the appropriate authority.

Using the BEA MessageQ Monitor Utility

To troubleshoot BEA MessageQ performance problems, use the BEA MessageQ Monitor utility. You can use this utility to determine traffic counts, the status of queues, cross-group connections, and attached programs. You can also use it to determine whether messages are flowing correctly by monitoring detailed information about the queues involved. For more information on the BEA MessageQ Monitor utility, see the BEA MessageQ installation and configuration guide for your system.

Using the runmqsc MQSeries Utility

Use the `runmqsc` MQSeries utility to determine the state and status of the MQSeries components used by the QMB. You can use the `runmqsc` utility as follows:

- Define and alter queues
- Define and alter channels

- Display queue definition and status information
- Display channel definitions and status information
- Start and stop channels

See the *MQSeries Command Reference* for a complete list of valid commands.

Troubleshooting BEA MessageQ MQSeries Connection Problems

To troubleshoot problems with BEA MessageQ MQSeries Connection, view the log files produced by BEA MessageQ, the QMB, and IBM MQSeries applications. Typically, these log files contain important information about successful and unsuccessful system events.

Note that the most efficient way to troubleshoot message queuing applications is to follow the queues used by the application. Start by making sure you are familiar with the flow of messages in your application as the message goes from queue to queue. Use the tools available on each platform to determine and follow the message flow. The most common approach is to use message counts and queue depths to follow a message.

Queue Message Bridge Log Files

You can view the QMB log files to troubleshoot your messaging application. The QMB produces log files that provide the following information:

- QMB program initialization parameter settings
- Any errors detected when sending or receiving messages
- Startup parameters

The QMB program logs events and errors to the log defined at startup with the `-l` option to the `qmbsrv` command. To request a more detailed log file, there are additional command parameters that you can use:

- The `-e` parameter requests program event logging (all gets, puts, and so on).
- The `-t` parameter requests message trace logging.
- The `-v` parameter requests verbose trace logging. This is only recommended for troubleshooting purposes when the `-t` parameter does not provide enough information.

Logging generates a fair amount of information; do not leave it on for extended periods of time. Event logging is generally used to follow message flow. Trace logging is used for problem determination.

Listing 4-1 shows a `QMBDM` log file created with the `-e` parameter.

