WebSphere MQ

Using Java

Version 6.0

SC34-6591-02

WebSphere MQ

Using Java

Version 6.0

SC34-6591-02

Note!
Before using this information and the product it supports, be sure to read the general information under Appendix G, "Notices," on page 681.

Third edition (March 2007)

This edition of the book applies to the following products:

- IBM WebSphere MQ, Version 6.0
- IBM WebSphere MQ for z/OS, Version 6.0

and to any subsequent releases and modifications until otherwise indicated in new editions.

Contents

Figures

Tables

About this book

This book describes:

- WebSphere® MQ classes for Java™, which can be used to access WebSphere MQ systems
- WebSphere MQ classes for Java Message Service, which can be used to access both Java Message Service (JMS) and WebSphere MQ applications

Note: See the WebSphere MQ README file for information that expands and corrects the information in this book.

Who this book is for

This information is written for programmers who are familiar with the procedural WebSphere MQ application programming interface as described in the *WebSphere MQ Application Programming Guide*. It shows how to transfer this knowledge to become productive with the WebSphere MQ Java programming interfaces.

What you need to know to understand this book

You need:

- Knowledge of the Java programming language
- Understanding of the purpose of the message queue interface (MQI) as described in the *WebSphere MQ Application Programming Guide* and the chapter about Call Descriptions in the *WebSphere MQ Application Programming Reference*
- Experience of WebSphere MQ programs in general, or familiarity with the content of the other WebSphere MQ publications

Users intending to use WebSphere MQ classes for Java with CICS® Transaction Server for OS/390® or CICS Transaction Server for z/OS® also need to be familiar with:

- Customer Information Control System (CICS) concepts
- Using the CICS Java Application Programming Interface (API)
- Running Java programs from within CICS

How to use this book

Part 1 of this book tells you how to use WebSphere MQ base Java and WebSphere MQ JMS; Part 2 helps programmers wanting to use WebSphere MQ base Java; Part 3 helps programmers wanting to use WebSphere MQ JMS.

First, read the topics in Part 1 that introduce you to WebSphere MQ base Java and WebSphere MQ JMS. Then use the programming guidance in Part 2 or 3 to understand how to use the classes to send and receive WebSphere MQ messages in the environment you want to use.

Terms used in this book

The term *WebSphere MQ base Java* means WebSphere MQ classes for Java.

The term *WebSphere MQ JMS* means WebSphere MQ classes for Java Message Service.

The term *WebSphere MQ Java* means WebSphere MQ classes for Java and WebSphere MQ classes for Java Message Service combined.

The term *i5/OS*® means any release of i5/OS or OS/400® supported by the current version of WebSphere MQ for iSeries™.

Linux® is used as a general term for any of the following platforms:

- Linux (POWER™ platform)
- Linux (x86 platform)
- Linux (zSeries® platform)

UNIX® *system* is used as a general term for any of the following platforms:

- AIX®
- HP-UX
- Linux
- Solaris

Windows® *system*, or just *Windows*, is used as a general term for any of the following platforms:

- Windows 2000
- Windows Server 2003
- Windows XP

The term *IP address* means either an Internet Protocol Version 4 (IPv4) address, expressed as a sequence of decimal numbers separated by dots, or an Internet Protocol Version 6 (IPv6) address, expressed as a sequence of hexadecimal numbers separated by colons.

Summary of changes

This section describes the changes in this edition of *WebSphere MQ Using Java*. Changes since the previous edition of the book are marked by revision bars to the left of the changes.

Changes for this edition (SC34-6591-02)

This edition contains the documentation for the following new function:

- Support for a 64-bit JVM when running applications in WebSphere Application Server on z/OS
- The WebSphere MQ resource adapter

This edition also contains various editorial improvements, clarifications, and corrections.

Changes for the previous edition (SC34-6591-01)

The locations of WebSphere MQ libraries for the new 64-bit platforms are added

This edition also contains various editorial improvements, clarifications, and corrections.

Part 1. Guidance for users

Chapter 1. Getting started

This chapter gives an overview of WebSphere MQ classes for Java and WebSphere MQ classes for Java Message Service and their uses.

What are WebSphere MQ classes for Java?

WebSphere MQ classes for Java (also referred to as WebSphere MQ base Java) allow a Java application to:

- Connect to WebSphere MQ as a WebSphere MQ client
- Connect directly to a WebSphere MQ server

WebSphere MQ base Java encapsulates the Message Queue Interface (MQI), the native WebSphere MQ API.

What are WebSphere MQ classes for Java Message Service?

WebSphere MQ classes for Java Message Service (also referred to as WebSphere MQ JMS) is a set of Java classes that implement Sun's Java Message Service (JMS) interfaces to enable JMS programs to access WebSphere MQ systems. This book describes an implementation of Version 1.1 of the JMS API specification, which is backwards compatible with previous versions of the specification.

Using WebSphere MQ JMS as the API to write WebSphere MQ applications has a number of benefits. Some advantages derive from JMS being an open standard with multiple implementations. Other advantages come from additional features that are present in WebSphere MQ JMS, but not in WebSphere MQ base Java.

Benefits arising from the use of an open standard include:

- The protection of investment, both in skills and application code
- The availability of people skilled in JMS application programming
- The ability to plug in different JMS implementations to fit different requirements

Sun's Web site at `http://java.sun.com` provides more information about the benefits of the JMS API.

The extra function provided over WebSphere MQ base Java includes:

- Asynchronous message delivery. Messages can be delivered to an application as they arrive, on a separate thread.
- Message selectors.
- Support for publish/subscribe messaging.
- Structured, more abstract, message classes. Implementation details are left to the JMS provider.

Connection options

Programmable options allow WebSphere MQ Java to connect to WebSphere MQ in either of the following ways:

- As a WebSphere MQ client using Transmission Control Protocol/Internet Protocol (TCP/IP)

- In bindings mode, connecting directly to WebSphere MQ using the Java Native Interface (JNI)

Table 1 shows which of these connection modes can be used for each platform.

Table 1. Platforms and connection modes

Application platform	Can an application connect in client mode?	Can an application connect in bindings mode?
AIX	Yes	Yes
HP-UX	Yes	Yes[1]
i5/OS	Yes	Yes
Linux (POWER platform)	Yes	Yes
Linux (x86 platform)	Yes	Yes
Linux (zSeries platform)	Yes	No[2]
Solaris	Yes	Yes
Windows 2000	Yes	Yes
Windows Server 2003	Yes	Yes
Windows XP	Yes	Yes
z/OS	No[3]	Yes

Note:

1. HP-UX Java bindings support is available only for HP-UXv11 systems running the POSIX draft 10 pthreaded version of WebSphere MQ.
2. On Linux (zSeries platform), only TCP/IP client connectivity is supported.
3. However, a WebSphere MQ JMS application running under WebSphere Application Server on z/OS can connect in client mode.

In addition, WebSphere MQ JMS publish/subscribe applications can connect directly across TCP/IP to a broker of any of the following products:

- WebSphere MQ Event Broker, Version 2.1
- WebSphere Business Integration Event Broker, Version 5.0
- WebSphere Business Integration Message Broker, Version 5.0

The following sections describe the client mode and bindings mode connection options in more detail.

Client connection

To use WebSphere MQ Java as a WebSphere MQ client, you can install it either on the WebSphere MQ server machine, which may also contain a Web server, or on a separate machine. If you install WebSphere MQ Java on the same machine as a Web server, you can download and run WebSphere MQ client applications on machines that do not have WebSphere MQ Java installed locally.

Bindings connection

When used in bindings mode, WebSphere MQ Java uses the Java Native Interface (JNI) to call directly into the existing queue manager API, rather than

communicating through a network. In some environments, connecting in bindings mode can provide better performance for WebSphere MQ Java applications than connecting in client mode.

Prerequisites

For the latest information about the prerequisites for WebSphere MQ Java, see the WebSphere MQ README file.

To develop WebSphere MQ Java applications, you need a Java 2 Software Development Kit (SDK).

To run WebSphere MQ Java applications, you need the following software components:

- A WebSphere MQ queue manager, for applications that connect to a queue manager
- One of the following publish/subscribe brokers, for publish/subscribe applications:
  - WebSphere MQ Publish/Subscribe
  - WebSphere MQ Integrator, Version 2
  - WebSphere MQ Event Broker, Version 2.1
  - WebSphere Business Integration Event Broker, Version 5.0
  - WebSphere Business Integration Message Broker, Version 5.0
- A Java Runtime Environment (JRE), for each system on which you run applications
- For i5/OS, QShell, which is option 30 of the operating system
- For z/OS, UNIX System Services (USS)

To determine the supported Java 2 SDKs for your platform, see www.ibm.com/software/integration/websphere/mqplatforms/supported.html . The supported JREs are those JREs that are embedded in the supported Java 2 SDKs.

To support Secure Sockets Layer (SSL) authentication fully, you need a JRE at Version 1.4.2 for your platform. A WebSphere MQ Java application can use SSL to obtain a secure connection to a queue manager, with authentication, message integrity, and data encryption.

If you require SSL connections to use cryptographic modules that have been FIPS 140-2 certified, you need the IBM® Java JSSE FIPS provider (IBMJSSEFIPS). Every IBM Java 2 SDK and JRE at Version 1.4.2 contains IBMJSSEFIPS.

You can use Internet Protocol Version 6 (IPv6) addresses in your WebSphere MQ Java applications provided IPv6 addresses are supported by your Java virtual machine (JVM) and the TCP/IP implementation on your operating system. The WebSphere MQ JMS administration tool (see Chapter 5, “Using the WebSphere MQ JMS administration tool,” on page 35) also accepts IPv6 addresses.

To use the WebSphere MQ JMS administration tool, you need one of the following service provider packages, supplied with WebSphere MQ:

- Lightweight Directory Access Protocol (LDAP) - ldap.jar, providerutil.jar.
- File system - fscontext.jar, providerutil.jar.

Prerequisites

These packages provide the Java Naming and Directory Service (JNDI) service. This is the resource that stores physical representations of the administered objects. Users of WebSphere MQ JMS probably use an LDAP server for this purpose, but the tool also supports the use of the file system context service provider. If you use an LDAP server, configure it to store JMS objects. For information to assist with this configuration, see the documentation for your LDAP server.

To use the XOpen/XA facilities of WebSphere MQ JMS on i5/OS you need a specific PTF. See the WebSphere MQ README file for further information.

Chapter 2. Installation and configuration

This chapter describes the directories and files that are created when you install WebSphere MQ Java, and tells you how to configure WebSphere MQ Java after installation.

What is installed

The latest version of WebSphere MQ Java is installed with WebSphere MQ. You might need to override default installation options to make sure this is done.

Refer to the following books for more information about installing WebSphere MQ:

WebSphere MQ for AIX Quick Beginnings
WebSphere MQ for HP-UX Quick Beginnings
WebSphere MQ for iSeries Quick Beginnings
WebSphere MQ for Linux Quick Beginnings
WebSphere MQ for Solaris Quick Beginnings
WebSphere MQ for Windows Quick Beginnings
WebSphere MQ for z/OS Program Directory

WebSphere MQ base Java is contained in the Java archive (JAR) file com.ibm.mq.jar.

WebSphere MQ JMS is contained in the JAR file com.ibm.mqjms.jar.

The following JAR files are also supplied with WebSphere MQ Java. These files are required by a WebSphere MQ JMS application that connects directly to a publish/subscribe broker.

- CL3Export.jar
- CL3Nonexport.jar
- rmm.jar

The following JAR file is also supplied with WebSphere MQ Java. This file is required by any WebSphere MQ JMS application.

- dhbcore.jar

The following Java libraries from Sun Microsystems are distributed with WebSphere MQ Java:

- connector.jar (Version 1.0)
- fscontext.jar (Version 1.2)
- jms.jar (Version 1.1)
- jndi.jar (Version 1.2.1)
- jta.jar (Version 1.0.1)
- ldap.jar (Version 1.2.2)
- providerutil.jar (Version 1.2)

The sample application called Postcard is in postcard.jar. For more information about this application; see “JMS Postcard” on page 17.

The Javadoc tool has been used to generate the HTML pages containing the specifications of the WebSphere MQ base Java and WebSphere MQ JMS APIs. This documentation is in mqjmsapi.jar.

When installation is complete, files and samples are installed in the locations shown in "Installation directories."

After installation, on any platform other than Windows, you must update your environment variables as described in "Environment variables."

Installation directories

Table 2 shows where the WebSphere MQ Java files are installed on each platform.

Table 2. WebSphere MQ Java installation directories

Platform	Directory
AIX	/usr/mqm/java
HP-UX, Linux, and Solaris	/opt/mqm/java
i5/OS	/QIBM/ProdData/mqm/java
Windows	*install_dir*\Java
z/OS	*install_dir*/mqm/V6R0M0/java
Note: *install_dir* is the directory in which you installed WebSphere MQ. On Windows, this directory is normally C:\Program Files\IBM\WebSphere MQ. On z/OS, this directory is likely to be /usr/lpp.	

Some sample applications, such as the Installation Verification Programs (IVPs), are supplied with WebSphere MQ. Table 3 shows where the sample applications are installed on each platform. The WebSphere MQ base Java samples are in a subdirectory called base, and the WebSphere MQ JMS samples are in a subdirectory called jms.

Table 3. Samples directories

Platform	Directory
AIX	/usr/mqm/samp/java
HP-UX, Linux, and Solaris	/opt/mqm/samp/java
i5/OS	/QIBM/ProdData/mqm/java/samples
Windows	*install_dir*\Tools\Java
z/OS	*install_dir*/mqm/V6R0M0/java/samples
Note: *install_dir* is the directory in which you installed WebSphere MQ. On Windows, this directory is normally C:\Program Files\IBM\WebSphere MQ. On z/OS, this directory is likely to be /usr/lpp.	

Environment variables

Before you can run WebSphere MQ Java applications, the setting for your CLASSPATH environment variable must include the appropriate WebSphere MQ Java code. To run the sample applications, the CLASSPATH setting must also include the appropriate samples directories.

To run WebSphere MQ base Java applications, including the WebSphere MQ base Java sample applications, use the CLASSPATH setting for your platform as shown in Table 4 on page 9.

Table 4. CLASSPATH setting to run WebSphere MQ base Java applications, including the WebSphere MQ base Java sample applications

Platform	CLASSPATH setting
AIX	CLASSPATH=/usr/mqm/java/lib/com.ibm.mq.jar: /usr/mqm/samp/java/base:
HP-UX, Linux, and Solaris	CLASSPATH=/opt/mqm/java/lib/com.ibm.mq.jar: /opt/mqm/samp/java/base:
i5/OS	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mq.jar: /QIBM/ProdData/mqm/java/samples/base:
Windows	CLASSPATH=*install_dir*\Java\lib\com.ibm.mq.jar; *install_dir*\Tools\Java\base;
z/OS	CLASSPATH=*install_dir*/mqm/V6R0M0/java/lib/com.ibm.mq.jar: *install_dir*/mqm/V6R0M0/java/samples/base:
Note: *install_dir* is the directory in which you installed WebSphere MQ. On Windows, this directory is normally C:\Program Files\IBM\WebSphere MQ. On z/OS, this directory is likely to be /usr/lpp.	

To run WebSphere MQ JMS applications, including the WebSphere MQ JMS sample applications, use the CLASSPATH setting for your platform as shown in Table 5.

Table 5. CLASSPATH setting to run WebSphere MQ JMS applications, including the WebSphere MQ JMS sample applications

Platform	CLASSPATH setting
AIX	CLASSPATH=/usr/mqm/java/lib/com.ibm.mqjms.jar: /usr/mqm/samp/java/jms:
HP-UX, Linux, and Solaris	CLASSPATH=/opt/mqm/java/lib/com.ibm.mqjms.jar: /opt/mqm/samp/java/jms:
i5/OS	CLASSPATH=/QIBM/ProdData/mqm/java/lib/com.ibm.mqjms.jar: /QIBM/ProdData/mqm/java/samples/jms:
Windows	CLASSPATH=*install_dir*\Java\lib\com.ibm.mqjms.jar; *install_dir*\Tools\Java\jms;
z/OS	CLASSPATH=*install_dir*/mqm/V6R0M0/java/lib/com.ibm.mqjms.jar: *install_dir*/mqm/V6R0M0/java/samples/jms:
Note: *install_dir* is the directory in which you installed WebSphere MQ. On Windows, this directory is normally C:\Program Files\IBM\WebSphere MQ. On z/OS, this directory is likely to be /usr/lpp.	