Listing 4-1 QMBDM Log File

```
QMB: Dec 16 16:48:34 Version: 5.0-00 for HP-UX

QMB: Dec 16 16:48:34 MQS Client Maximum Active : 100
QMB: Dec 16 16:48:34 MQS Client Inactivity Timer: 300
QMB: Dec 16 16:48:34 Event Logging Enabled
MQS: Dec 16 16:48:35 Connected to Queue Manager QMGR1, Hconn: 1073800288
SEM: Dec 16 16:48:35 Create: /var/tmp/dmq/b_5653/g_00005/ipc/qmb.s Attached:1
Locked : 0
MMF: Dec 16 16:48:35 Name: /var/tmp/dmq/b_5653/g_00005/ipc/qmb.mmf Size: 16804
SHM: Dec 16 16:48:35 Create: b_5653/g_00005/ipc/qmb.m Size:20480
Address: 3233009664
MMF: Dec 16 16:48:35 Mapped: /var/tmp/dmq/b_5653/g_00005/ipc/qmb.mmf
Size:20480 Address: 3233009664
DMQ: Dec 16 16:48:35 Attach - Queue Number : 5.8
QMB: Dec 16 16:48:35 MQS Reply Queue: MQS_DMC_REPLYQ
DMQ: Dec 16 16:48:35 Locate Queue  = DMQ_DMC_REPLYQ    Name length   = 14

QMB: Dec 16 16:48:35 QMB Direction is QMBDM

DMQ: Dec 16 16:48:35 DMQ Reply Queue: DMQ_DMC_REPLYQ MRQ Address:'5.7'

QMB: Dec 16 16:48:35 OPEN: /home3/popp/mqsc_test/dmc500_config.dat
DMQ: Dec 16 16:48:35 Locate Queue  = MQS_ECHO    Name length   = 8
CFG: Dec 16 16:48:35 LSQ: MQS_ECHO  Format Name:
CFG: Dec 16 16:48:35 Q Table LSQ: MQS_ECHO CI: 12 LTYPE: D
```

```
CFG: Dec 16 16:48:35 Q Table RSQ: MQS_ECHO_SERVER RTYPE: S Header: N State: 1
MQS: Dec 16 16:48:35 Assigned MQS CI: 150
CFG: Dec 16 16:48:35 LSQ: DMQ_ECHO  Format Name:
CFG: Dec 16 16:48:35 Q Table LSQ: DMQ_ECHO CI: 150 LTYPE: M
CFG: Dec 16 16:48:35 Q Table RSQ: DMQ_ECHO_SERVER RTYPE: S Header: N State: 1

QMB: Dec 16 16:48:35 User Defined Priority Mapping is Enabled

QMB: Dec 16 16:48:35 DMQ LSQ Count: 1 MQS LSQ Count: 2

DMQ: Dec 16 16:48:35 SEL_MASK 1073817752 Que: 0
DMQ: Dec 16 16:48:35 SEL_MASK 1073817836 Que: 7
DMQ: Dec 16 16:48:35 SEL_MASK 1073817920 Que: 12
DMQ: Dec 16 16:48:35 PAMS_SET_SELECT Success - 3 Queues
DMQ: Dec 16 16:48:35 Locate Queue  = QMBMD    Name length   = 5
DMQ: Dec 16 16:48:35 Register Interest in DMQ process '5.9'
DMQ: Dec 16 16:48:35 PUT tar: 5.99 class:29 type:-1180 len:16 dm:39 prio:0
rspq:0.8
DMQ: Dec 16 16:48:35 Posting DMQ_RECV Select VAR: 1 MODE: -1
DMQ: Dec 16 16:48:35 GET: Q:8 src:5.99 class:29 type:-1182 len:6
prio:0
endian:1
DMQ: Dec 16 16:48:35 TPQ: 8 Src: 5.99 Class: 29 Type: -1182
DMQ: Dec 16 16:48:35 QMB AVAIL Registration Success
DMQ: Dec 16 16:48:35 Posting DMQ_RECV Select VAR: 1 MODE: -1
.
.
.
DMQ: Dec 16 16:48:37 QMB Process AVAILABLE '5.9'
DMQ: Dec 16 16:48:37 Posting DMQ_RECV Select VAR: 1 MODE: -1
DMQ: Dec 16 16:50:02 GET: Q:12 src:5.201 class:32000 type:-5010 len:100 prio:0
endian:1
DMQ: Dec 16 16:50:02 LSQ: 12 RSQ: MQS_ECHO_SERVER SRC: 5.201
MQS: Dec 16 16:50:02 Open RSQ: MQS_ECHO_SERVER

MQS: Dec 16 16:50:02 Open Queue: MQS_ECHO_SERVER Hobj:1073903024
MQS: Dec 16 16:50:02 PUT - DMQ CLI ADD: 00005.201
MQS: Dec 16 16:50:02 Send Request: RSQ: MQS_ECHO_SERVER Size: 100 Persistence: 0
Msg: DMQ_CLI
MQS: Dec 16 16:50:02 MQPUT: Send Queue: MQS_ECHO_SERVER CompCode: 0
DMQ: Dec 16 16:50:02 Posting DMQ_RECV Select VAR: 1 MODE: -1
DMQ: Dec 16 16:50:02 GET: Q:12 src:5.201 class:32000 type:-5010 len:100 prio:0
endian:1
DMQ: Dec 16 16:50:02 LSQ: 12 RSQ: MQS_ECHO_SERVER SRC: 5.201
MQS: Dec 16 16:50:02 PUT - DMQ CLI ADD: 00005.201
MQS: Dec 16 16:50:02 Send Request: RSQ: MQS_ECHO_SERVER Size: 100 Persistence: 0
Msg: DMQ_CLI
MQS: Dec 16 16:50:02 MQPUT: Send Queue: MQS_ECHO_SERVER
CompCode: 0
```

```
DMQ: Dec 16 16:50:02 Posting DMQ_RECV Select VAR: 1 MODE: -1
DMQ: Dec 16 16:50:20 GET: Q:8 src:5.202 class:32000 type:-5099 len:0 prio:0
endian:1
DMQ: Dec 16 16:50:20 TPQ: 8 Src: 5.202 Class: 32000 Type: -5099

QMB: Dec 16 16:50:20 QMB Server SHUTDOWN Request...

QMB: Dec 16 16:50:20 QMB Server Exiting
MMF: Dec 16 16:50:20 Write : /var/tmp/dmq/b_5653/g_00005/ipc/qmb.mmf Bytes: 20480
MMF: Dec 16 16:50:20 Close : /var/tmp/dmq/b_5653/g_00005/ipc/qmb.mmf
SEM: Dec 16 16:50:20 Delete: /var/tmp/dmq/b_5653/g_00005/ipc/qmb.s
SHM: Dec 16 16:50:20 Delete: /var/tmp/dmq/b_5653/g_00005/ipc/qmb.m Size:20480
Address: 3233009664
MQS: Dec 16 16:50:20 Successfully disconnected from Queue Manager
QMGR1


QMB: Dec 16 16:50:20 QMB Server exiting
```

Using IBM MQSeries Log Files

You can use the IBM MQSeries log files to determine any failures or events associated with various components. There are two ways to locate the IBM MQSeries log files:

- If the queue manager name is known and available, use the following pathnames on UNIX and Windows NT systems respectively:

  `/var/mqm/qmgrs/QMGRNAME/errors/AMQERR01.LOG`

  `c:\mqm\qmgrs\qmname\errors\AMQERR01.LOG`

- If the queue manager is not available, use the following pathnames on UNIX and Windows NT systems respectively:

  `/var/mqm/qmgrs/@SYSTEM/errors/AMQERR01.LOG`

  `c:\mqm\qmgrs\@SYSTEM\errors\AMQERR01.LOG`

- For first failure support technology, use the following pathnames on UNIX and Windows NT systems respectively:

  `/var/mqm/errors/AMQ`*`nnnnn`*`.`*`mm`*`.FDC`

  `c:\mqm\errors\AMQ`*`nnnnn`*`.`*`mm`*`.FDC`

Here, *nnnnn* is the process ID of the process reporting the error and *mm* is a sequence number, which is normally 0.

Using the BEA MessageQ MQSeries Connection Utility

The BEA MessageQ MQSeries Connection distribution media includes a utility called `qmb_util` which facilitates the sending of control messages to QMB processes. This utility allows you to control the behavior of a target QMB process, without having to close the process or rewrite the client or server application to send the desired control message to that process.