To run WebSphere MQ base Java and WebSphere MQ JMS applications, including the WebSphere MQ base Java and WebSphere MQ JMS sample applications, simply combine the CLASSPATH settings shown in Table 4 and Table 5.

The scripts provided with WebSphere MQ Java use the following environment variables:

MQ_JAVA_DATA_PATH
This environment variable specifies the directory for log and trace output. For information about how MQ_JAVA_DATA_PATH is used, see "Solving problems" on page 33.

MQ_JAVA_INSTALL_PATH
This environment variable specifies the directory where WebSphere MQ Java is installed, as shown in Table 2 on page 8.

MQ_JAVA_LIB_PATH
This environment variable specifies the directory where the WebSphere MQ Java libraries are stored, as shown in Table 6. Some scripts supplied with WebSphere MQ Java, such as IVTRun, use this environment variable.

On Windows, all the environment variables are set automatically during installation. On any other platform, you must set them yourself. On a UNIX system, you can use the script **setjmsenv** to set the environment variables. On AIX, **setjmsenv** is in the /usr/mqm/java/bin directory and, on HP-UX, Linux, and Solaris, it is in the /opt/mqm/java/bin directory.

On i5/OS, the environment variable QIBM_MULTI_THREADED must be set to `Y`. You can then run multithreaded applications in the same way that you run single threaded applications.

The WebSphere MQ Java libraries

To specify the location of the Java Native Interface (JNI) libraries, start your application using a **java** command with the following format:

```
java -Djava.library.path=library_path application_name
```

where *library_path* is the path to the WebSphere MQ Java libraries, which include the JNI libraries. Table 6 shows the location of the WebSphere MQ Java libraries for each platform.

Table 6. The location of the WebSphere MQ Java libraries for each platform

Platform	Directory containing the WebSphere MQ Java libraries
AIX	/usr/mqm/java/lib (32-bit libraries) /usr/mqm/java/lib64 (64-bit libraries)
HP-UX Linux (POWER, x86-64 and zSeries s390x platforms) Solaris (x86-64 and Sparc platforms)	/opt/mqm/java/lib (32-bit libraries) /opt/mqm/java/lib64 (64-bit libraries)
Linux (x86 platform) Linux (zSeries platform)	/opt/mqm/java/lib
Windows	*install_dir*\Java\lib
z/OS	*install_dir*/mqm/V6R0M0/java/lib (31-bit and 64-bit libraries)
Note: *install_dir* is the directory in which you installed WebSphere MQ. On Windows, this directory is normally C:\Program Files\IBM\WebSphere MQ. On z/OS, this directory is likely to be /usr/lpp.	

Note:

1. On AIX, HP-UX, Linux (POWER platform), or Solaris, use either the 32-bit libraries or the 64-bit libraries. Use the 64-bit libraries only if you are running your application in a 64-bit Java virtual machine (JVM) on a 64-bit platform. Otherwise, use the 32-bit libraries.
2. On Windows, you can use the PATH environment variable to specify the location of the WebSphere MQ Java libraries instead of specifying their location on the java command.
3. To use WebSphere MQ Java in bindings mode on i5/OS, ensure that the library QMQMJAVA is in your library list.

4. On z/OS, you can use either a 31-bit or 64-bit Java virtual machine (JVM) when running applications in WebSphere Application Server. In other environments on z/OS, you can use only a 31-bit JVM. You do not have to specify which libraries to use and you do not need to modify the system path to use 64-bit support.

STEPLIB configuration on z/OS

On z/OS, the STEPLIB used at runtime must contain the WebSphere MQ SCSQAUTH library. From UNIX System Services, you can add this using a line in your `.profile` as shown below, replacing `thlqual` with the high level data set qualifier that you chose when installing WebSphere MQ:

```
export STEPLIB=thlqual.SCSQAUTH:$STEPLIB
```

In other environments, you typically need to edit the startup JCL to include SCSQAUTH on the STEPLIB concatenation:

```
STEPLIB DD DSN=thlqual.SCSQAUTH,DISP=SHR
```

Running WebSphere MQ Java applications under the Java 2 Security Manager

WebSphere MQ Java can run with the Java 2 Security Manager enabled. To successfully run applications with the Security Manager enabled, you must configure your JVM with a suitable policy definition file.

The simplest way to do this is to change the policy file supplied with the JRE. On most systems this file is stored in the path `lib/security/java.policy`, relative to your JRE directory. You can edit policy files using your preferred editor or the `policytool` program supplied with your JRE.

You need to give authority to the com.ibm.mq.jar and com.ibm.mqjms.jar files so that they can:

- Create sockets (in client mode)
- Load the native library (in bindings mode)
- Read various properties from the environment

The system property os.name must be available to the WebSphere MQ Java classes when running under the Java 2 Security Manager.

Here is an example of a policy file entry that allows WebSphere MQ Java to run successfully under the default security manager. Replace the string `/opt/mqm` in this example with the location where WebSphere MQ Java is installed on your system.

```
grant codeBase "file:/opt/mqm/java/lib/com.ibm.mq.jar" {
  permission java.net.SocketPermission "*","connect";
  permission java.lang.RuntimePermission "loadLibrary.*";
};

grant codeBase "file:/opt/mqm/java/lib/com.ibm.mqjms.jar" {
  permission java.util.PropertyPermission "MQJMS_LOG_DIR","read";
  permission java.util.PropertyPermission "MQJMS_TRACE_LEVEL","read";
  permission java.util.PropertyPermission "MQJMS_TRACE_DIR","read";
  permission java.util.PropertyPermission "MQ_JAVA_INSTALL_PATH","read";
  permission java.util.PropertyPermission "file.separator","read";
  permission java.util.PropertyPermission "os.name","read";
```

```
  permission java.util.PropertyPermission "user.name","read";
  permission java.util.PropertyPermission "com.ibm.mq.jms.cleanup","read";
  permission java.util.PropertyPermission "com.ibm.mq.localaddress","read";
};
```

This example of a policy file enables the WebSphere MQ Java classes to work correctly under the security manager, but you might still need to enable your own code to run correctly before your applications will work.

The sample code shipped with WebSphere MQ Java has not been specifically enabled for use with the security manager; however the IVT tests run with the above policy file and the default security manager in place.

Running WebSphere MQ base Java applications under CICS Transaction Server

To run a WebSphere MQ base Java application as a transaction under CICS Transaction Server for OS/390 or CICS Transaction Server for z/OS, you must:

1. Define the application and transaction to CICS by using the supplied CEDA transaction.
2. Ensure that the WebSphere MQ CICS adapter is installed in your CICS system. (See *WebSphere MQ for z/OS System Setup Guide* for details.)
3. Ensure that the JVM environment specified in CICS includes the appropriate CLASSPATH and LIBPATH entries.
4. Initiate the transaction by using any of your normal processes.

For more information on running CICS Java transactions, refer to your CICS system documentation.

Chapter 3. Using WebSphere MQ classes for Java (WebSphere MQ base Java)

This chapter tells you how to:

- Configure your system to run the sample applications to verify your WebSphere MQ base Java installation.
- Modify the procedures to run your own applications.

The procedures depend on the connection option you want to use. Follow the instructions in the section that is appropriate for your requirements.

Remember to check the WebSphere MQ README file for later or more specific information for your environment.

Before attempting to run any WebSphere MQ base Java applications in bindings mode, make sure that you have configured WebSphere MQ as described in the Quick Beginnings book for your platform or the *WebSphere MQ for z/OS System Setup Guide*.

Configuring your queue manager to accept client connections

Use the following procedures to configure your queue manager to accept incoming connection requests from the clients.

TCP/IP client

1. Define a server connection channel using the following procedures:

   For i5/OS:

   a. Start your queue manager by using the STRMQM command.

   b. Define a sample channel called JAVA.CHANNEL by issuing the following command:

   ```
   CRTMQMCHL CHLNAME(JAVA.CHANNEL) CHLTYPE(*SVRCN) MQMNAME(QMGRNAME)
             MCAUSERID(SOMEUSERID)
             TEXT('Sample channel for WebSphere MQ classes for Java')
   ```

   where `QMGRNAME` is the name of your queue manager, and `SOMEUSERID` is an i5/OS user ID with appropriate authority to the WebSphere MQ resources.

   For z/OS:

   Note: You must have the Client attachment feature installed on your target queue manager in order to connect using TCP/IP.

   a. Start your queue manager by using the START QMGR command.

   b. Define a sample channel called JAVA.CHANNEL by issuing the following command:

   ```
   DEF CHL('JAVA.CHANNEL') CHLTYPE(SVRCONN) TRPTYPE(TCP)
   DESCR('Sample channel for WebSphere MQ classes for Java')
   ```

   For the other platforms:

   a. Start your queue manager by using the `strmqm` command.

b. Type the following command to start the runmqsc program:

```
runmqsc [QMNAME]
```

c. Define a sample channel called JAVA.CHANNEL by issuing the following command:

```
DEF CHL('JAVA.CHANNEL') CHLTYPE(SVRCONN) TRPTYPE(TCP) MCAUSER(' ') +
DESCR('Sample channel for WebSphere MQ classes for Java')
```

2. Start a listener program with the following commands:

For UNIX and Windows systems:

Issue the command:

```
runmqlsr -t tcp [-m QMNAME] -p 1414
```

Note: If you use the default queue manager, you can omit the -m option.

For i5/OS:

Issue the command:

```
STRMQMLSR MQMNAME(QMGRNAME)
```

where QMGRNAME is the name of your queue manager.

For z/OS:

a. Ensure your channel initiator is started. If not, start it by issuing the START CHINIT command.

b. Start the listener by issuing the command START LISTENER TRPTYPE(TCP) PORT(1414)

Verifying with the sample application

An installation verification program, MQIVP, is supplied with WebSphere MQ base Java. You can use this program to test all the connection modes of WebSphere MQ base Java. The program prompts for a number of choices and other data to determine which connection mode you want to verify. Use the following procedure to verify your installation:

1. If you are going to run the program in client mode, configure your queue manager as described in "Configuring your queue manager to accept client connections" on page 13.
2. The user ID associated with the program when it runs must have authority to access certain resources of the queue manager. Grant the following authorities to the user ID:
   - The authority to connect to the queue manager, and the authority to inquire on the attributes of the queue manager object
   - The authority to put messages on the queue SYSTEM.DEFAULT.LOCAL.QUEUE, and the authority to get messages from the queue

   For information about how to grant authorities, see the following books:
   - *WebSphere MQ for iSeries System Administration Guide*, if the queue manager is running on i5/OS
   - *WebSphere MQ System Administration Guide*, if the queue manager is running on a UNIX system or Windows
   - *WebSphere MQ for z/OS System Setup Guide*, if the queue manager is running on z/OS

If you are going to run the program in client mode, see also *WebSphere MQ Clients*.

Perform the remaining steps of this procedure on the system on which you are going to run the program.

3. Make sure that you have updated your CLASSPATH environment variable according to the instructions in "Environment variables" on page 8.
4. At a command prompt, enter:

   ```
   java -Djava.library.path=library_path MQIVP
   ```

   where *library_path* is the path to the WebSphere MQ Java libraries (see "The WebSphere MQ Java libraries" on page 10).

   The program tries to:

   a. Connect to the queue manager
   b. Open the queue SYSTEM.DEFAULT.LOCAL.QUEUE, put a message on the queue, get a message from the queue, and then close the queue
   c. Disconnect from the queue manager
   d. Return a message if the operations are successful
5. At the prompt marked (§):
   - To use a TCP/IP connection, enter a WebSphere MQ server host name.
   - To use native connection (bindings mode), leave the field blank. (Do not enter a name.)

Here is an example of the prompts and responses you might see. The actual prompts and your responses depend on your WebSphere MQ network.

```
Please enter the IP address of the MQ server              : ipaddress(§)
Please enter the port to connect to                       : (1414)(§§)
Please enter the server connection channel name            : channelname(§§)
Please enter the queue manager name                       : qmname
Success: Connected to queue manager.
Success: Opened SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Put a message to SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Got a message from SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Closed SYSTEM.DEFAULT.LOCAL.QUEUE
Success: Disconnected from queue manager


Tests complete -
SUCCESS: This MQ Transport is functioning correctly.
Press Enter to continue ...
```

Note:

1. On z/OS, leave the field blank at prompt marked (§).
2. If you choose server connection, you do not see the prompts marked (§§).
3. On i5/OS, you can issue the command `java MQIVP` only from QShell. Alternatively, you can run the application by using the CL command `RUNJVA CLASS(MQIVP)`.

Solving WebSphere MQ base Java problems

If a program does not complete successfully, run the installation verification program, and follow the advice given in the diagnostic messages. Both of these programs are described in Chapter 3, "Using WebSphere MQ classes for Java (WebSphere MQ base Java)," on page 13.

If the problems continue and you need to contact the IBM service team, you might be asked to turn on the trace facility. Refer to the following sections for the appropriate procedures for your system.

Tracing the sample application

To trace the `MQIVP` program, enter the following:

```
java -Djava.library.path=library_path MQIVP -trace n
```

where *library_path* is the path to the WebSphere MQ Java libraries (see "The WebSphere MQ Java libraries" on page 10), and `n` is a number between 1 and 5, depending on the level of detail required (the greater the number, the more information that is gathered).

For more information about how to use trace, see "Tracing WebSphere MQ base Java programs" on page 95.

Error messages

Here are some of the more common error messages that you might see:

Unable to identify local host IP address
The server is not connected to the network.

Connect the server to the network and retry.

MQRC_ADAPTER_CONN_LOAD_ERROR
If you see this z/OS error , ensure that the WebSphere MQ SCSQANLE and SCSQAUTH datasets are in your STEPLIB statement.

Chapter 4. Using WebSphere MQ classes for Java Message Service (WebSphere MQ JMS)

This chapter tells you how to:

- Set up and use JMS Postcard
- Set up your system to use the test and sample programs
- Run the point-to-point Installation Verification Test (IVT) program to verify your WebSphere MQ classes for Java Message Service installation
- Run the sample publish/subscribe Installation Verification Test (PSIVT) program to verify your publish/subscribe installation
- Run your own programs

Before attempting to run any WebSphere MQ JMS applications in bindings mode, make sure that you have configured WebSphere MQ as described in the Quick Beginnings book for your platform or the *WebSphere MQ for z/OS System Setup Guide*.

JMS Postcard

JMS Postcard is a simple way to do the following:

- Verify that you have successfully installed WebSphere MQ and WebSphere MQ JMS on one computer and, optionally, on others as well
- Introduce you to messaging

Note: JMS Postcard is ***not*** supported on WebSphere MQ for z/OS or WebSphere MQ for iSeries.

Setting up JMS Postcard

To use JMS Postcard, make sure that the Java Messaging feature of WebSphere MQ for Windows (WebSphere MQ JMS) is installed. You also need a working Java Runtime Environment (JRE) at Java 1.3.1 level or later.

Before you can successfully run the JMS Postcard application, define the environment variables CLASSPATH, MQ_JAVA_DATA_PATH, MQ_JAVA_INSTALL_PATH, and MQ_JAVA_LIB_PATH. On Windows systems these variables are set as part of the install process. On other platforms you must set them yourself. For more information about these variables, see “Environment variables” on page 8.

Many operations that the Postcard application carries out on your behalf require the user to be a member of the WebSphere MQ administrators group (mqm). If you are not a member of mqm, get a member of the mqm group to set up the *default configuration* on your behalf. See “JMS Postcard default configuration” on page 20.

Starting

To start the JMS Postcard application, run the `postcard` script. This is supplied in the `java/bin` directory of the WebSphere MQ installation.

The first time that you run JMS Postcard, it asks you to complete the default configuration, which sets up a suitable queue manager to act as mailbox. See "JMS Postcard default configuration" on page 20.

Whenever you start a Postcard application, you must sign on and enter a nickname. (There are advanced options available on the sign-on dialog, see "Sign-on advanced options" for details).

Sign-on

The sign-on dialog has a check box labelled Advanced. Check this to see the extended dialog where you can choose which queue manager is used by the Postcard program.

Note:

1. If you have no queue managers at all, or just the default configuration, the checkbox is disabled.
2. Depending on what queue managers and clusters you have, the checkbox and options are in one of various combinations of enabled, disabled, and preselected.

Sign-on advanced options

Use default configuration as mailbox

This is the easiest way to use JMS Postcard on one or several computers. Make sure that the default configuration is installed on all the computers, that one of them holds the repository, and that all the others use the first one as their repository; this puts them all in the same cluster.

Choose queue manager as mailbox

Use the drop-down list to choose any one of your local queue managers. If you want to send postcards between two queue managers (on one or more computers) this way, make sure that one of the following conditions is true:

- The queue managers are in the same cluster (for more information about clusters, see the *WebSphere MQ Queue Manager Clusters* book).
- There are explicit connections between the queue managers.

Sending a postcard

To send a postcard successfully, you need two instances of the Postcard application with different nicknames. For example, suppose you start the Postcard application and use the nickname Will, and then start it again using the nickname Tim. Will can send postcards to Tim and Tim can send postcards to Will.

If Will and Tim are connected to the same queue manager, see "Running JMS Postcard with one queue manager" on page 19.

If Tim is on a different queue manager manager (on the same or a different computer from Will), see "Running JMS Postcard with two queue managers" on page 19.

When the postcard arrives successfully, you know that your WebSphere MQ installation and WebSphere MQ JMS are working correctly.

For an alternative way of verifying the installation of WebSphere MQ JMS, run the IVTRun application from the command line. See "Running the point-to-point IVT" on page 26 for more information about this.

Running JMS Postcard with one queue manager

If you have already started the Postcard application with a nickname, for example, Will, and you want to send a postcard to a second nickname on this computer, follow these steps:

1. Move the first Postcard (Will) to one side of your screen, then start a second Postcard by running the `postcard` shell script again.
2. Enter your second nickname, for example Tim.
3. On Will's Postcard fill in the **To** field with your second nickname, Tim. (You can leave the **On** field empty and Postcard will fill it in for you, or you can type in the queue manager name that you see below the **Message** box after **On**).
4. Click in the **Message** box, type your message in, and click the **Send** button.
5. Look in Tim's Postcard to see the message arrive, and double-click on it to see the postcard itself.
6. Try using Tim to send a message back to Will. You can do this by selecting the message that arrived in Tim's list, and clicking the **Reply** button.

Note: See "JMS Postcard configuration" on page 20 for advice about configuration.

Running JMS Postcard with two queue managers

If you have already started JMS Postcard with a nickname, for example Will, and you want to send a postcard to a second nickname on a second queue manager on this, or another, computer, follow these steps:

1. Start the second Postcard, choosing one of the following:
   - JMS Postcard
     - On this computer, run the **postcard** shell script again, then in the sign-on dialog check Advanced and select the second queue manager you want to use.
     - On another computer, run the **postcard** shell script; or, on Windows systems, open WebSphere MQ First Steps and click on JMS Postcard.
   - MQI Postcard on Windows systems:

     either start from WebSphere MQ First Steps (to use the default configuration), or open the WebSphere MQ Explorer, right-click on the queue manager you want to use and click **All Tasks->Start a Postcard...**
2. When the sign-on dialog appears, enter your second nickname (for example, Tim).
3. In the Postcard application on Will's computer, fill in the **To** field with your second nickname (Tim), and in the **On** field put the queue manager name of the second postcard where Tim is. If you don't know this name, on Tim's computer in the Postcard look below the Message box after **On:**; alternatively if both queue managers are in the default configuration cluster, you can just type in the short TCP/IP name of Tim's computer and Postcard builds that into the queue manager name in the same way that the task that creates the default configuration does.
4. Type your message, and click **Send**.

   Look in Tim's Postcard to see the message arrive, and double-click on it to see the postcard itself.
5. Try sending a message from Tim's computer back to Will.

You can do this by selecting the message that arrived in Tim's list, and clicking **Reply**.

Note: See "JMS Postcard configuration."

See also "How JMS Postcard works."

JMS Postcard configuration

The Postcard application needs a suitable queue manager to act as mailbox. See "JMS Postcard default configuration" for the easiest way to get one. You will be prompted to install this default configuration the first time you start the Postcard application (see "Starting" on page 17).

Instead of using the default configuration, you can also start the Postcard application using any other local queue manager.

If you want to send postcards to another computer, or to other queue managers, the default configuration must include the option of being joined in the same cluster. The other queue managers must either be in the same cluster or you must create a connection explicitly between them.

See also "How JMS Postcard works."

JMS Postcard default configuration

Installing the default configuration creates a special queue manager (with queues and channels), and optionally joins it to a cluster, to enable you to use the JMS Postcard application to verify your installation and see messaging working.

On WebSphere MQ for Windows, the Default Configuration Wizard automatically opens when JMS Postcard is started and the wizard has not already been run on this computer.

On platforms other than Windows systems, you can also run the `DefaultConfiguration` script, provided that there are no existing queue managers on this computer. On Windows systems, run Default Configuration from First Steps.

Note: You must be a member of the WebSphere MQ administrators group (mqm) to complete default configuration successfully. If you are not a member of mqm, get a member of the mqm group to set up the default configuration on your behalf.

How JMS Postcard works

This section tells you how the JMS Postcard works, including:

Starting up

When JMS Postcard starts, it checks to see what queue managers exist on this computer, and initializes the sign-on dialog accordingly. If there are no queue managers at all, it prompts you to install the default configuration.

JMS Postcard uses the Java Message Service method queueConnectionFactory.createQueueConnection() to connect to the default queue manager.

Receiving messages

All the time JMS Postcard is running, it polls a queue called postcard for incoming messages from other Postcard applications. If there is no queue called postcard, JMS Postcard creates one.

When JMS Postcard starts running, it creates a Java Message Service QueueReceiver object for the local postcard queue, providing as a parameter a selector string that filters the messages to be received from the queue by the Correlation Identifier (CorrelId field). The selector string defines that the postcard client should only receive messages where the CorrelId field matches the nickname of the user. The words from the message data are then presented in the JMS Postcard window.

Sending messages

If you did not enter a computer name in the **On:** field, JMS Postcard assumes that the recipient is on the same queue manager.

If you entered a name, JMS Postcard checks for the existence of a queue manager with this name, first using the exact name supplied, and then using a prefix in the same format as that created by the default configuration.

In both cases, it issues a session.createQueue('postcard'), and sets the base queue manager name to the string supplied.

Finally, it builds a JMS BytesMessage from your nickname and the words you typed in, and runs queueSender.send(theMessage) to put the message onto the queue.

How the postcards get there

When other instances of Postcard on this computer use the same queue manager and queue, the messages are being put and got from the one queue. This does, however, verify that the WebSphere MQ code installed on this computer is configured and working correctly.

JMS Postcard can only send to another queue manager if a connection to that queue manager exists. This connection exists because either both queue managers are members of the same cluster, or you have explicitly created a connection yourself. JMS Postcard can therefore assume that it can connect to the queue manager, and connects to it, opens the queue, and puts a message, as already described, leaving all the work of getting the message there to the WebSphere MQ cluster code. In other words, JMS Postcard uses only one piece of code for putting the message, and does not need to know whether the message is going to another computer.

In JMS Postcard, when session.createSender('postcard') is called, the cluster code checks the repository to find the other queue manager, and to check that the queue exists, and throws an exception if this was not possible for any reason.

When queueSender.send(theMessage) is called, the cluster code opens a channel to the other queue manager (creating it if necessary) and sends the message.

Discard the channel afterwards, if the cluster optimizing code does not need it. If the queue managers are on different computers, that is all handled by the cluster code.

Tidying up undeliverable messages

If you sent a postcard message to John, but never ran a Postcard application with the nickname John, the message would sit on the queue for ever. To prevent this, JMS Postcard sets the Message Lifetime (Expiry) field in the Message Descriptor (MQMD) to 48 hours. After that time, the message is discarded, wherever it may be (possibly even still in transmission).

Exchanging messages between different WebSphere MQ Postcard applications

You can exchange messages between all the different types of Postcard application as follows:

- **MQI Postcard** on WebSphere MQ for Windows.
- **JMS Postcard** on Windows systems and other operating systems such as UNIX.
- **MQSeries® Postcard** on previous versions of MQSeries for Windows, with the exception that it cannot *receive* messages from JMS Postcard.
- **MQ Everyplace® Postcard** on WebSphere MQ Everyplace on pervasive devices. For this, a connection must be explicitly set up between the queue managers. See the WebSphere MQ Everyplace product documentation for further information.

Customizing JMS Postcard

Normally JMS Postcard uses standard Java Swing settings for font size and background color. But if it detects a `postcard.ini` file on startup, JMS Postcard uses settings specified in this file instead. You can also change the trace setting.

Edit the sample file `postcard.ini` in the `bin` directory of the WebSphere MQ classes for Java installation and set your preferred settings for font size, and screen foreground and background colors.

Note: The precise use of upper and lower case letters in the keywords, as in the following examples, must be strictly observed when you set these properties.

Setting screen colors

By setting the Background and Foreground properties, you can change the background and foreground colors of controls used in the Postcard application.

```
Background=000000
Foreground=FFFFFF
```

This example selects white text on a black background. The values represent intensity levels for red, green, and blue colors using a hexadecimal scale from 00 to FF. Other examples of colors are FF0000 (bright red), 00FF00 (bright green) and 0000FF (bright blue).

Setting font size

```
MinimumFont=20
```

This example selects a minimum font size of 20 points. Any value smaller than 13 is ignored.

Using an external browser for online help
`WebBrowser=nautilus`

This setting is only applicable on non-Windows systems. The internal browser used for displaying online help information cannot be customized. This setting allows you to identify an alternative browser.

Tracing the Postcard application
`Trace=1`

Set this to start trace output. Note that the trace output is sent to the `trc` subdirectory of the directory defined by the MQ_JAVA_DATA_PATH system environment variable. If the application cannot write to this directory, trace output is directed to the system console.

You can also use the MQJMS_TRACE_LEVEL parameter on the `java` command line to start tracing. See "Tracing programs" on page 33 for more about tracing applications.

Post installation setup

Note: Remember to check the WebSphere MQ README file. It might contain information that supersedes the information in this book.

After installation, on any platform other than Windows, you must update your environment variables as described in "Environment variables" on page 8.

Additional setup for publish/subscribe mode

Before you can use the WebSphere MQ JMS implementation of JMS publish/subscribe, some additional setup is required:

- Ensure that you have access to a publish/subscribe broker.
- Ensure that the broker is running.
- Create the WebSphere MQ JMS system queues.

  This step is not required for direct connection to a broker.

You also need to know publish/subscribe concepts as discussed in Chapter 11, "Writing WebSphere MQ JMS publish/subscribe applications," on page 327.

Ensure that you have access to a publish/subscribe broker
With WebSphere MQ JMS, you have the choice of these brokers:

- WebSphere MQ Publish/Subscribe
- WebSphere MQ Integrator, Version 2
- WebSphere MQ Event Broker, Version 2.1
- WebSphere Business Integration Event Broker, Version 5.0
- WebSphere Business Integration Message Broker, Version 5.0

Differences between these brokers are discussed in Chapter 11, "Writing WebSphere MQ JMS publish/subscribe applications," on page 327. Read the documentation for each broker for installation and configuration instructions.

Note, however, that broker based subscription stores are not supported by WebSphere MQ Integrator, Version 2. For more information about subscription stores, see "Subscription stores" on page 363.

Ensure that the broker is running

WebSphere MQ Publish/Subscribe

To verify that the broker is installed and running, use the command:

```
dspmqbrk -m MY.QUEUE.MANAGER
```

where `MY.QUEUE.MANAGER` is the name of the queue manager on which the broker is running. If the broker is running, a message similar to the following is displayed:

```
WebSphere MQ message broker for queue manager MY.QUEUE.MANAGER running.
```

If the operating system reports that it cannot run the `dspmqbrk` command, ensure that the WebSphere MQ Publish/Subscribe broker is installed properly.

If the operating system reports that the broker is not active, start it using the command:

```
strmqbrk -m MY.QUEUE.MANAGER
```

WebSphere MQ Integrator, Version 2, WebSphere Business Integration Event Broker, Version 5.0, or WebSphere Business Integration Message Broker, Version 5.0

To verify that the broker is installed and running, refer to the product documentation.

The command to start the broker is:

```
mqsistart MYBROKER
```

where `MYBROKER` is the name of the broker.

WebSphere MQ Event Broker, Version 2.1

To verify that the broker provided in WebSphere MQ Event Broker is installed and running, refer to the product documentation.

The command to start the broker in WebSphere MQ Event Broker is:

```
wmqpsstart MYBROKER
```

where `MYBROKER` is the name of the broker.

Create the WebSphere MQ JMS system queues

This does not apply if you use a direct connection to a broker.

For a publish/subscribe implementation to work correctly, you must create a number of system queues. A script is supplied, in the `bin` subdirectory of the WebSphere MQ JMS installation, to assist with this task. To use the script, enter the following commands:

For i5/OS:

1. Copy the script from the integrated file system to a native file system library using a command similar to:

   ```
   CPYFRMSTMF FROMSTMF('/QIBM/ProdData/mqm/java/bin/MQJMS_PSQ.mqsc')
              TOMBR('/QSYS.LIB/QGPL.LIB/QCLSRC.FILE/MQJMS_PSQ.MBR')
   ```

2. Call the script file using STRMQMMQSC:

   ```
   STRMQMMQSC SRCMBR(MQJMS_PSQ) SRCFILE(QGPL/QCLSRC)
   ```

For z/OS:

1. Copy the script from the HFS into a PDS using a TSO command similar to:

   ```
   OGET '/usr/lpp/mqm/java/bin/MQJMS_PSQ.mqsc' 'USERID.MQSC(MQJMSPSQ)'
   ```

   The PDS should be of fixed-block format with a record length of 80.
2. Either use the CSQUTIL application to execute this command script, or add the script to the CSQINP2 DD concatenation in your queue manager's started task JCL. In either case, refer to the *WebSphere MQ for z/OS System Setup Guide* and the *WebSphere MQ for z/OS System Administration Guide* for further details.

For the other platforms:

```
runmqsc MY.QUEUE.MANAGER < MQJMS_PSQ.mqsc
```

If an error occurs, check that you typed the queue manager name correctly and that the queue manager is running.

For a broker running on a remote queue manager

For operation with a broker running on a remote queue manager, further setup is required.

1. Define a transmission queue on the remote queue manager with a queue name matching the local queue manager. These names must match for correct routing of messages by WebSphere MQ.
2. Define a sender channel on the remote queue manager and a receiver channel on the local queue manager. The sender channel should use the transmission queue defined in step 1.
3. Set up the local queue manager for communication with the remote broker:
   a. Define a local transmission queue with the same name as the queue manager running the remote broker.
   b. Define local sender and remote receiver channels to the remote broker queue manager. The sender channel must use the transmission queue defined in step 3a.
4. To operate the remote broker, take the following steps:
   a. Start the remote broker queue manager.
   b. Start a listener for the remote broker queue manager (TCP/IP channels).
   c. Start the sender and receiver channels to the local queue manager.
   d. Start the broker on the remote queue manager.

      An example command is

      ```
      strmqbrk -m MyBrokerMgr
      ```
5. To operate the local queue manager to communicate with the remote broker, take the following steps:
   a. Start the local queue manager.
   b. Start a listener for the local queue manager.
   c. Start the sender and receiver channels to the remote broker queue manager.