The `qmb_util` utility is located in `/`*`install_dir`*`/bin` on UNIX systems and in *`dev`*`:\`*`install_dir`*`\bin` on Windows NT systems.

How the BEA MessageQ MQSeries Connection Utility Works

The `qmb_util` utility sends a message containing the appropriate BEA MessageQ class and message type to perform the desired control function. In most cases, this message is sent to the `QMBDM` permanent queue.

Table 4-2 describes the valid control functions listed by message type.

Table 4-2 Valid Control Functions Listed by Message Type

Message Type	Description
`MSG_TYPE_PURGE_CI`	Purges all nonpersistent connection indexes (CI) from the CI table. This message is processed only by the permanent `QMBDM` server.
`MSG_TYPE_PURGE_CI_ALL`	Purges all CIs from the CI table. This message is processed only by the permanent `QMBDM` server.

Table 4-2 Valid Control Functions Listed by Message Type

Message Type	Description
`MSG_TYPE_NEW_LOG`	Closes the existing log and opens a new log. Sending this message to the permanent `QMBDM` server results in messages being transmitted to the associated `QMBMD` server and all of their Temporary Primary Queues (TPQs).
`MSG_TYPE_LOAD_CONFIG`	Reloads the QMB configuration file. Sending this message to the permanent `QMBMD` server results in messages being transmitted to the associated `QMBDM` server and all Temporary Primary Queues (TPQs).
`MSG_TYPE_RSQ_REGISTER`	Requires the RSQ registration structure in the message body of the message structure. Sending this message to the permanent `QMBDM` server results in messages being transmitted to the associated `QMBMD` server and all Temporary Primary Queues (TPQs).
`MSG_TYPE_EVENT_LOG`	Toggles QMB Server event logging. If event logging is enabled (that is, if you started the QMB with the `-e` parameter), use this message type to disable event logging. Likewise, if event logging is disabled, use this message type to enable event logging. This message must be sent to each QMB Server that requires a change in event logging status.
`MSG_TYPE_TRACE_LOG`	Toggles internal trace logging. If trace logging is on (that is, if you started the QMB with the `-t` parameter), use this message type to toggle trace logging off. Likewise, if trace logging is off, use this message type to toggle trace logging on. If verbose tracing is enabled, this message disables trace logging. Verbose logging can only be enabled using the -v parameter on the QMB Server command line. This message must be sent to each QMB Server that requires a change in trace logging status.
`MSG_TYPE_DUMP_QTABLES`	Dumps MQSeries Message Descriptor (MQMD) to a log file. This message is processed only by the permanent `QMBDM` server.

Table 4-2 Valid Control Functions Listed by Message Type

Message Type	Description
MSG_TYPE_TERMINATE	Terminates a QMB Server. If this message is sent to a Permanent QMB Server, the termination request is forwarded to all associated TPQs.

The qmb_util utility sends most control messages to the QMBDM primary queue (which is also the default target [QMBDM]). However, some control messages work on a specific QMBDM or QMBMD process and must be directed to that process's group and queue. For example, you can terminate a specific temporary QMB process without terminating the permanent process.

The qmb_util allows you to change the target group and queue so that control messages can be sent to queues other than the QMBDM primary queue (the default target).

Starting the BEA MessageQ MQSeries Connection Utility

Before you can run the qmb_util utility, set the DMQ_BUS_ID and DMQ_GROUP_ID environment variables. Then run qmb_util by entering the following command:

```
% qmb_util
```

The qmb_util displays the menu as shown in Listing 4-2.

Listing 4-2 The qmb_util Main Menu

```
        MessageQ MQSeries Connection Utility V5.0

1)  Toggle the QMB event log switch  (MSG_TYPE_EVENT_LOG)
2)  Toggle the QMB trace log switch  (MSG_TYPE_TRACE_LOG)
3)  Reload QMB config file           (MSG_TYPE_LOAD_CONFIG)
4)  Close old and open new log file  (MSG_TYPE_NEW_LOG)
5)  Purge non-persistent CI only     (MSG_TYPE_PURGE_CI)
6)  Purge all CI                     (MSG_TYPE_PURGE_CI_ALL)
7)  Dump MQMD to Log File            (MSG_TYPE_DUMP_QTABLES)
8)  Dynamic Service Registration     (MSG_TYPE_RSQ_REGISTER)
9)  Terminate the QMB processes      (MSG_TYPE_TERMINATE)
```

```
10) Change target group and queue
11) Set target group and queue to default [QMBDM] primary
12) Exit this utility

Default Target [QMBDM]  5.12
Current Target Default [QMBDM]  5.12

Enter Message Choice:
```

Note the field, below the menu, labelled "`Current Target Default [QMBDM]`." The value in this field (5.12) indicates tha target to which control messages are sent. (Specifically, "5.12" represents group 5 and queue 12 of the target.) For more information, see "Understanding Current and Default Target Groups and Queues" in this chapter.

Understanding Current and Default Target Groups and Queues

When you invoke `qmb_util`, the default target line is identified in a field below the main menu, as shown in Listing 4-2. The value in this field depends on whether or not the location of the target was known when `qmb_util` was started. Table 4-3 describes valid values in the "Default Target" and "Current Target Default" fields.