Queues that require authorization for non-privileged users

Non-privileged users need authorization granted to access the queues used by JMS. For details about access control in WebSphere MQ, see the chapter about protecting WebSphere MQ objects in the *WebSphere MQ System Administration Guide*.

For JMS point-to-point mode, the access control issues are similar to those for the WebSphere MQ classes for Java:

- Queues that are used by QueueSender need put authority.
- Queues that are used by QueueReceivers and QueueBrowsers need get, inq, and browse authorities.
- The QueueSession.createTemporaryQueue method needs access to the model queue that is defined in the QueueConnectionFactory temporaryModel field (by default this is SYSTEM.DEFAULT.MODEL.QUEUE).

For JMS publish/subscribe mode, the following system queues are used:

- SYSTEM.JMS.ADMIN.QUEUE
- SYSTEM.JMS.REPORT.QUEUE
- SYSTEM.JMS.MODEL.QUEUE
- SYSTEM.JMS.PS.STATUS.QUEUE
- SYSTEM.JMS.ND.SUBSCRIBER.QUEUE
- SYSTEM.JMS.D.SUBSCRIBER.QUEUE
- SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE
- SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE
- SYSTEM.BROKER.CONTROL.QUEUE

Also, any application that publishes messages needs access to the STREAM queue that is specified in the topic connection factory being used. The default value for this is SYSTEM.BROKER.DEFAULT.STREAM.

If you use ConnectionConsumer, additional authorization might be needed. Queues to be read by the ConnectionConsumer must have get, inq and browse authorities. The system dead-letter queue, and any backout-requeue queue or report queue used by the ConnectionConsumer must have put and passall authorities.

Running the point-to-point IVT

This section describes the point-to-point installation verification test program (IVT) that is supplied with WebSphere MQ JMS.

The IVT verifies the installation by connecting to the default queue manager on the local machine, using the WebSphere MQ JMS in bindings mode. It then sends a message to the `SYSTEM.DEFAULT.LOCAL.QUEUE` queue and reads it back again.

You can run the program in one of two possible modes.

With JNDI lookup of administered objects
JNDI mode forces the program to obtain its administered objects from a JNDI namespace, which is the expected operation of JMS client applications. (See "Administering JMS objects" on page 39 for a description of administered objects). This invocation method has the same prerequisites as the administration tool (see Chapter 5, "Using the WebSphere MQ JMS administration tool," on page 35).

Without JNDI lookup of administered objects
If you do not want to use JNDI, you can create the administered objects at runtime by running the IVT in non-JNDI mode. Because a JNDI-based repository is relatively complex to set up, run the IVT first without JNDI.

Point-to-point verification without JNDI

A script, named `IVTRun` on UNIX, or `IVTRun.bat` on Windows systems, is provided to run the IVT. This file is installed in the `bin` subdirectory of the installation.

To run the test without JNDI, issue the following command:

```
IVTRun [-t] -nojndi [-m <qmgr>]
```

For client mode, to run the test without JNDI, issue the following command:

```
IVTRun [-t] -nojndi -client -m <qmgr> -host <hostname> [-port <port>]
            [-channel <channel>]
```

where:

-t turns tracing on (by default, tracing is off)

qmgr is the name of the queue manager to which you want to connect

hostname
is the host on which the queue manager is running

port is the TCP/IP port on which the queue manager's listener is running (default 1414)

channel
is the client connection channel (default SYSTEM.DEF.SVRCONN)

If the test completes successfully, you should see output similar to the following:

```
5724-H72, 5655-L82, 5724-L26 (c) Copyright IBM Corp. 2002,2005. All Rights Reserved.
Websphere MQ classes for Java(tm) Message Service 6.0
Installation Verification Test


Creating a QueueConnectionFactory
Creating a Connection
Creating a Session
Creating a Queue
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again

Got message:
JMS Message class: jms_text
  JMSType:         null
  JMSDeliveryMode: 2
  JMSExpiration:   0
  JMSPriority:     4
  JMSMessageID:    ID:414d5120514d5f63616c6c616e6169734286ac4120000a02
  JMSTimestamp:    1101826963911
  JMSCorrelationID:null
  JMSDestination:  queue:///SYSTEM.DEFAULT.LOCAL.QUEUE
  JMSReplyTo:      null
  JMSRedelivered:  false
  JMS_IBM_PutDate:20041130
  JMSXAppID:WebSphere MQ Client for Java
  JMS_IBM_Format:MQSTR
  JMS_IBM_PutApplType:28
  JMS_IBM_MsgType:8
  JMSXUserID:mwhite
  JMS_IBM_PutTime:15024393
  JMSXDeliveryCount:1
A simple text message from the MQJMSIVT program
Reply string equals original string
```

```
Closing QueueReceiver
Closing QueueSender
Closing Session
Closing Connection
IVT completed OK
IVT finished
```

Point-to-point verification with JNDI

To run the IVT with JNDI, the LDAP server must be running and must be configured to accept Java objects. If the following message occurs, it indicates that there is a connection to the LDAP server, but that the server is not correctly configured:

```
Unable to bind to object
```

This message means that either the server is not storing Java objects, or the permissions on the objects or the suffix are not correct. For more help in this situation, see the documentation for your LDAP server.

Also, the following administered objects must be retrievable from a JNDI namespace:

- MQQueueConnectionFactory
- MQQueue

A script, named `IVTSetup` on UNIX, or `IVTSetup.bat` on Windows systems, is provided to create these objects automatically. Enter the command:

```
IVTSetup
```

The script invokes the WebSphere MQ JMS Administration tool (see Chapter 5, “Using the WebSphere MQ JMS administration tool,” on page 35) and creates the objects in a JNDI namespace.

The MQQueueConnectionFactory is bound under the name `ivtQCF` (for LDAP, `cn=ivtQCF`). All the properties are default values:

```
TRANSPORT(BIND)
PORT(1414)
HOSTNAME(localhost)
CHANNEL(SYSTEM.DEF.SVRCONN)
VERSION(1)
CCSID(819)
TEMPMODEL(SYSTEM.DEFAULT.MODEL.QUEUE)
QMANAGER()
```

The MQQueue is bound under the name `ivtQ` (`cn=ivtQ`). The value of the `QUEUE` property becomes `QUEUE(SYSTEM.DEFAULT.LOCAL.QUEUE)`. All other properties have default values:

```
PERSISTENCE(APP)
QUEUE(SYSTEM.DEFAULT.LOCAL.QUEUE)
EXPIRY(APP)
TARGCLIENT(JMS)
ENCODING(NATIVE)
VERSION(1)
CCSID(1208)
PRIORITY(APP)
QMANAGER()
```

Once the administered objects are created in the JNDI namespace, run the `IVTRun` (`IVTRun.bat` on Windows systems) script using the following command:

```
IVTRun [ -t ]  -url "<providerURL>" [ -icf <initCtxFact> ]
```

where:

-t turns tracing on (by default, tracing is off)

providerURL

Note: Enclose the *providerURL* string in quotation marks (").
This is the JNDI location of the administered objects. If the default initial context factory is in use, this is an LDAP URL of the form:

```
"ldap://hostname.company.com/contextName"
```

If a file system service provider is used, (see `initCtxFact` below), the URL is of the form:

```
"file://directorySpec"
```

initCtxFact

is the classname of the initial context factory. The default is for an LDAP service provider, and has the value:

```
com.sun.jndi.ldap.LdapCtxFactory
```

If a file system service provider is used, set this parameter to:

```
com.sun.jndi.fscontext.RefFSContextFactory
```

If the test completes successfully, the output is similar to the non-JNDI output, except that the create `QueueConnectionFactory` and `Queue` lines indicate retrieval of the object from JNDI. The following shows an example.

```
5724-H72, 5655-L82, 5724-L26 (c) Copyright IBM Corp. 2002,2005. All Rights Reserved.
Websphere MQ classes for Java(tm) Message Service 6.0
Installation Verification Test

Using administered objects, please ensure that these are available

Retrieving a QueueConnectionFactory from JNDI
Creating a Connection
Creating a Session
Retrieving a Queue from JNDI
Creating a QueueSender
Creating a QueueReceiver
Creating a TextMessage
Sending the message to SYSTEM.DEFAULT.LOCAL.QUEUE
Reading the message back again

Got message:
JMS Message class: jms_text
  JMSType:         null
 ...
 ...
```

Although not strictly necessary, it is good practice to remove objects that are created by the IVTSetup script from the JNDI namespace. A script called `IVTTidy` (`IVTTidy.bat` on Windows systems) is provided for this purpose.

IVT error recovery

If the test is not successful, note the following:

- For help with any error messages involving the classpath, check that your classpath is set correctly, as described in "Post installation setup" on page 23.
- The IVT might fail with a message `failed to create MQQueueManager`, with an additional message including the number 2059. This indicates that WebSphere

MQ failed to connect to the default local queue manager on the machine on which you ran the IVT. Check that the queue manager is running, and that it is marked as the default queue manager.

- A message `failed to open MQ queue` indicates that WebSphere MQ connected to the default queue manager, but could not open the SYSTEM.DEFAULT.LOCAL.QUEUE. This might indicate that either the queue does not exist on your default queue manager, or that the queue is not enabled for PUT and GET. Add or enable the queue for the duration of the test.

The IVT tests whether the following Java archive (JAR) files are accessible to your Java virtual machine (JVM):

- com.ibm.mqjms.jar
- com.ibm.mq.jar
- connector.jar
- jms.jar
- jndi.jar
- jta.jar
- ldap.jar
- providerutil.jar

The publish/subscribe installation verification test

The publish/subscribe installation verification test (PSIVT) program is supplied only in compiled form. It is in the `com.ibm.mq.jms` package.

The PSIVT requires a suitable publish/subscribe broker that is running. See "Additional setup for publish/subscribe mode" on page 23 for a list of the supported publish/subscribe brokers and instructions on how to start each of them.

The PSIVT attempts to:

1. Create a publisher, `p`, publishing on the topic `MQJMS/PSIVT/Information`
2. Create a subscriber, `s`, subscribing on the topic `MQJMS/PSIVT/Information`
3. Use `p` to publish a simple text message
4. Use `s` to receive a message waiting on its input queue

When you run the PSIVT, the publisher publishes the message, and the subscriber receives and displays the message. The publisher publishes to the broker's default stream. The subscriber is non-durable, does not perform message selection, and accepts messages from local connections. It performs a synchronous receive, waiting a maximum of 5 seconds for a message to arrive.

You can run the PSIVT, like the IVT, in either JNDI mode or standalone mode. JNDI mode uses JNDI to retrieve a `TopicConnectionFactory` and a `Topic` from a JNDI namespace. If JNDI is not used, these objects are created at runtime.

Publish/subscribe verification without JNDI

A script named `PSIVTRun` (`PSIVTRun.bat` on Windows systems) is provided to run PSIVT. The file is in the `bin` subdirectory of the installation.

To run the test without JNDI, issue the following command:

```
PSIVTRun -nojndi [-m <qmgr>] [-bqm <broker>] [-t]
```

For client mode, to run the test without JNDI, issue the following command:

```
PSIVTRun -nojndi -client -m <qmgr> -host <hostname> [-port <port>]
               [-channel <channel>] [-bqm <broker>] [-t]
```

where:

-nojndi
indicates no JNDI lookup of the administered objects

qmgr is the name of the queue manager to which you wish to connect

hostname
is the host on which the queue manager is running

port is the TCP/IP port on which the queue manager's listener is running (default 1414)

channel
is the client connection channel (default SYSTEM.DEF.SVRCONN)

broker
is the name of the remote queue manager on which the broker is running. If this is not specified, the value used for **qmgr** is assumed.

-t turns tracing on (default is off)

If the test completes successfully, output is similar to the following:

```
5724-H72, 5655-L82, 5724-L26 (c) Copyright IBM Corp. 2002,2005. All Rights Reserved.
Websphere MQ classes for Java(tm) Message Service 6.0
Publish/Subscribe Installation Verification Test

Creating a TopicConnectionFactory
Creating a Connection
Creating a Session
Creating a Topic
Creating a TopicPublisher
Creating a TopicSubscriber
Creating a TextMessage
Adding text
Publishing the message to topic://MQJMS/PSIVT/Information
Waiting for a message to arrive [5 secs max]...

Got message:
JMS Message class: jms_text
  JMSType:         null
  JMSDeliveryMode: 2
  JMSExpiration:   0
  JMSPriority:     4
  JMSMessageID:    ID:414d5120514d5f63616c6c616e6169734286ac4120002304
  JMSTimestamp:    1102075124453
  JMSCorrelationID:AMQ QM_callanais
  JMSDestination:  topic://MQJMS/PSIVT/Information
  JMSReplyTo:      null
  JMSRedelivered:  false
  JMS_IBM_PutDate:20041203
  JMSXAppID:QM_callanais
  JMS_IBM_Format:MQSTR
  JMS_IBM_PutApplType:26
  JMS_IBM_MsgType:8
  JMSXUserID:mwhite
  JMS_IBM_PutTime:11584446
  JMSXDeliveryCount:1
A simple text message from the MQJMSPSIVT program
Reply string equals original string
Closing TopicSubscriber
```

```
Closing TopicPublisher
Closing Session
Closing Connection
PSIVT finished
```

Publish/subscribe verification with JNDI

To run the PSIVT in JNDI mode, two administered objects must be retrievable from a JNDI namespace:

- A TopicConnectionFactory bound under the name `ivtTCF`
- A Topic bound under the name `ivtT`

You can define these objects by using the WebSphere MQ JMS Administration Tool (see Chapter 5, "Using the WebSphere MQ JMS administration tool," on page 35) and using the following commands:

```
DEFINE TCF(ivtTCF)
```

This command defines the TopicConnectionFactory.

```
DEFINE T(ivtT) TOPIC(MQJMS/PSIVT/Information)
```

This command defines the Topic.

These definitions assume that a default queue manager, on which the broker is running, is available. For details on configuring these objects to use a non-default queue manager, see "Administering JMS objects" on page 39. These objects must reside in a context pointed to by the `-url` command-line parameter described below.

To run the test in JNDI mode, enter the following command:

```
PSIVTRun [ -t ]  -url "<providerURL>" [ -icf <initCtxFact> ]
```

where:

-t means turn tracing on (by default, tracing is off)

providerURL

Note: Enclose the *providerURL* string in quotation marks (").
This is the JNDI location of the administered objects. If the default initial context factory is in use, this is an LDAP URL of the form:

```
"ldap://hostname.company.com/contextName"
```

If a file system service provider is used, (see `initCtxFact` below), the URL is of the form:

```
"file://directorySpec"
```

initCtxFact
is the classname of the initial context factory. The default is for an LDAP service provider, and has the value:

```
com.sun.jndi.ldap.LdapCtxFactory
```

If a file system service provider is used, set this parameter to:

```
com.sun.jndi.fscontext.RefFSContextFactory
```

If the test completes successfully, output is similar to the non-JNDI output, except that the create `QueueConnectionFactory` and `Queue` lines indicate retrieval of the object from JNDI.

PSIVT error recovery

If the test is not successful, note the following:

- The following message:

  ```
  *** No broker response. Please ensure broker is running. ***
  ```

  indicates that the broker is installed on the target queue manager, but its control queue contains some outstanding messages. For instructions on how to start it, see "Additional setup for publish/subscribe mode" on page 23.
- If the following message is displayed:

  ```
  Unable to connect to queue manager: <default>
  ```

  ensure that your WebSphere MQ system has configured a default queue manager.
- If the following message is displayed:

  ```
  Unable to connect to queue manager: ...
  ```

  ensure that the administered TopicConnectionFactory that the PSIVT uses is configured with a valid queue manager name. Alternatively, if you used the `-nojndi` option, ensure that you supplied a valid queue manager (using the `-m` option).
- If the following message is displayed:

  ```
  Unable to access broker control queue on queue manager: ...
  Please ensure the broker is installed on this queue manager
  ```

  ensure that the administered TopicConnectionFactory that the PSIVT uses is configured with the name of the queue manager on which the broker is installed. If you used the `-nojndi` option, ensure that you supplied a queue manager name (using the `-m` option).