Table 4-3 Target Line Display

Display from Main Menu	Description
`Default Target [QMBDM] G.Q`	The target was located during startup. (*`G.Q`* indicates a valid group and queue number such as 5.12.)
`Default Target [QMBDM] UNDEFINED`	The `qmb_util` utility could not locate the `QMBDM` process.
`Current Target Default [QMBDM]G.Q`	The default target was located during start up. (*`G.Q`* indicates a valid group and queue number such as 5.12.)

Table 4-3 Target Line Display

Display from Main Menu	Description
`Current Target Group:[UNDEFINED]` `Queue : [UNDEFINED]`	The `qmb_util` utility could not locate the current target.

After `qmb_util` is started, the current target may be changed to any `group.queue` using choice 10 on the menu (`Change target group and queue`).

After you enter a new target group and queue, the `qmb_util` displays a message similar to the following:

```
Current Target Group : [x] Queue : [x]
```

It is this current target group to which `qmb_util` control messages are sent. Control messages must be sent to their proper target for the desired QMB action to take place.

Most `qmb_util` control messages need only a target and message type in order to complete the requested action. Some control messages, however, require additional information. The `qmb_util` utility prompts for this information.

Selecting the Terminate QMB Process Message Choice

To terminate the QMB process that you specify, select choice 9 from the Connection Utility Menu. When you select choice 9, the menu shown in Listing 4-3 is displayed.

Listing 4-3 Terminate Queue Message Bridge Process Menu

```
1) Terminate QMBDM & all temporary instances  (MessageQ to MQSeries)
2) Terminate QMBMD & all temporary instances  (MQSeries to MessageQ)
3) Terminate a single temporary QMB instance  (DMQ Address required)
4) Terminate QMBDM, QMBMD and all temporary instances  (Bridge Set)

Select terminate choice or <CR> to exit:
```

Selecting the Dynamic Service Registration Message Choice

You can change the registration of a queue (the BEA MessageQ RSQ) by selecting choice 8 (`Dynamic Service Registration`) from the menu. When you select choice 8, the prompts shown in Listing 4-4 are displayed:

Listing 4-4 Dynamic Service Registration for a BEA MessageQ RSQ

```
Enter LSQ name to assign the RSQ to:
Enter LSQ Owner  (M or D):
Enter MessageQ RSQ group number
Enter MessageQ RSQ queue number:
```

When you select the Dynamic Service Registration message choice to change the IBM MQSeries RSQ, the prompts shown in Listing 4-5 are displayed:

Listing 4-5 Dynamic Service Registration for an MQSeries RSQ

```
Enter LSQ name to assign the RSQ to:
Enter LSQ Owner (M or D):
Enter MQSeries queue name of RSQ:
```

Selecting the Close Old and Open New Log File Message Choice

To close the QMB Server's current log file and open a new log file, select choice 4 (`Close old and open new log file`) from the Connection Utility menu. The new log file has the same name as the old log file except that the new one has a tilde (~) suffix. For example, if the name of the current log file is `dmc_dm.log`, selecting choice

4 causes the log file to be closed and a file named `dmc_dm.log~` to be created. A subsequent invocation of message choice causes another tilde to be appended to the file name. In this case, a file named `dmc_dm.log~~` will be created.

If this message is sent to the permanent `QMBDM` server, it is automatically forwarded to the associated `QMBMD` server and all of their Temporary Primary Queues (TPQs).

Selecting the Reload QMB Configuration File Message

To direct the QMB Servers to reload their configuration files, select choice 3 (`Reload QMB config file`) from the Connection Utility menu. This allows the configuration file to be updated and reloaded without stopping and restarting the QMB Servers. New queue definitions can be added through this process, but queue definitions cannot be deleted. To delete queue definitions the QMB Servers must be restarted.

The Priority Mapping section of the configuration file can be modified and then reloaded to dynamically update the priority mapping scheme in use.

If this message is sent to the permanent QMBDM Server then it is automatically forwarded to the associated QMBMD Server and all of their TPQs.

Exiting the BEA MessageQ MQSeries Connection Utility

To exit `qmb_util`, select choice 12 (`Exit this utility`) from the Connection Utility menu.

A Programming Examples

The BEA MessageQ MQSeries Connection media includes a set of programming examples that show how applications can use interprocess message queuing to exchange information. The following programming examples are included:

- BEA MessageQ MQSeries Connection client/server programs
- Programs for testing BEA MessageQ MQSeries Connection

The programming examples are located in `/install_dir/examples/mqsc` on UNIX systems and in `dev:\install_dir\examples\mqsc` on Windows NT systems.

Note: Do not modify these examples in the `example` directory. If you want to use an example as a starting point to develop an application, copy the programming example to your working directory and edit it there.

This appendix discusses the following topics:

- Using the Programming Examples
- Testing the Programming Examples

Using the Programming Examples

This section describes the programming examples for BEA MessageQ and IBM MQSeries applications. The programming examples consist of a set of client and server programs designed to test a simple message exchange between two processes

running in the BEA MessageQ and IBM MQSeries environments. The programming examples report statistical results on message exchange rates and are designed to run in pairs.

Table A-1 describes the programming examples that are provided in the kit.

Table A-1 Programming Examples

Program Name	Description
QMB_DMQECHO	A BEA MessageQ server application that attaches to an RSQ and receives requests from an IBM MQSeries LSQ.
QMB_MQSECHO	An IBM MQSeries client application that reads from an RSQ and receives requests from an BEA MessageQ LSQ.
QMB_DMQCLIENT	A BEA MessageQ client application that sends requests to a BEA MessageQ LSQ which are then forwarded by the QMBDM to the corresponding IBM MQSeries RSQ.
QMB_MQSCLIENT	An IBM MQSeries client application that sends requests to an IBM MQSeries LSQ which are then forwarded by the QMBMD to the corresponding BEA MessageQ RSQ.

The programming examples are designed to echo back messages from the following client and servers:

- BEA MessageQ client (QMB_DMQCLIENT) to an IBM MQSeries server (QMB_MQSECHO).
- IBM MQSeries client (QMB_MQSCLIENT) to a BEA MessageQ server (QMB_DMQECHO).

When using these programming examples, you can have more than one instance of the client running, but only one instance of the server running.

Building the Programming Examples

To build the programs, invoke the appropriate makefile by using the make command. Two makefiles are available: */install_dir/*examples/mqsc/makefile for UNIX systems and *dev*:*install_dir*\examples\mqsc\mqsc.mak for Windows NT systems. For Windows NT, build the examples using the following command:

```
nmake -f mqsc.mak
```

QMB_DMQECHO

QMB_DMQECHO is a BEA MessageQ program designed to listen on a well-known address. It returns any message it receives, immediately after receipt, to the sender.

In this example, the address is named DMQ_ECHO_SERVER. The BEA MessageQ group initialization file must contain an entry for a queue named DMQ_ECHO_SERVER. This entry must be included in the QUEUE CONFIGURATION SECTION (%QCT) or in the GROUP NAME TABLE SECTION (%GNT).

The command syntax for the QMB_DMQECHO program is as follows:

```
% qmb_dmqecho [-q queue_name] [-d] [-?]
```

Table A-2 describes the QMB_DMQECHO parameters.

Table A-2 QMB_DMQECHO Command Parameters

Parameter	Description
-q	Defines the source queue name. This parameter is optional. The default name is DMQ_ECHO_SERVER.
-d	Enables debug mode to print trace information. This parameter is optional. The default setting is debugging mode disabled.
-?	Prints a simple help message. This parameter is optional.

QMB_MQSECHO

QMB_MQSECHO is an IBM MQSeries application program designed to listen on the queue named on the command line, which must be a valid, configured IBM MQSeries queue.

The command syntax for the QMB_MQSECHO program is as follows:

```
% qmb_mqsecho -r qname [-m MQMname] [-d] [-?]
```

Table A-3 describes the QMB_MQSECHO parameters.

Table A-3 QMB_MQSECHO Command Parameters

Parameter	Description
`-r qname`	Specifies the name of a valid, configured IBM MQSeries RSQ from which `QMB_MQSECHO` expects to read requests. This parameter is required.
`-m MQMname`	Specifies the name of the IBM MQSeries queue manager (MQM) to which `QMB_MQSECHO` should connect. This parameter is optional. If no MQM name is supplied, the default IBM MQSeries MQM is used.
`-d`	Enables debugging mode to print trace information. This parameter is optional. The default setting is debugging mode disabled.
`-?`	Prints a simple help message. This parameter is optional.

QMB_DMQCLIENT

`QMB_DMQCLIENT` is a BEA MessageQ application designed to send requests to the `QMB_MQSECHO` and then read the replies. The target group is the BEA MessageQ group in which the QMB is running. The target queue number is the BEA MessageQ LSQ that maps to the RSQ on which `QMB_MQSECHO` is listening. The `dmc500_config.dat` file should include the name associated with the target queue number used by the `QMB_DMQCLIENT`. This file maps the LSQ to the RSQ in the IBM MQSeries environment.

The command syntax of the `QMB_DMQCLIENT` program is as follows:

```
% qmb_dmqclient -g n -q n [-i] [-l n] [-b n] [-s n] [-d] [-p n] [-c correlid] [-j]
[-?]
```

Table A-4 describes the parameters for the `QMB_DMQCLIENT` program.

Table A-4 QMB_DMQCLIENT Command Parameters

Parameter	Description
`-g n`	Specifies the number of the target BEA MessageQ LSQ group. This parameter is required.

Table A-4 QMB_DMQCLIENT Command Parameters

Parameter	Description
`-q` *n*	Specifies the number of the target BEA MessageQ LSQ queue. This parameter is required.
`-i`	Specifies interactive mode. If this parameter is specified, the following parameters are ignored: `-l`, `-b`, and `-s`. This parameter is optional.
`-l` *n*	Specifies the loop count, which is the number of iterations. This parameter is optional. The default is 1 iteration.
`-b` *n*	Specifies the burst count, which is the number of messages per iteration. This parameter is optional. The default is 1 message.
`-s` *n*	Specifies the size of the message, in bytes. The minimum size is 60, which allows for the sequence number and time stamp. If this parameter is not specified, the default is 60. The maximum is 4194304 bytes.
`-d`	Specifies debugging mode, which prints trace information. This parameter is optional.The default is debugging mode disabled.
`-p`	The number indicating the priority of the message. This parameter is optional. The valid range is 0-1 for BEA MessageQ V4.0A and 0-99 for BEA MessageQ V5.0.
`-c`	The correlation identifier. This parameter is optional. The default is no correlation identifier. This feature is supported only for BEA MessageQ V5.0.
`-j`	Enables journaling. This parameter is optional. The default is no journaling.
`-?`	Prints a simple help message. This parameter is optional.