Solving problems

If a program does not complete successfully, run the installation verification program, which is described in "Running the point-to-point IVT" on page 26, and follow the advice given in the diagnostic messages.

Tracing programs

The WebSphere MQ JMS trace facility is provided to help IBM staff to diagnose customer problems.

Trace is disabled by default, because the output rapidly becomes large, and is unlikely to be of use in normal circumstances.

If you are asked to provide trace output, enable it by setting the Java property MQJMS_TRACE_LEVEL to one of the following values:

on traces WebSphere MQ JMS calls only

base traces both WebSphere MQ JMS calls and the underlying WebSphere MQ base Java calls

For example:

```
java -Djava.library.path=library_path
     -DMQJMS_TRACE_LEVEL=base MyJMSProg
```

where *library_path* is the path to the WebSphere MQ Java libraries (see "The WebSphere MQ Java libraries" on page 10).

To disable trace, set MQJMS_TRACE_LEVEL to **off**.

By default, trace is output to a file named `mqjms.trc` in the current working directory. You can redirect it to a different directory by using the Java property `MQJMS_TRACE_DIR`. For example:

```
java -Djava.library.path=library_path
     -DMQJMS_TRACE_LEVEL=base -DMQJMS_TRACE_DIR=/somepath/tracedir MyJMSProg
```

Logging

The WebSphere MQ JMS log facility is provided to report serious problems, particularly those that might indicate configuration errors rather than programming errors. By default, log output is sent to the `System.err` stream, which usually appears on the `stderr` of the console in which the JVM is run.

You can redirect the output to a file by using a Java property that specifies the new location, for example:

```
java -Djava.library.path=library_path
     -DMQJMS_LOG_DIR=/mydir/forlogs MyJMSProg
```

where *library_path* is the path to the WebSphere MQ Java libraries (see "The WebSphere MQ Java libraries" on page 10).

When the log is redirected to a file, it is output in a binary form. To view the log, the utility `formatLog` (`formatLog.bat` on Windows systems) is provided, which converts the file to plain text format. The utility is stored in the `bin` directory of your WebSphere MQ JMS installation. Run the conversion as follows:

```
formatLog <inputfile> <outputfile>
```

Chapter 5. Using the WebSphere MQ JMS administration tool

The administration tool enables administrators to define the properties of eight types of WebSphere MQ JMS object and to store them within a JNDI namespace. Then, JMS clients can use JNDI to retrieve these administered objects from the namespace and use them.

The JMS objects that you can administer by using the tool are:
- MQConnectionFactory
- MQQueueConnectionFactory
- MQTopicConnectionFactory
- MQQueue
- MQTopic
- MQXAConnectionFactory
- MQXAQueueConnectionFactory
- MQXATopicConnectionFactory

For details about these objects, refer to "Administering JMS objects" on page 39.

The tool also allows administrators to manipulate directory namespace subcontexts within the JNDI. See "Manipulating subcontexts" on page 39.

Invoking the administration tool

The administration tool has a command line interface. You can use this interactively, or use it to start a batch process. The interactive mode provides a command prompt where you can enter administration commands. In the batch mode, the command to start the tool includes the name of a file that contains an administration command script.

To start the tool in interactive mode, enter the command:

```
JMSAdmin [-t] [-v] [-cfg config_filename]
```

where:

-t Enables trace (default is trace off)

-v Produces verbose output (default is terse output)

-cfg config_filename
Names an alternative configuration file (see "Configuration" on page 36)

A command prompt is displayed, which indicates that the tool is ready to accept administration commands. This prompt initially appears as:

```
InitCtx>
```

indicating that the current context (that is, the JNDI context to which all naming and directory operations currently refer) is the initial context defined in the PROVIDER_URL configuration parameter (see "Configuration" on page 36).

As you traverse the directory namespace, the prompt changes to reflect this, so that the prompt always displays the current context.

To start the tool in batch mode, enter the command:

```
JMSAdmin <test.scp
```

where *test.scp* is a script file that contains administration commands (see "Administration commands" on page 38). The last command in the file must be the `END` command.

Configuration

Configure the administration tool with values for the following properties:

INITIAL_CONTEXT_FACTORY
The service provider that the tool uses. The supported values for this property are as follows:

- com.sun.jndi.ldap.LdapCtxFactory (for LDAP)
- com.sun.jndi.fscontext.RefFSContextFactory (for file system context)

On z/OS, com.ibm.jndi.LDAPCtxFactory is also supported and provides access to an LDAP server. However, this is incompatible with com.sun.jndi.ldap.LdapCtxFactory, in that objects created using one InitialContextFactory cannot be read or modified using the other.

You can also use an InitialContextFactory that is not in the list above. See "Using an unlisted InitialContextFactory" on page 37 for more details.

PROVIDER_URL
The URL of the session's initial context; the root of all JNDI operations carried out by the tool. Two forms of this property are supported:

- ldap://hostname/contextname
- file:[drive:]/pathname

SECURITY_AUTHENTICATION
Whether JNDI passes security credentials to your service provider. This property is used only when an LDAP service provider is used. This property can take one of three values:

- none (anonymous authentication)
- simple (simple authentication)
- CRAM-MD5 (CRAM-MD5 authentication mechanism)

If a valid value is not supplied, the property defaults to none. See "Security" on page 37 for more details about security with the administration tool.

These properties are set in a configuration file. When you invoke the tool, you can specify this configuration by using the `-cfg` command-line parameter, as described in "Invoking the administration tool" on page 35. If you do not specify a configuration file name, the tool attempts to load the default configuration file (`JMSAdmin.config`). It looks for this file first in the current directory, and then in the `<MQ_JAVA_INSTALL_PATH>/bin` directory, where `<MQ_JAVA_INSTALL_PATH>` is the path to your WebSphere MQ JMS installation.

The configuration file is a plain-text file that consists of a set of key-value pairs, separated by =. This is shown in the following example:

```
#Set the service provider
    INITIAL_CONTEXT_FACTORY=com.sun.jndi.ldap.LdapCtxFactory
#Set the initial context
    PROVIDER_URL=ldap://polaris/o=ibm_us,c=us
#Set the authentication type
    SECURITY_AUTHENTICATION=none
```

(A # in the first column of the line indicates a comment, or a line that is not used.)

The installation comes with a sample configuration file that is called `JMSAdmin.config`, and is found in the `<MQ_JAVA_INSTALL_PATH>/bin` directory. Edit this file to suit the setup of your system.

Using an unlisted InitialContextFactory

You can use the administration tool to connect to JNDI contexts other than those listed in "Configuration" on page 36 by using three parameters defined in the JMSAdmin configuration file.

To use a different InitialContextFactory:

1. Set the INITIAL_CONTEXT_FACTORY property to the required class name.
2. Define the behavior of the InitialContextFactory using the USE_INITIAL_DIR_CONTEXT, NAME_PREFIX and NAME_READABILITY_MARKER properties.

The settings for these properties are described in the sample configuration file comments.

You do not need to define the three properties listed here, if you use one of the supported INITIAL_CONTEXT_FACTORY values. However, you can give them values to override the system defaults. If you omit one or more of the three InitialContextFactory properties, the administration tool provides suitable defaults based on the values of the other properties.

Security

You need to understand the effect of the `SECURITY_AUTHENTICATION` property described in "Configuration" on page 36.

- If you set this parameter to `none`, JNDI does not pass any security credentials to the service provider, and *anonymous authentication* is performed.
- If you set the parameter to either `simple` or `CRAM-MD5`, security credentials are passed through JNDI to the underlying service provider. These security credentials are in the form of a user distinguished name (User DN) and password.

If security credentials are required, you are prompted for these when the tool initializes. Avoid this by setting the PROVIDER_USERDN and PROVIDER_PASSWORD properties in the JMSAdmin configuration file.

Note: If you do not use these properties, the text typed, *including the password*, is echoed to the screen. This may have security implications.

The tool does no authentication itself; the task is delegated to the LDAP server. The LDAP server administrator must set up and maintain access privileges to different parts of the directory. If authentication fails, the tool displays an appropriate error message and terminates.

More detailed information about security and JNDI is in the documentation at Sun's Java web site (`http://java.sun.com`).

Administration commands

When the command prompt is displayed, the tool is ready to accept commands. Administration commands are generally of the following form:

```
verb [param]*
```

where `verb` is one of the administration verbs listed in Table 7. All valid commands consist of at least one (and only one) verb, which appears at the beginning of the command in either its standard or short form.

The parameters a verb can take depend on the verb. For example, the `END` verb cannot take any parameters, but the `DEFINE` verb can take any number of parameters. Details of the verbs that take at least one parameter are discussed in later sections of this chapter.

Table 7. Administration verbs

Verb	Short form	Description
ALTER	ALT	Change at least one of the properties of a given administered object
DEFINE	DEF	Create and store an administered object, or create a new subcontext
DISPLAY	DIS	Display the properties of one or more stored administered objects, or the contents of the current context
DELETE	DEL	Remove one or more administered objects from the namespace, or remove an empty subcontext
CHANGE	CHG	Alter the current context, allowing the user to traverse the directory namespace anywhere below the initial context (pending security clearance)
COPY	CP	Make a copy of a stored administered object, storing it under an alternative name
MOVE	MV	Alter the name under which an administered object is stored
END		Close the administration tool

Verb names are not case-sensitive.

Usually, to terminate commands, you press the carriage return key. However, you can override this by typing the + symbol directly before the carriage return. This enables you to enter multiline commands, as shown in the following example:

```
DEFINE Q(BookingsInputQueue) +
       QMGR(QM.POLARIS.TEST) +
       QUEUE(BOOKINGS.INPUT.QUEUE) +
       PORT(1415) +
       CCSID(437)
```

Lines beginning with one of the characters *, #, or / are treated as comments, or lines that are ignored.

Manipulating subcontexts

Use the verbs `CHANGE`, `DEFINE`, `DISPLAY` and `DELETE` to manipulate directory namespace subcontexts. Their use is described in Table 8.

Table 8. Syntax and description of commands used to manipulate subcontexts

Command syntax	Description
DEFINE CTX(ctxName)	Attempts to create a new child subcontext of the current context, having the name ctxName. Fails if there is a security violation, if the subcontext already exists, or if the name supplied is not valid.
DISPLAY CTX	Displays the contents of the current context. Administered objects are annotated with `a`, subcontexts with `[D]`. The Java type of each object is also displayed.
DELETE CTX(ctxName)	Attempts to delete the current context's child context having the name ctxName. Fails if the context is not found, is non-empty, or if there is a security violation.
CHANGE CTX(ctxName)	Alters the current context, so that it now refers to the child context having the name ctxName. One of two special values of ctxName can be supplied: **=UP** moves to the current context's parent **=INIT** moves directly to the initial context Fails if the specified context does not exist, or if there is a security violation.

Administering JMS objects

This section describes the eight types of object that the administration tool can handle. It includes details about each of their configurable properties and the verbs that can manipulate them.

Object types

Table 9 shows the eight types of administered objects. The Keyword column shows the strings that you can substitute for *TYPE* in the commands shown in Table 10 on page 40.

Table 9. The JMS object types that are handled by the administration tool

Object Type	Keyword	Description
MQConnectionFactory	CF	The WebSphere MQ implementation of the JMS ConnectionFactory interface. This represents a factory object for creating connections in the both the point-to-point and publish/subscribe domains.
MQQueueConnectionFactory	QCF	The WebSphere MQ implementation of the JMS QueueConnectionFactory interface. This represents a factory object for creating connections in the point-to-point domain.

Table 9. The JMS object types that are handled by the administration tool (continued)

Object Type	Keyword	Description
MQTopicConnectionFactory	TCF	The WebSphere MQ implementation of the JMS TopicConnectionFactory interface. This represents a factory object for creating connections in the publish/subscribe domain.
MQQueue	Q	The WebSphere MQ implementation of the JMS Queue interface. This represents a destination for messages in the point-to-point domain.
MQTopic	T	The WebSphere MQ implementation of the JMS Topic interface. This represents a destination for messages in the publish/subscribe domain.
MQXAConnectionFactory[1]	XACF	The WebSphere MQ implementation of the JMS XAConnectionFactory interface. This represents a factory object for creating connections in both the point-to-point and publish/subscribe domains, and where the connections use the XA versions of JMS classes.
MQXAQueueConnectionFactory[1]	XAQCF	The WebSphere MQ implementation of the JMS XAQueueConnectionFactory interface. This represents a factory object for creating connections in the point-to-point domain that use the XA versions of JMS classes.
MQXATopicConnectionFactory[1]	XATCF	The WebSphere MQ implementation of the JMS XATopicConnectionFactory interface. This represents a factory object for creating connections in the publish/subscribe domain that use the XA versions of JMS classes.

Note:

1. These classes are provided for use by vendors of application servers. They are unlikely to be directly useful to application programmers.

Verbs used with JMS objects

You can use the verbs `ALTER`, `DEFINE`, `DISPLAY`, `DELETE`, `COPY`, and `MOVE` to manipulate administered objects in the directory namespace. Table 10 summarizes their use. Substitute *TYPE* with the keyword that represents the required administered object, as listed in Table 9 on page 39.

Table 10. Syntax and description of commands used to manipulate administered objects

Command syntax	Description
ALTER *TYPE*(name) [property]*	Attempts to update the given administered object's properties with the ones supplied. Fails if there is a security violation, if the specified object cannot be found, or if the new properties supplied are not valid.

Table 10. Syntax and description of commands used to manipulate administered objects (continued)

Command syntax	Description
DEFINE *TYPE*(name) [property]*	Attempts to create an administered object of type *TYPE* with the supplied properties, and store it under the name name in the current context. Fails if there is a security violation, if the supplied name is not valid or already exists, or if the properties supplied are not valid.
DISPLAY *TYPE*(name)	Displays the properties of the administered object of type *TYPE*, bound under the name name in the current context. Fails if the object does not exist, or if there is a security violation.
DELETE *TYPE*(name)	Attempts to remove the administered object of type *TYPE*, having the name name, from the current context. Fails if the object does not exist, or if there is a security violation.
COPY *TYPE*(nameA) *TYPE*(nameB)	Makes a copy of the administered object of type *TYPE*, having the name nameA, naming the copy nameB. This all occurs within the scope of the current context. Fails if the object to be copied does not exist, if an object of name nameB already exists, or if there is a security violation.
MOVE *TYPE*(nameA) *TYPE*(nameB)	Moves (renames) the administered object of type *TYPE*, having the name nameA, to nameB. This all occurs within the scope of the current context. Fails if the object to be moved does not exist, if an object of name nameB already exists, or if there is a security violation.

Creating objects

Objects are created and stored in a JNDI namespace using the following command syntax:

```
DEFINE TYPE(name) [property]*
```

That is, the DEFINE verb, followed by a *TYPE*(name) administered object reference, followed by zero or more *properties* (see "Properties" on page 42).

LDAP naming considerations

To store your objects in an LDAP environment, you must give them names that comply with certain conventions. One of these is that object and subcontext names must include a prefix, such as cn= (common name), or ou= (organizational unit).

The administration tool simplifies the use of LDAP service providers by allowing you to refer to object and context names without a prefix. If you do not supply a prefix, the tool automatically adds a default prefix to the name you supply. For LDAP this is cn=.

You can change the default prefix by setting the NAME_PREFIX property in the JMSAdmin configuration file, as described in "Using an unlisted InitialContextFactory" on page 37.

This is shown in the following example.

```
InitCtx> DEFINE Q(testQueue)

InitCtx> DISPLAY CTX

    Contents of InitCtx

      a  cn=testQueue              com.ibm.mq.jms.MQQueue

     1 Object(s)
       0 Context(s)
       1 Binding(s), 1 Administered
```

Note that, although the object name supplied (`testQueue`) does not have a prefix, the tool automatically adds one to ensure compliance with the LDAP naming convention. Likewise, submitting the command `DISPLAY Q(testQueue)` also causes this prefix to be added.