QMB_MQSCLIENT

`QMB_MQSCLIENT` is an IBM MQSeries client application designed to send requests to `QMB_DMQECHO` and then read any replies from the named reply queue. The target queue is the IBM MQSeries LSQ queue name that maps to the RSQ on which QMB_DMQECHO is listening. The target queue is specified in the `config.dat` file.

The command syntax for the `QMB_MQSCLIENT` program is as follows:

```
% qmb_mqsclient -t name -r name [-m MQMname][-d][-i] [-l n] [-b n] [-s n] [-p n]
[-c correlid] [-j] [-?]
```

Table B-5 describes the QMB_MQSCLIENT parameters.

Table A-5 QMB_MQSCLIENT Command Parameters

Parameter	Description
-t *name*	Specifies the name of the target IBM MQSeries LSQ queue. This parameter is required.
-r *name*	Specifies the name of the reply queue. This parameter is required.
-m *name*	Specifies the name of the IBM MQSeries Queue Manager (MQM). This parameter is optional. If no MQM is specified, the default MQM is used.
-d	Enables debugging mode, which prints trace information. This parameter is optional.
-i	Specifies interactive mode. The program prompts for a text message. If this parameter is specified, the following parameters are ignored: -l, -b, and -s.
-l *n*	Specifies the loop count. This parameter is optional. The default count is 1.
-b *n*	Specifies the burst count. This parameter is optional. The default count is 1.
-s *n*	Specifies the size of the message, in bytes. The minimum size is 60 bytes. If the size is not specified, the default is 60, which allows for the sequence number and time stamp. The maximum size is 4194304 bytes.
-p *n*	The number indicating the priority of the message. This parameter is optional. The valid range is 0-9 for IBM MQSeries V5.0. The default is 0.
-c *correlid*	The correlation identifier. This parameter is optional. The default is no correlation identifier. This feature is supported only for BEA MessageQ V5.0.
-j	Persistent messaging is enabled. This parameter is optional. The default is persistent messaging disabled.
-?	Prints a simple help message

Running the QMB_MQSECHO and QMB_DMQCLIENT Test Pair

To run the QMB_MQSECHO and QMB_DMQCLIENT test pair, you must first set up the QMB configuration file and the BEA MessageQ initialization file with the proper parameters. Then, you can invoke the QMB_MQSECHO and QMB_DMQCLIENT programs.

For example, suppose the QMB is running on group 5 and the desired target queue of QMB_DMQCLIENT is MQS_ECHO_SERVER. In this case, the QMB_DMQCLIENT sends a request to the queue number defined as the LSQ in the DMC320_CONFIG.DAT file that maps to the MQS_ECHO_SERVER. Listing A-1 shows a QMB configuration file.

Listing A-1 Queue Message Bridge Configuration File

```
!LSQ        LSQ     RSQ              RSQ
!Name       Owner   Name             Association
!
MQS_ECHO    D       MQS_ECHO_SERVER  S
```

Listing A-2 shows the entry you must provide for group 5 in the QUEUE CONFIGURATION SECTION (%QCT) of the BEA MessageQ initialization file, group.init.

Listing A-2 QUEUE CONFIGURATION SECTION

```
!QUEUE CONFIGURATION SECTION
!
% QCT

!Queue   Queue  Byte   Msg   Quota  UCB  Queue Owner Conf  Perm   Name
!Name    Number Quota  Quota Enable Send Type  Queue Style Active Scope Security
MQS_ECHO          12   64000   100   None   .    M     0     .     Y      L     N
```

To run the test pair, you must invoke both the QMB_MQSECHO and QMB_DMQCLIENT programs. Enter the following command on the IBM MQSeries system to invoke the QMB_MQSECHO program:

```
% qmb_mqsecho  -r MQS_ECHO_SERVER
```

Enter the following command on the BEA MessageQ system to invoke the QMB_DMQCLIENT program:

```
% qmb_dmqclient -g 5 -q 12 -l 10 -b 2 -s 1024 -d
```

In this example, MQS_ECHO, which is defined in the BEA MessageQ initialization file, is identified by the -g 5 and -q 12 parameters.

Running the QMB_MQSCLIENT and QMB_DMQECHO Test Pair

Before you can run the QMB_MQSCLIENT and QMB_DMQECHO test pair, you must first set up the QMB configuration file and the BEA MessageQ initialization file with the proper parameters.

For example, suppose the target queue of the QMB_MQSCLIENT is DMQ_ECHO_SERVER. In this case, QMB_MQSCLIENT sends a request to the LSQ (defined in the dmc500_config.dat file) that maps to the RSQ (named DMQ_ECHO_SERVER).

Listing A-3 shows the dmc500_config.dat file.

Listing A-3 Queue Message Bridge Configuration File

```
!LSQ        LSQ     RSQ               RSQ
!Name       Owner   Name              Association
!
DMQ_ECHO     M      DMQ_ECHO_SERVER   S
```

Listing A-4 shows the QUEUE CONFIGURATION SECTION in which group 5 is defined in the BEA MessageQ Initialization File (group.init).