You might need to configure your LDAP server to store Java objects. For information to assist with this configuration, see the documentation for your LDAP server.

Properties

A property consists of a name-value pair in the format:

```
PROPERTY_NAME(property_value)
```

Property names are not case-sensitive, and are restricted to the set of recognized names shown in Table 11. This table also shows the valid property values for each property.

Table 11. Property names and valid values

Property	Short form	Valid values (defaults in bold)
BROKERCCDURSUBQ	CCDSUB	• **SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE** • Any valid string. See "Configuring durable topic subscribers" on page 362.
BROKERCCSUBQ	CCSUB	• **SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE** • Any valid string. See "Configuring nondurable message consumers" on page 362.
BROKERCONQ	BCON	• **SYSTEM.BROKER.CONTROL.QUEUE** • Any string
BROKERDURSUBQ	BDSUB	• **SYSTEM.JMS.D.SUBSCRIBER.QUEUE** • Any valid string. See "Configuring durable topic subscribers" on page 362.
BROKERPUBQ	BPUB	• **SYSTEM.BROKER.DEFAULT.STREAM** • Any string
BROKERPUBQMGR	BPQM	Any string
BROKERQMGR	BQM	Any string
BROKERSUBQ	BSUB	• **SYSTEM.JMS.ND.SUBSCRIBER.QUEUE** • Any valid string. See "Configuring nondurable message consumers" on page 362.

Table 11. Property names and valid values (continued)

Property	Short form	Valid values (defaults in bold)
BROKERVER	BVER	• **V1** - To use a WebSphere MQ Publish/Subscribe broker, or to use a broker of WebSphere MQ Integrator, WebSphere MQ Event Broker, WebSphere Business Integration Event Broker, or WebSphere Business Integration Message Broker in compatibility mode. This is the default value if TRANSPORT is set to BIND or CLIENT. • **V2** - To use a broker of WebSphere MQ Integrator, WebSphere MQ Event Broker, WebSphere Business Integration Event Broker, or WebSphere Business Integration Message Broker in native mode. This is the default value if TRANSPORT is set to DIRECT or DIRECTHTTP.
CCDTURL[1]	CCDT	• **Not set** • A uniform resource locator (URL)
CCSID	CCS	• **819** - This is the default for a connection factory. • **1208** - This is the default for a destination. • Any positive integer
CHANNEL[1]	CHAN	• **SYSTEM.DEF.SVRCONN** • Any string
CLEANUP	CL	• **SAFE** • ASPROP • NONE • STRONG
CLEANUPINT	CLINT	• **3 600 000** • Any positive integer
CLIENTID	CID	Any string
CLONESUPP	CLS	• **DISABLED** - Only one instance of a durable topic subscriber can run at a time. • ENABLED[2] - Two or more instances of the same durable topic subscriber can run simultaneously, but each instance must run in a separate Java virtual machine (JVM).
COMPHDR	HC	• **NONE** • SYSTEM
COMPMSG	MC	• **NONE** • A list of one or more of the following values separated by blank characters: RLE ZLIBFAST ZLIBHIGH

Table 11. Property names and valid values (continued)

Property	Short form	Valid values (defaults in bold)
CONNOPT[3]	CNOPT	• **STANDARD** - The nature of the binding between the application and the queue manager depends on the platform on which the queue manager is running and how the queue manager is configured. • SHARED - The application and the local queue manager agent run in separate units of execution but share some resources. • ISOLATED - The application and the local queue manager agent run in separate units of execution and share no resources. • FASTPATH - The application and the local queue manager agent run in the same unit of execution. • SERIALQM - The application requests exclusive use of the connection tag within the scope of the queue manager. • SERIALQSG - The application requests exclusive use of the connection tag within the scope of the queue sharing group to which the queue manager belongs. • RESTRICTQM - The application requests shared use of the connection tag, but there are restrictions on the shared use of the connection tag within the scope of the queue manager. • RESTRICTQSG - The application requests shared use of the connection tag, but there are restrictions on the shared use of the connection tag within the scope of the queue sharing group to which the queue manager belongs.
CONNTAG	CNTAG	Any string. The value is truncated if it is longer than 128 bytes.
DESCRIPTION	DESC	Any string
DIRECTAUTH	DAUTH	• **BASIC** - No authentication, username authentication, or password authentication • CERTIFICATE - Public key certificate authentication
ENCODING	ENC	See “The ENCODING property” on page 58
EXPIRY	EXP	• **APP** - Expiry may be defined by the JMS application. • UNLIM - No expiry occurs. • Any positive integer representing expiry in milliseconds.

Table 11. Property names and valid values (continued)

Property	Short form	Valid values (defaults in bold)
FAILIFQUIESCE	FIQ	• **YES** - Calls to certain methods fail if the queue manager is in a quiescing state. If an application detects that the queue manager is quiescing, the application can complete its immediate task and close the connection, allowing the queue manager to stop. • NO - No method call fails because the queue manager is in a quiescing state. If you specify this value, an application cannot detect that the queue manager is quiescing. The application might continue to perform operations against the queue manager, and therefore prevent the queue manager from stopping.
HOSTNAME	HOST	• **localhost** • Any string
LOCALADDRESS	LA	• **Not set** • A string in the format [*ip-addr*][(*low-port*[,*high-port*])] Here are some examples: **9.20.4.98** The channel binds to address 9.20.4.98 locally **9.20.4.98(1000)** The channel binds to address 9.20.4.98 locally and uses port 1000 **9.20.4.98(1000,2000)** The channel binds to address 9.20.4.98 locally and uses a port in the range 1000 to 2000 **(1000)** The channel binds to port 1000 locally **(1000,2000)** The channel binds to a port in the range 1000 to 2000 locally You can specify a host name instead of an IP address. For a direct connection to a broker, this property is relevant only when multicast is used, and the value of the property must not contain a port number, or a range of port numbers. The only valid values of the property in this case are null, an IP address, or a host name.
MAPNAMESTYLE	MNST	• **STANDARD** - Map messages are sent in the current format, which can be interpreted only by the current version or Version 5.3 of WebSphere MQ JMS. • COMPATIBLE - Map messages are sent in an earlier format, which can be interpreted by any version of WebSphere MQ JMS, including versions earlier than Version 5.3.
MAXBUFFSIZE	MBSZ	• **1000** • Any positive integer

Table 11. Property names and valid values (continued)

Property	Short form	Valid values (defaults in bold)
MSGBATCHSZ	MBS	• **10** • Any positive integer
MSGRETENTION	MRET	• **Yes** - Unwanted messages remain on the input queue • No - Unwanted messages are dealt with according to their disposition options
MSGSELECTION	MSEL	• **CLIENT** - Message selection is done by the client. • BROKER - Message selection is done by the broker.
MULTICAST	MCAST	• **DISABLED** - Messages are not delivered to a message consumer using multicast transport. This is the default value for ConnectionFactory and TopicConnectionFactory objects. • **ASCF** - Messages are delivered to a message consumer according to the multicast setting for the connection factory associated with the message consumer. The multicast setting for the connection factory is noted at the time that the message consumer is created. This value is valid only for Topic objects, and is the default value for Topic objects. • ENABLED - If the topic is configured for multicast in the broker, messages are delivered to a message consumer using multicast transport. A reliable quality of service is used if the topic is configured for reliable multicast. • RELIABLE - If the topic is configured for reliable multicast in the broker, messages are delivered to the message consumer using multicast transport with a reliable quality of service. If the topic is not configured for reliable multicast, you cannot create a message consumer for the topic. • NOTR - If the topic is configured for multicast in the broker, messages are delivered to the message consumer using multicast transport. A reliable quality of service is not used even if the topic is configured for reliable multicast.
OPTIMISTICPUBLICATION	OPTPUB	• **NO** - When a publisher publishes a message, the WebSphere MQ JMS client does not return control to the publisher until it has completed all the processing associated with the call and can report the outcome to the publisher. • YES - When a publisher publishes a message, the WebSphere MQ JMS client returns control to the publisher immediately, before it has completed all the processing associated with the call and can report the outcome to the publisher. The WebSphere MQ JMS client reports the outcome only when the publisher commits the message.

Table 11. Property names and valid values (continued)

Property	Short form	Valid values (defaults in bold)
OUTCOMENOTIFICATION	NOTIFY	• **YES** - When a subscriber acknowledges or commits a message, the WebSphere MQ JMS client does not return control to the subscriber until it has completed all the processing associated with the call and can report the outcome to the subscriber. • NO[4] - When a subscriber acknowledges or commits a message, the WebSphere MQ JMS client returns control to the subscriber immediately, before it has completed all the processing associated with the call and can report the outcome to the subscriber.
PERSISTENCE	PER	• **APP** - Persistence is defined by the JMS application. • QDEF - Persistence takes the value of the queue default. • PERS - Messages are persistent. • NON - Messages are nonpersistent. • HIGH - See "JMS persistent messages" on page 365.
POLLINGINT	PINT	• **5000** • Any positive integer
PORT		• **1414** - This is the default value if TRANSPORT is set to CLIENT. • **1506** - This is the default value if TRANSPORT is set to DIRECT or DIRECTHTTP. • Any positive integer
PRIORITY	PRI	• **APP** - Priority is defined by the JMS application. • QDEF - Priority takes the value of the queue default. • Any integer in the range 0-9.
PROCESSDURATION	PROCDUR	• **UNKNOWN** - A subscriber can give no guarantee about how quickly it can process any message it receives. • SHORT - A subscriber guarantees to process quickly any message it receives before returning control to the WebSphere MQ JMS client.
PROXYHOSTNAME	PHOST	• **Not set** • The host name of the proxy server
PROXYPORT	PPORT	• **443** • The port number of the proxy server
PUBACKINT	PAI	• **25** • Any positive integer
QMANAGER	QMGR	Any string
QUEUE	QU	Any string

Table 11. Property names and valid values (continued)

Property	Short form	Valid values (defaults in bold)
RECEIVEISOLATION	RCVISOL	• **COMMITTED** - A subscriber receives only those messages on the subscriber queue that have been committed. • UNCOMMITTED[5] - A subscriber can receive messages that have not been committed on the subscriber queue.
RECEXIT	RCX	• **Not set** • A string comprising one or more items separated by commas, where each item is one of the following: – The name of a class that implements the WebSphere MQ base Java interface, MQReceiveExit (for a channel receive exit written in Java) – A string in the format *libraryName*(*entryPointName*) (for a channel receive exit not written in Java)
RECEXITINIT	RCXI	• **Not set** • A string comprising one or more items of user data separated by commas
RESCANINT	RINT	• **5000** • Any positive integer
SECEXIT	SCX	• **Not set** • The name of a class that implements the WebSphere MQ base Java interface, MQSecurityExit (for a channel security exit written in Java) • A string in the format *libraryName*(*entryPointName*) (for a channel security exit not written in Java)
SECEXITINIT	SCXI	• **Not set** • Any string
SENDEXIT	SDX	• **Not set** • A string comprising one or more items separated by commas, where each item is one of the following: – The name of a class that implements the WebSphere MQ base Java interface, MQSendExit (for a channel send exit written in Java) – A string in the format *libraryName*(*entryPointName*) (for a channel send exit not written in Java)
SENDEXITINIT	SDXI	• **Not set** • A string comprising one or more items of user data separated by commas
SPARSESUBS	SSUBS	• **NO** - Subscriptions receive frequent matching messages. • YES - Subscriptions receive infrequent matching messages. This value requires that the subscription queue can be opened for browse.

Table 11. Property names and valid values (continued)

Property	Short form	Valid values (defaults in bold)
SSLCIPHERSUITE	SCPHS	• **Not set** • See "SSL properties" on page 59
SSLCRL	SCRL	• **Not set** • Space-separated list of LDAP URLs. See "SSL properties" on page 59
SSLFIPSREQUIRED	SFIPS	• **NO** - An SSL connection can use any CipherSuite that is not supported by the IBM Java JSSE FIPS provider (IBMJSSEFIPS). • YES - An SSL connection must use a CipherSuite that is supported by IBMJSSEFIPS.
SSLPEERNAME	SPEER	• **Not set** • See "SSL properties" on page 59
SSLRESETCOUNT	SRC	• **0** • Zero, or any positive integer less than or equal to 999 999 999. See "SSL properties" on page 59
STATREFRESHINT	SRI	• **60 000** • Any positive integer
SUBSTORE	SS	• **MIGRATE** • QUEUE • BROKER
SYNCPOINTALLGETS	SPAG	• **No** • Yes
TARGCLIENT	TC	• **JMS** - The target of the message is a JMS application. • MQ - The target of the message is a non-JMS WebSphere MQ application.
TARGCLIENTMATCHING	TCM	• **YES** - If an incoming message does not have an MQRFH2 header, the TARGCLIENT property of the Queue object derived from the JMSReplyTo header field of the message is set to MQ. If the message does have an MQRFH2 header, the TARGCLIENT property is set to JMS instead. • NO - The TARGCLIENT property of the Queue object derived from the JMSReplyTo header field of an incoming message is always set to JMS.
TEMPMODEL	TM	• **SYSTEM.DEFAULT.MODEL.QUEUE** • Any string
TEMPQPREFIX	TQP	Any string
TOPIC	TOP	Any string

Table 11. Property names and valid values (continued)

Property	Short form	Valid values (defaults in bold)
TRANSPORT	TRAN	• **BIND** - For a bindings connection • CLIENT - For a client connection • DIRECT - For a direct connection to a broker of WebSphere MQ Event Broker, WebSphere Business Integration Event Broker, or WebSphere Business Integration Message Broker • DIRECTHTTP - For a direct connection using HTTP tunnelling.
USECONNPOOLING	UCP	• **Yes** • No

Note:

1. The CCDTURL and CHANNEL properties of an object cannot both be set at the same time.
2. Running two or more instances of the same durable topic subscriber simultaneously contravenes the *Java Message Service Specification, Version 1.1*.
3. The binding options, STANDARD, SHARED, ISOLATED, and FASTPATH, are ignored if the application connects in client mode. The SHARED, ISOLATED, and FASTPATH options are ignored by a queue manager running on z/OS. The connection tag options, SERIALQM, SERIALQSG, RESTRICTQM, and RESTRICTQSG, are supported only by a queue manager running on z/OS. For a more detailed explanation of the connection options, see the *WebSphere MQ Application Programming Reference*.
4. If you specify NO, and a message is rolled back after the WebSphere MQ JMS client has returned control to the subscriber, the subscriber still retains a copy of the message but is not informed of the rollback. In this situation, a subscriber might receive the same message more than once.
5. The value UNCOMMITTED has an effect only if PROCESSDURATION has the value SHORT. It has no effect if PROCESSDURATION has the value UNKNOWN. If you specify UNCOMMITTED, ensure that a subscriber acknowledges or commits each message individually.

Many of the properties are relevant only to a specific subset of the object types. Table 12 shows for each property which object types are valid, and gives a brief description of each property. The object types are identified using keywords; refer to Table 9 on page 39 for an explanation of these.

Numbers refer to notes at the end of the table. See also “Property dependencies” on page 57. Appendix A, “Mapping between administration tool properties and programmable properties,” on page 633 shows the relationship between properties set by the tool and programmable properties.

Table 12. The valid combinations of property and object type

Property	CF[1]	QCF	TCF	Q	T	XACF[1]	XAQCF	XATCF	Description
BROKERCCDURSUBQ					Y				The name of the queue from which durable subscription messages are retrieved for a ConnectionConsumer
BROKERCCSUBQ	Y		Y			Y		Y	The name of the queue from which non-durable subscription messages are retrieved for a ConnectionConsumer

Table 12. The valid combinations of property and object type (continued)

Property	CF[1]	QCF	TCF	Q	T	XACF[1]	XAQCF	XATCF	Description
BROKERCONQ	Y		Y			Y		Y	Broker's control queue name
BROKERDURSUBQ					Y				The name of the queue from which durable subscription messages are retrieved
BROKERPUBQ	Y		Y		Y	Y		Y	The name of the queue where published messages are sent (the stream queue)
BROKERPUBQMGR					Y				The name of the queue manager that owns the queue where messages published on the topic are sent
BROKERQMGR	Y		Y			Y		Y	The name of the queue manager on which the broker is running
BROKERSUBQ	Y		Y			Y		Y	The name of the queue from which non-durable subscription messages are retrieved
BROKERVER	Y[2]		Y[2]		Y	Y		Y	The version of the broker being used
CCDTURL[3]	Y	Y	Y			Y	Y	Y	A uniform resource locator (URL) that identifies the name and location of the file containing the client channel definition table and specifies how the file can be accessed
CCSID	Y	Y	Y	Y	Y	Y	Y	Y	The coded-character-set-ID to be used on connections
CHANNEL[3]	Y	Y	Y			Y	Y	Y	The name of the client connection channel being used
CLEANUP	Y		Y			Y		Y	Cleanup Level for BROKER or MIGRATE Subscription Stores
CLEANUPINT	Y		Y			Y		Y	The interval between background executions of the publish/subscribe cleanup utility
CLIENTID	Y[2]	Y	Y[2]			Y	Y	Y	A string identifier for the client
CLONESUPP	Y		Y			Y		Y	Whether two or more instances of the same durable topic subscriber can run simultaneously
COMPHDR	Y	Y	Y			Y	Y	Y	A list of the techniques that can be used for compressing header data on a connection
COMPMSG	Y	Y	Y			Y	Y	Y	A list of the techniques that can be used for compressing message data on a connection
CONNOPT	Y	Y	Y			Y	Y	Y	Options that control how the application connects to the queue manager

Table 12. The valid combinations of property and object type (continued)

Property	CF[1]	QCF	TCF	Q	T	XACF[1]	XAQCF	XATCF	Description
CONNTAG[4]	Y	Y	Y			Y	Y	Y	A tag that the queue manager associates with the resources updated by the application within a unit of work while the application is connected to the queue manager
DESCRIPTION	Y[2]	Y	Y[2]	Y	Y	Y	Y	Y	A description of the stored object
DIRECTAUTH	Y		Y						To enable SSL authentication for a direct connection[5]
ENCODING				Y	Y				The encoding scheme used for this destination
EXPIRY				Y	Y				The period after which messages at a destination expire
FAILIFQUIESCE	Y	Y	Y	Y	Y	Y	Y	Y	Whether calls to certain methods fail if the queue manager is in a quiescing state
HOSTNAME	Y[2]	Y	Y[2]			Y	Y	Y	The host name or IP address of the system on which the queue manager resides or, for a direct connection to a broker, the system on which the broker resides
LOCALADDRESS	Y	Y	Y			Y	Y	Y	For a connection to a queue manager, this property specifies either or both of the following: • The local network interface to be used • The local port, or range of local ports, to be used For a direct connection to a broker, this property is relevant only when multicast is used, and specifies the local network interface to be used.
MAPNAMESTYLE	Y	Y	Y						Determines how the body of a map message is encoded when the message is sent.
MAXBUFFSIZE	Y		Y					Y	The maximum number of received messages that can be stored in an internal message buffer while waiting to be processed by the client application. This property applies only when TRANSPORT has the value DIRECT or DIRECTHTTP.
MSGBATCHSZ	Y	Y	Y			Y	Y	Y	The maximum number of messages to be taken from a queue in one packet when using asynchronous message delivery
MSGRETENTION	Y	Y				Y	Y		Whether or not the connection consumer keeps unwanted messages on the input queue

Table 12. The valid combinations of property and object type (continued)

Property	CF[1]	QCF	TCF	Q	T	XACF[1]	XAQCF	XATCF	Description
MSGSELECTION	Y		Y			Y		Y	Determines whether message selection is done by the JMS client or by the broker. If TRANSPORT has the value DIRECT, message selection is always done by the broker and the value of MSGSELECTION is ignored. Message selection by the broker is not supported when BROKERVER has the value V1.
MULTICAST	Y		Y		Y				To enable multicast transport on a direct connection and, if enabled, to specify the precise way in which multicast transport is used to deliver messages from the broker to a message consumer. The property has no effect on how a message producer sends messages to the broker.[5]
OPTIMISTICPUBLICATION	Y		Y						Whether the WebSphere MQ JMS client returns control immediately to a publisher that has just published a message, or whether it returns control only after it has completed all the processing associated with the call and can report the outcome to the publisher
OUTCOMENOTIFICATION	Y		Y						Whether the WebSphere MQ JMS client returns control immediately to a subscriber that has just acknowledged or committed a message, or whether it returns control only after it has completed all the processing associated with the call and can report the outcome to the subscriber
PERSISTENCE				Y	Y				The persistence of messages sent to a destination
POLLINGINT	Y	Y	Y			Y	Y	Y	If each message listener within a session has no suitable message on its queue, this is the maximum interval, in milliseconds, that elapses before each message listener tries again to get a message from its queue. If it frequently happens that no suitable message is available for any of the message listeners in a session, consider increasing the value of this property. This property is relevant only if TRANSPORT has the value BIND or CLIENT.
PORT	Y[2]	Y	Y[2]			Y	Y	Y	The port on which the queue manager or broker listens

Table 12. The valid combinations of property and object type (continued)

Property	CF[1]	QCF	TCF	Q	T	XACF[1]	XAQCF	XATCF	Description
PRIORITY				Y	Y				The priority for messages sent to a destination
PROCESSDURATION	Y		Y						Whether a subscriber guarantees to process quickly any message it receives before returning control to the WebSphere MQ JMS client
PROXYHOSTNAME	Y		Y						The host name of the proxy server for a direct connection[5]
PROXYPORT	Y		Y						The port number of the proxy server for a direct connection[5]
PUBACKINT	Y		Y			Y		Y	The number of messages published by a publisher before the WebSphere MQ JMS client requests an acknowledgement from the broker. If you lower the value of this property, the client requests acknowledgements more often, and therefore the performance of the publisher decreases. If you raise the value, the client takes a longer time to throw an exception if the broker fails. This property is relevant only if TRANSPORT has the value BIND or CLIENT.
QMANAGER	Y	Y	Y	Y		Y	Y	Y	The name of the queue manager to connect to. But, if your application uses a client channel definition table to connect to a queue manager, see “Using a client channel definition table” on page 351.
QUEUE				Y					The underlying name of the queue representing this destination
RECEIVEISOLATION	Y		Y						Whether a subscriber might receive messages that have not been committed on the subscriber queue
RECEXIT	Y	Y	Y			Y	Y	Y	Identifies a channel receive exit, or a sequence of receive exits to be run in succession
RECEXITINIT	Y	Y	Y			Y	Y	Y	The user data that is passed to channel receive exits when they are called

Table 12. The valid combinations of property and object type (continued)

Property	CF[1]	QCF	TCF	Q	T	XACF[1]	XAQCF	XATCF	Description
RESCANINT	Y	Y				Y	Y		When a message consumer in the point-to-point domain uses a message selector to select which messages it wants to receive, the WebSphere MQ JMS client searches the WebSphere MQ queue for suitable messages in the sequence determined by the `MsgDeliverySequence` attribute of the queue. When the client finds a suitable message and delivers it to the consumer, the client resumes the search for the next suitable message from its current position in the queue. The client continues to search the queue in this way until it reaches the end of the queue, or until the interval of time in milliseconds, as determined by the value of this property, has expired. In each case, the client returns to the beginning of the queue to continue its search, and a new time interval commences.
SECEXIT	Y	Y	Y			Y	Y	Y	Identifies a channel security exit
SECEXITINIT	Y	Y	Y			Y	Y	Y	The user data that is passed to a channel security exit when it is called
SENDEXIT	Y	Y	Y			Y	Y	Y	Identifies a channel send exit, or a sequence of send exits to be run in succession
SENDEXITINIT	Y	Y	Y			Y	Y	Y	The user data that is passed to channel send exits when they are called
SPARSESUBS	Y		Y						Controls the message retrieval policy of a TopicSubscriber object
SSLCIPHERSUITE	Y	Y	Y			Y	Y	Y	The CipherSuite to use for an SSL connection
SSLCRL	Y	Y	Y			Y	Y	Y	CRL servers to check for SSL certificate revocation
SSLFIPSREQUIRED	Y	Y	Y			Y	Y	Y	Whether an SSL connection must use a CipherSuite that is supported by the IBM Java JSSE FIPS provider (IBMJSSEFIPS)
SSLPEERNAME	Y	Y	Y			Y	Y	Y	For SSL, a *distinguished name* skeleton that must match that provided by the queue manager
SSLRESETCOUNT	Y	Y	Y			Y	Y	Y	For SSL, the total number bytes sent and received by a connection before the secret key that is used for encryption is renegotiated.

Table 12. The valid combinations of property and object type (continued)

Property	CF[1]	QCF	TCF	Q	T	XACF[1]	XAQCF	XATCF	Description
STATREFRESHINT	Y		Y			Y		Y	The interval, in milliseconds, between refreshes of the long running transaction that detects when a subscriber loses its connection to the queue manager. This property is relevant only if SUBSTORE has the value QUEUE. For more information about setting this property, see "Subscription stores" on page 363.
SUBSTORE	Y		Y			Y		Y	Where WebSphere MQ JMS should store persistent data relating to active subscriptions
SYNCPOINTALLGETS	Y	Y	Y			Y	Y	Y	Whether all gets should be performed under syncpoint
TARGCLIENT[6]				Y	Y				Whether the WebSphere MQ RFH2 format is used to exchange information with target applications
TARGCLIENTMATCHING	Y	Y				Y	Y		Whether a reply message, sent to the queue identified by the JMSReplyTo header field of an incoming message, has an MQRFH2 header only if the incoming message has an MQRFH2 header
TEMPMODEL	Y	Y				Y	Y		The name of the model queue from which JMS temporary queues are created
TEMPQPREFIX	Y	Y				Y	Y		The prefix that is used to form the name of a WebSphere MQ dynamic queue. The rules for forming the prefix are the same as those for forming the contents of the *DynamicQName* field in a WebSphere MQ object descriptor, structure MQOD, but the last non blank character must be an asterisk. If no value is specified for the property, the value used is `CSQ.*` on z/OS and `AMQ.*` on the other platforms.
TOPIC					Y				The underlying name of the topic representing this destination
TRANSPORT	Y[2]	Y	Y[2]			Y	Y	Y	Whether a connection uses bindings or client mode to connect to a queue manager, or whether it is a direct connection to a broker
USECONNPOOLING	Y	Y	Y			Y	Y	Y	Whether to use connection pooling

Table 12. The valid combinations of property and object type (continued)

Property	CF[1]	QCF	TCF	Q	T	XACF[1]	XAQCF	XATCF	Description

Note:

1. This object type applies to JMS 1.1 only.
2. Only the BROKERVER, CLIENTID, DESCRIPTION, HOSTNAME, LOCALADDRESS, PORT, and TRANSPORT properties are supported for a TopicConnectionFactory object, or a JMS 1.1 domain independent ConnectionFactory object, when connecting directly to a broker.
3. The CCDTURL and CHANNEL properties of an object cannot both be set at the same time.
4. The CONNTAG property is supported only by a queue manager running on z/OS.
5. See Appendix C, "Connecting to other products," on page 639.
6. The TARGCLIENT property indicates whether the WebSphere MQ RFH2 format is used to exchange information with target applications.

   The MQJMS_CLIENT_JMS_COMPLIANT constant indicates that the RFH2 format is used to send information. Applications that use WebSphere MQ JMS understand the RFH2 format. Set the MQJMS_CLIENT_JMS_COMPLIANT constant when you exchange information with a target WebSphere MQ JMS application.

   The MQJMS_CLIENT_NONJMS_MQ constant indicates that the RFH2 format is not used to send information. Typically, this value is used for an existing WebSphere MQ application (that is, one that does not handle RFH2).

Property dependencies

Some properties have dependencies on each other. This might mean that it is meaningless to supply a property unless another property is set to a particular value. The specific property groups where this can occur are

- Client properties
- Properties for a direct connection to a broker
- Exit initialization strings

Client properties

For a connection to a queue manager, the following properties are relevant only if TRANSPORT has the value CLIENT:

- HOSTNAME
- PORT
- CHANNEL
- LOCALADDRESS
- CCDTURL
- CCSID
- COMPHDR
- COMPMSG
- RECEXIT
- RECEXITINIT
- SECEXIT
- SECEXITINIT
- SENDEXIT
- SENDEXITINIT
- SSLCIPHERSUITE
- SSLCRL
- SSLFIPSREQUIRED
- SSLPEERNAME

- SSLRESETCOUNT

Using the administration tool, you cannot set values for these properties if TRANSPORT has the value BIND.

If TRANSPORT has the value CLIENT, the default value of the BROKERVER property is V1 and the default value of the PORT property is 1414. If you set the value of BROKERVER or PORT explicitly, a later change to the value of TRANSPORT does not override your choices.

Properties for a direct connection to a broker

Only the following properties are relevant if TRANSPORT has the value DIRECT or DIRECTHTTP:

- BROKERVER
- CLIENTID
- DESCRIPTION
- HOSTNAME
- PORT
- LOCALADDRESS
- MULTICAST (only supported for DIRECT)

If TRANSPORT has the value DIRECT or DIRECTHTTP, the default value of the BROKERVER property is V2, and the default value of the PORT property is 1506. If you set the value of BROKERVER or PORT explicitly, a later change to the value of TRANSPORT does not override your choices.

Exit initialization strings

Do not set any of the exit initialization strings without supplying the corresponding exit name. The exit initialization properties are:

- RECEXITINIT
- SECEXITINIT
- SENDEXITINIT

For example, specifying `RECEXITINIT(myString)` without specifying `RECEXIT(some.exit.classname)` causes an error.

The ENCODING property

The valid values that the `ENCODING` property can take are constructed from three sub-properties:

integer encoding

Either normal or reversed

decimal encoding

Either normal or reversed

floating-point encoding

IEEE normal, IEEE reversed, or z/OS.

The `ENCODING` is expressed as a three-character string with the following syntax:

```
{N|R}{N|R}{N|R|3}
```

In this string:

- `N` denotes normal
- `R` denotes reversed
- `3` denotes z/OS

- The first character represents *integer encoding*
- The second character represents *decimal encoding*
- The third character represents *floating-point encoding*

This provides a set of twelve possible values for the ENCODING property.

There is an additional value, the string NATIVE, which sets appropriate encoding values for the Java platform.

The following examples show valid combinations for ENCODING:

```
ENCODING(NNR)
ENCODING(NATIVE)
ENCODING(RR3)
```

SSL properties

When you specify TRANSPORT(CLIENT), you can enable Secure Sockets Layer (SSL) encrypted communication using the SSLCIPHERSUITE property. Set this property to a valid CipherSuite provided by your JSSE provider; it must match the CipherSpec named on the SVRCONN channel named by the CHANNEL property.

However, CipherSpecs (as specified on the SVRCONN channel) and CipherSuites (as specified on ConnectionFactory objects) use different naming schemes to represent the same SSL encryption algorithms. If a recognized CipherSpec name is specified on the SSLCIPHERSUITE property, JMSAdmin issues a warning and maps the CipherSpec to its equivalent CipherSuite. See Appendix D, "SSL CipherSpecs and CipherSuites," on page 645 for a list of CipherSpecs recognized by WebSphere MQ and JMSAdmin.

If you require a connection to use a CipherSuite that is supported by the IBM Java JSSE FIPS provider (IBMJSSEFIPS), set the SSLFIPSREQUIRED property of the connection factory to YES. The default value of this property is NO, which means that a connection can use any CipherSuite that is not supported by IBMJSSEFIPS. The property is ignored if SSLCIPHERSUITE is not set.

The SSLPEERNAME matches the format of the SSLPEER parameter, which can be set on channel definitions. It is a list of attribute name and value pairs separated by commas or semicolons. For example:

```
SSLPEERNAME(CN=QMGR.*, OU=IBM, OU=WEBSPHERE)
```

The set of names and values makes up a *distinguished name*. For more details about distinguished names and their use with WebSphere MQ, see the *WebSphere MQ Security* book.