Listing A-4 QUEUE CONFIGURATION SECTION

```
!QUEUE CONFIGURATION SECTION
!
% QCT

!Queue   Queue  Byte   Msg   Quota  UCB  Queue Owner Conf  Perm   Name
!Name    Number Quota  Quota Enable Send Type  Queue Style Active Scope Security
!
DMQ_ECHO_SERVER 15 64000 100 None   .    M     0     .     Y      L     N
```

If the `DMQ_ECHO_SERVER` runs on a remote BEA MessageQ group, the entry shown in Listing A-5 is needed in the `GROUP NAME TABLE SECTION` (`%GNT`):

Listing A-5 GROUP NAME TABLE SECTION

```
!GROUP NAME TABLE SECTION
!
% GNT
DMQ_ECHO_SERVER 62.15 L
```

To run the test pair, you must invoke both the `QMB_DMQECHO` and `QMB_MQSCLIENT` programming examples. On the IBM MQSeries system, enter the following command to invoke the `QMB_MQSCLIENT` program:

```
% qmb_mqsclient -t DMQ_ECHO -r MQS_CLIENT_REPLYQ -m QMNAME  -l 10
  -b 2  -s 1024  -d
```

Enter the following command on a BEA MessageQ system to invoke the `QMB_DMQECHO` program:

```
% qmb_dmqecho  -d
```

Testing the Programming Examples

BEA MessageQ and IBM MQSeries both provide programming examples that help you learn how these products work. You can also use these programs to verify the BEA MessageQ MQSeries Connection installation and configuration. To verify, use the BEA MessageQ Test Utility (`dmqtestm` or `dmqtestc`) and the IBM MQSeries programming examples (`amqsget0` and `amqsput0`). With these programming examples and the Test Utility, simple datagram messages can be exchanged by different queue messaging environments (BEA MessageQ and IBM MQSeries) if the BEA MessageQ MQSeries Connection servers (QMB server) are properly configured.

For more information about the BEA MessageQ Test Utility, see the *BEA MessageQ Programmer's Guide*.

For information about the IBM MQSeries `amqsget0` and `amqsput0` programming examples, see the *MQSeries Application Programming Guide*.

You can perform simple datagram messaging with out-of-the-box BEA MessageQ and IBM MQSeries test tools and examples, if there is a properly configured QMB server pair. However, you cannot perform other types of message exchange (such as request or reply) because these applications do not follow programming conventions that allow the QMB to exchange messages. To perform request and reply message exchange, you need to modify the applications to pass the appropriate BEA MessageQ type and class fields and to fill in the IBM MQSeries `MQS_CLIENT_REPLYQ` fields.

BEA MessageQ MQSeries Connection provides an example `dmc500_config.dat` file, `dmc500_dmq_group.init` file, an MQSC command file (`dmc500_mqmdef.cfg`), and example scripts to start a QMB server pair.

The following examples are based on the configuration established and the queue associations defined in the `dmc500_config.dat` file.

Note: Use the `dmc500_dmq_group.init` file for BEA MessageQ startup. You can copy this file from */*`install_dir`*/*`templates` on UNIX systems and from `dev`:\`install_dir`\`templates` on Windows NT systems.

Testing the IBM MQSeries Connection to BEA MessageQ

To send an IBM MQSeries message to BEA MessageQ, complete the following procedure:

1. In a test window on the BEA MessageQ side, run the `dmqtestc` utility. Using this utility, Attach By Name to the `DMQ_ECHO_SERVER` queue.
2. Enter the command to start `amqsput` on the system running IBM MQSeries. Invoke `amqsput` with `DMQ_ECHO` as the target queue and `QMGR1` as the queue manager name. (The target queue is the IBM MQSeries LSQ, which maps to the BEA MessageQ RSQ through the `dmc500_config.dat` file.) For example, on an HP-UX system:

   ```
   # /opt/mqm/samp/bin/amqsput DMQ_ECHO QMGR1
   ```

   When `amqsput` starts, it displays the start message and the name of the target queue.
3. Enter messages. `amqsput` expects ASCII input followed by the newline character. Once `amqsput` gets a message, it displays it. For example:

   ```
   Hello message 1 from MQSeries
   Hello message 2 from MQSeries
   Last message from MQSeries
   ```
4. From the `dmqtestc` utility, issue three `GET` commands to read the inbound messages from `amqsput`.

See the following section for instructions on testing the BEA MessageQ connection to IBM MQSeries.

Testing the BEA MessageQ Connection to IBM MQSeries

On the BEA MessageQ side, you can test the connection using `dmqtestc`. Complete the following procedure:

1. Invoke the `dmqtestc` program.
2. Issue a `PUT` command with the text "`Hello from BEA MessageQ`" to queue `MQS_ECHO`. (You can use the `LOCATE_Q` command to return the queue number fro `MQS_ECHO`.)

3. Run `amqsget` with a source queue name of `MQS_ECHO_SERVER` and a queue manager name of `QMGR1`. (The source queue name is the name of the IBM MQSeries RSQ which maps to the MessageQ LSQ through the `dmc500_config.dat` file.) For example, on an HP-UX system:

   ```
   # /opt/mqm/samp/bin/amqsget MQS_ECHO_SERVER QMGR1
   ```

   The `amqsget` utility should have received one message similar to the following:

   ```
   Sample AMQSGET start
   message <Hello from BEA MessageQ>
   no more messages
   Sample AMQSGET end
   ```

B Messages

The qmb_util utility allows you to control the behavior of a target QMB process (by sending a control message to that process) without having to close the process or rewrite the client or server application. Control messages are requests to the QMB Server sent via the BEA MessageQ API function pams_put_msg.

This appendix describes the control messages that can be sent to QMB processes using qmb_util:

- DUMP_QTABLES
- EVENT_LOG
- LOAD_CONFIG
- NEW_LOG
- PURGE_CI
- PURGE_CI_ALL
- QMB_TERMINATE
- RSQ_REGISTER
- TRACE_LOG

For more information on the qmb_util utility, see "Using the BEA MessageQ MQSeries Connection Utility" in Chapter 4, "Managing the BEA MessageQ MQSeries Connection Environment."

DUMP_QTABLES

The DUMP_QTABLES message dumps IBM MQSeries Message Descriptor (MQMD) to a log file.

C Message Structure

```
#include qmb_user.h
```

Arguments

Arguments	pams_put_msg Format
Type	MSG_TYPE_DUMP_QTABLES
Class	MSG_CLAS_QMB
Size	0

Message Data Fields

None

EVENT_LOG

The EVENT_LOG message toggles QMB Server event logging. If event logging is enabled (that is, if you started the QMB with the -e parameter), use this message type to disable event logging. Likewise, if event logging is disabled, use this message type to enable event logging.

This message must be sent to each QMB Server that requires a change in event logging status.

C Message Structure

```
#include qmb_user.h
```

Arguments

Arguments	pams_put_msg Format
Type	MSG_TYPE_EVENT_LOG
Class	MSG_CLAS_QMB
Size	0

Message Data Fields

None.

LOAD_CONFIG

The LOAD_CONFIG message reloads the QMB configuration file. Sending this message to the permanent QMBMD server results in messages being transmitted to the associated QMBDM server and all Temporary Primary Queues (TPQs).

C Message Structure

```
#include qmb_user.h
```

Arguments

Arguments	pams_put_msg Format
Type	MSG_TYPE_LOAD_CONFIG
Class	MSG_CLAS_QMB
Size	0

Message Data Fields

None

NEW_LOG

The NEW_LOG message closes the existing log and opens a new log. Sending this message to the permanent QMBDM server results in messages being transmitted to the associated QMBMD server and all of their Temporary Primary Queues (TPQs).

C Message Structure

```
#include qmb_user.h
```

Arguments

Arguments	pams_put_msg Format
Type	MSG_TYPE_NEW_LOG
Class	MSG_CLAS_QMB
Size	0

Message Data Fields

None

PURGE_CI

The `PURGE_CI` message purges all nonpersistent connection indexes (CI) from the CI table. This message is processed only by the permanent `QMBDM` server

C Message Structure

```
#include qmb_user.h
```

Arguments

Arguments	pams_put_msg Format
Type	`MSG_TYPE_PURGE_CI`
Class	`MSG_CLAS_QMB`
Size	`0`

Message Data Fields

None

PURGE_CI_ALL

The PURGE_CI_ALL message purges all CIs from the CI table. This message is processed only by the permanent QMBDM server

C Message Structure

```
#include qmb_user.h
```

Arguments

Arguments	pams_put_msg Format
Type	MSG_TYPE_PURGE_CI_ALL
Class	MSG_CLAS_QMB
Size	0

Message Data Fields

None

QMB_TERMINATE

The QMB_TERMINATE message terminates a QMB Server. If this message is sent to a Permanent QMB Server, the termination request is forwarded to all associated TPQs.

C Message Structure

```
#include qmb_user.h
```

Arguments

Arguments	pams_put_msg Format
Type	MSG_TYPE_QMB_TERMINATE
Class	MSG_CLAS_QMB
Size	0

Message Data Fields

None

RSQ_REGISTER

The RSQ_REGISTER message requires the RSQ registration structure in the message body of the message structure. Sending this message to the permanent QMBDM server results in messages being transmitted to the associated QMBMD server and all Temporary Primary Queues (TPQs).

C Message Structure

```
#include qmb_user.h
```

Arguments

Arguments	pams_put_msg Format
Type	MSG_TYPE_RSQ_REGISTER
Class	MSG_CLAS_QMB
Size	sizeof(rsq_reg_struct);

RSQ Registration Structure

```
typedef struct {                /* (MAX_MQS_LEN = 49)          */
   char      lsq[49];           /* LSQ name to assign RSQ to   */
   char      lowner;            /* LSQ Owner [D]MQ or [M]QS    */
   char      rsq[49];           /* RSQ name of MQS Appl Queue  */
   char      rfu;               /* Reserved Future Use         */
   q_address rsq_add;           /* DMQ address of  DMQ RSQ     */
} rsq_reg_struct;
```

TRACE_LOG

The TRACE_LOG message toggles internal trace logging. If trace logging is on (that is, if you started the QMB with the -t parameter), use this message type to toggle trace logging off. Likewise, if trace logging is off, use this message type to toggle trace logging on.

If verbose tracing is enabled, this message disables trace logging. Verbose logging can only be enabled using the -v parameter on the QMB Server command line.

This message must be sent to each QMB Server that requires a change in trace logging status.

C Message Structure

```
#include qmb_user.h
```

Arguments

Arguments	pams_put_msg Format
Type	MSG_TYPE_TRACE_LOG
Class	MSG_CLAS_QMB
Size	0

Message Data Fields

None

Index

E

F

G

I

L

M

N

P

Q