The example given checks the identifying certificate presented by the server at connect-time. For the connection to succeed, the certificate must have a Common Name beginning QMGR., and must have at least two Organizational Unit names, the first of which is IBM and the second WEBSPHERE. Checking is case-insensitive.

If SSLPEERNAME is not set, no such checking is performed. SSLPEERNAME is ignored if SSLCIPHERSUITE is not set.

The SSLCRL property specifies zero or more CRL (Certificate Revocation List) servers. Use of this property requires a JVM at Java 2 v1.4. This is a space-delimited list of entries of the form:

`ldap://hostname:[port]`

optionally followed by a single /. If *port* is omitted, the default LDAP port of 389 is assumed. At connect-time, the SSL certificate presented by the server is checked against the specified CRL servers. See the *WebSphere MQ Security* book for more about CRL security.

If SSLCRL is not set, no such checking is performed. SSLCRL is ignored if SSLCIPHERSUITE is not set.

The SSLRESETCOUNT property represents the total number of bytes sent and received by a connection before the secret key that is used for encryption is renegotiated. The number of bytes sent is the number before encryption, and the number of bytes received is the number after decryption. The number of bytes also includes control information sent and received by the WebSphere MQ JMS client.

For example, to configure a ConnectionFactory object that can be used to create a connection over an SSL enabled MQI channel whose secret key is renegotiated after 4 MB of data have flowed, issue the following command to JMSAdmin:

```
ALTER CF(my.cf) SSLRESETCOUNT(4194304)
```

If the value of SSLRESETCOUNT is zero, which is the default value, the secret key is never renegotiated. The SSLRESETCOUNT property is ignored if SSLCIPHERSUITE is not set.

If you are using an HP or Sun Java 2 Software Development Kit (SDK) or Java Runtime Environment (JRE), do not set SSLRESETCOUNT to a value other than zero. If you do set SSLRESETCOUNT to a value other than zero, a connection fails when it attempts to renegotiate the secret key.

For more information about the secret key that is used for encryption on an SSL enabled channel, see *WebSphere MQ Security*.

Sample error conditions

The following are examples of the error conditions that might arise when creating an object:

CipherSpec mapped to CipherSuite

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) SSLCIPHERSUITE(RC4_MD5_US)
  WARNING: Converting CipherSpec RC4_MD5_US to
  CipherSuite SSL_RSA_WITH_RC4_128_MD5
```

Invalid property for object

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) PRIORITY(4)
  Unable to create a valid object, please check the parameters supplied
  Invalid property for a QCF: PRI
```

Invalid type for property value

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) CCSID(english)
  Unable to create a valid object, please check the parameters supplied
  Invalid value for CCS property: English
```

Property clash - client/bindings

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) HOSTNAME(polaris.hursley.ibm.com)
  Unable to create a valid object, please check the parameters supplied
  Invalid property in this context: Client-bindings attribute clash
```

Property clash - Exit initialization

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) SECEXITINIT(initStr)
  Unable to create a valid object, please check the parameters supplied
  Invalid property in this context: ExitInit string supplied
  without Exit string
```

Property value outside valid range

```
InitCtx/cn=Trash> DEFINE Q(testQ) PRIORITY(12)
  Unable to create a valid object, please check the parameters supplied
  Invalid value for PRI property: 12
```

Unknown property

```
InitCtx/cn=Trash> DEFINE QCF(testQCF) PIZZA(ham and mushroom)
  Unable to create a valid object, please check the parameters supplied
  Unknown property: PIZZA
```

The following are examples of error conditions that might arise on Windows when looking up JNDI administered objects from a JMS client.

1. If you are using the WebSphere JNDI provider, com.ibm.websphere.naming.WsnInitialContextFactory, you must use a forward slash (/) to access administered objects defined in sub-contexts; for example, jms/MyQueueName. If you use a backslash (\), an InvalidNameException is thrown.
2. If you are using the Sun JNDI provider, com.sun.jndi.fscontext.RefFSContextFactory, you must use a backslash (\) to access administered objects defined in sub-contexts; for example, ctx1\\fred. If you use a forward slash (/), a NameNotFoundException is thrown.

Part 2. Programming with WebSphere MQ base Java

Chapter 6. Introduction for programmers

This chapter contains general information for programmers. For more detailed information about writing programs, see Chapter 7, "Writing WebSphere MQ base Java applications," on page 67.

Why should I use the Java interface?

The WebSphere MQ classes for Java programming interface makes the many benefits of Java available to you as a developer of WebSphere MQ applications:

- The Java programming language is **easy to use**.

  There is no need for header files, pointers, structures, unions, and operator overloading. Programs written in Java are easier to develop and debug than their C and C++ equivalents.
- Java is **object-oriented**.

  The object-oriented features of Java are comparable to those of C++, but there is no multiple inheritance. Instead, Java uses the concept of an interface.
- Java is inherently **distributed**.

  The Java class libraries contain a library of routines for coping with TCP/IP protocols like HTTP and FTP. Java programs can access URLs as easily as accessing a file system.
- Java is **robust**.

  Java puts a lot of emphasis on early checking for possible problems, dynamic (runtime) checking, and the elimination of situations that are error prone. Java uses a concept of references that eliminates the possibility of overwriting memory and corrupting data.
- Java is **secure**.

  Java is intended to be run in networked or distributed environments, and a lot of emphasis has been placed on security. Java programs cannot overrun their runtime stack and cannot corrupt memory outside their process space. When Java programs are downloaded from the Internet, they cannot even read or write local files.
- Java programs are **portable**.

  There are no implementation-dependent aspects of the Java specification. The Java compiler generates an architecture-neutral object file format. The compiled code is executable on many processors, as long as the Java runtime system is present.

The WebSphere MQ classes for Java interface

The procedural WebSphere MQ application programming interface is built around the following verbs:

```
MQBACK, MQBEGIN, MQCLOSE, MQCMIT, MQCONN, MQCONNX,
MQDISC, MQGET, MQINQ, MQOPEN, MQPUT, MQPUT1, MQSET
```

These verbs all take, as a parameter, a handle to the WebSphere MQ object on which they are to operate. Because Java is object-oriented, the Java programming interface turns this round. Your program consists of a set of WebSphere MQ objects, which you act upon by calling methods on those objects.

When you use the procedural interface, you disconnect from a queue manager by using the call MQDISC(Hconn, CompCode, Reason), where *Hconn* is a handle to the queue manager.

In the Java interface, the queue manager is represented by an object of class MQQueueManager. You disconnect from the queue manager by calling the disconnect() method on that class.

```
// declare an object of type queue manager
MQQueueManager queueManager=new MQQueueManager();
...
// do something...
...
// disconnect from the queue manager
queueManager.disconnect();
```

Chapter 7. Writing WebSphere MQ base Java applications

To use WebSphere MQ classes for Java to access WebSphere MQ queues, you write Java applications that contain calls that put messages onto, and get messages from, WebSphere MQ queues. This chapter provides information to assist with writing Java applications to interact with WebSphere MQ systems. For details of individual classes, see Chapter 9, "Package com.ibm.mq," on page 105.

Connection differences

The way you program for WebSphere MQ classes for Java has some dependencies on the connection modes you want to use.

Client connections

When WebSphere MQ classes for Java is used as a client, it is similar to the WebSphere MQ C client, but has the following differences:

- It supports only TCP/IP.
- It does not read any WebSphere MQ environment variables at startup.
- Information that would be stored in a channel definition and in environment variables is stored in a class called Environment. Alternatively, this information can be passed as parameters when the connection is made.
- Error and exception conditions are written to a log specified in the MQException class. The default error destination is the Java console.
- It does not access information stored in a qm.ini file, or the equivalent information stored in the Windows Registry. Entries in a qm.ini file, such as the KeepAlive entry, are therefore ignored.

When used in client mode, WebSphere MQ classes for Java does not support the MQBEGIN call or fast path bindings.

For general information on WebSphere MQ clients, see the *WebSphere MQ Clients* book.

Bindings mode

The bindings mode of WebSphere MQ classes for Java differs from the client modes in the following ways:

- Most of the parameters provided by the MQEnvironment class are ignored
- The bindings support the MQBEGIN call and fast path bindings into the WebSphere MQ queue manager

Note: WebSphere MQ for iSeries and WebSphere MQ for z/OS do not support the use of MQBEGIN to initiate global units of work that are coordinated by the queue manager.

Defining which connection to use

The connection is determined by the setting of variables in the MQEnvironment class.

MQEnvironment.properties
This can contain the following key/value pairs:

- For client and bindings connections:
  `MQC.TRANSPORT_PROPERTY, MQC.TRANSPORT_MQSERIES`

MQEnvironment.hostname
Set the value of this variable follows:

- For client connections, set this to the host name of the WebSphere MQ server to which you want to connect
- For bindings mode, set this to null

Example application

The following code fragment demonstrates an application that uses bindings mode to:

1. Connect to a queue manager
2. Put a message onto SYSTEM.DEFAULT.LOCAL.QUEUE
3. Get the message back again

```
// ============================================================================
//
// Licensed Materials - Property of IBM
//
// 5724-H27, 5655-L82, 5724-L26
//
// (c) Copyright IBM Corp. 1995,2002,2005
//
// ============================================================================
// WebSphere MQ classes for Java sample application
//
// This sample runs as a Java Application using the command :- java MQSample
//
// @(#) javabase/samples/MQSample.java, java, j000 1.8 04/12/03 10:59:49

import com.ibm.mq.*;            // Include the WebSphere MQ classes for Java package


public class MQSample
{
  private String qManager = "your_Q_manager";  // define name of queue
                                               // manager to connect to.
  private MQQueueManager qMgr;                 // define a queue manager
                                               // object
  public static void main(String args[]) {
     new MQSample();
  }

  public MQSample() {
   try {

      // Create a connection to the queue manager

      qMgr = new MQQueueManager(qManager);

      // Set up the options on the queue we wish to open...
      // Note. All WebSphere MQ Options are prefixed with MQC in Java.

      int openOptions = MQC.MQOO_INPUT_AS_Q_DEF |
                        MQC.MQOO_OUTPUT ;

      // Now specify the queue that we wish to open,
      // and the open options...

      MQQueue system_default_local_queue =
              qMgr.accessQueue("SYSTEM.DEFAULT.LOCAL.QUEUE",
                               openOptions);

      // Define a simple WebSphere MQ message, and write some text in UTF format..

      MQMessage hello_world = new MQMessage();
      hello_world.writeUTF("Hello World!");

      // specify the message options...
```

```
      MQPutMessageOptions pmo = new MQPutMessageOptions(); // accept the // defaults,
                                                           // same as MQPMO_DEFAULT

      // put the message on the queue

      system_default_local_queue.put(hello_world,pmo);

      // get the message back again...
      // First define a WebSphere MQ message buffer to receive the message into..

      MQMessage retrievedMessage = new MQMessage();
      retrievedMessage.messageId = hello_world.messageId;

      // Set the get message options...

      MQGetMessageOptions gmo = new MQGetMessageOptions(); // accept the defaults
                                                           // same as  MQGMO_DEFAULT
      // get the message off the queue...

      system_default_local_queue.get(retrievedMessage, gmo);

      // And prove we have the message by displaying the UTF message text

      String msgText = retrievedMessage.readUTF();
      System.out.println("The message is: " + msgText);
      // Close the queue...
      system_default_local_queue.close();
      // Disconnect from the queue manager

      qMgr.disconnect();
    }
      // If an error has occurred in the above, try to identify what went wrong
      // Was it a WebSphere MQ error?
    catch (MQException ex)
    {
      System.out.println("A WebSphere MQ error occurred : Completion code " +
                         ex.completionCode + " Reason code " + ex.reasonCode);
    }
      // Was it a Java buffer space error?
    catch (java.io.IOException ex)
    {
      System.out.println("An error occurred whilst writing to the message buffer: " + ex);
    }
  }
} // end of sample
```

Operations on queue managers

This section describes how to connect to, and disconnect from, a queue manager using WebSphere MQ base Java.

Setting up the WebSphere MQ environment

The information in this section is not relevant if your application connects to a queue manager in bindings mode.

Before an application can connect to a queue manager in client mode, the application must set certain fields in the MQEnvironment class. These fields specify the following information, which is used during the connection attempt:

- Channel name
- Host name
- Port number

To specify the channel name and host name, use the following code:

```
MQEnvironment.hostname = "host.domain.com";
MQEnvironment.channel  = "java.client.channel";
```

This is equivalent to an MQSERVER environment variable setting of:

```
"java.client.channel/TCP/host.domain.com".
```

By default, the Java clients attempt to connect to a WebSphere MQ listener at port 1414. To specify a different port, use the code:

```
MQEnvironment.port = nnnn;
```

Connecting to a queue manager

You are now ready to connect to a queue manager by creating a new instance of the MQQueueManager class:

```
MQQueueManager queueManager = new MQQueueManager("qMgrName");
```

To disconnect from a queue manager, call the disconnect() method on the queue manager:

```
queueManager.disconnect();
```

If you call the disconnect method, all open queues and processes that you have accessed through that queue manager are closed. However, it is good programming practice to close these resources explicitly when you finish using them. To do this, use the close() method.

The commit() and backout() methods on a queue manager replace the MQCMIT and MQBACK calls that are used with the procedural interface.

Using a client channel definition table

As an alternative to creating a client connection channel definition by setting certain fields and environment properties in the MQEnvironment class, a WebSphere MQ base Java client application can use client connection channel definitions that are stored in a client channel definition table. These definitions are created by WebSphere MQ Script (MQSC) commands or WebSphere MQ Programmable Command Format (PCF) commands. When the application creates an MQQueueManager object, the WebSphere MQ base Java client searches the client channel definition table for a suitable client connection channel definition, and uses the channel definition to start an MQI channel. For more information about client channel definition tables and how to construct one, see *WebSphere MQ Clients*.

To use a client channel definition table, an application must first create a URL object. The URL object encapsulates a uniform resource locator (URL) that identifies the name and location of the file containing the client channel definition table and specifies how the file can be accessed.

For example, if the file ccdt1.tab contains a client channel definition table and is stored on the same system on which the application is running, the application can create a URL object in the following way:

```
java.net.URL chanTab1 = new URL("file:///home/admdata/ccdt1.tab");
```

As another example, suppose the file ccdt2.tab contains a client channel definition table and is stored on a system that is different to the one on which the application is running. If the file can be accessed using the FTP protocol, the application can create a URL object in the following way:

```
java.net.URL chanTab2 = new URL("ftp://ftp.server/admdata/ccdt2.tab");
```

After the application has created a URL object, the application can create an MQQueueManager object using one of the constructors that takes a URL object as a parameter. Here is an example:

```
MQQueueManager mars = new MQQueueManager("MARS", chanTab2);
```

This statement causes the WebSphere MQ base Java client to access the client channel definition table identified by the URL object chanTab2, search the table for a suitable client connection channel definition, and then use the channel definition to start an MQI channel to the queue manager called MARS.

Note the following points that apply if an application uses a client channel definition table:

- When the application creates an MQQueueManager object using a constructor that takes a URL object as a parameter, no channel name must be set in the MQEnvironment class, either as a field or as an environment property. If a channel name is set, the WebSphere MQ base Java client throws an MQException. The field or environment property specifying the channel name is considered to be set if its value is anything other than null, an empty string, or a string containing all blank characters.
- The queueManagerName parameter on the MQQueueManager constructor can have one of the following values:
  - The name of a queue manager
  - An asterisk (*) followed by the name of a queue manager group
  - An asterisk (*)
  - Null, an empty string, or a string containing all blank characters

  These are the same values that can be used for the *QMgrName* parameter on an MQCONN call issued by a client application that is using Message Queue Interface (MQI). For more information about the meaning of these values therefore, see the *WebSphere MQ Application Programming Reference* and *WebSphere MQ Clients*. The way that the WebSphere MQ base Java client uses the queueManagerName parameter to search the client channel definition table is also as described in these books. If your application uses connection pooling, see also "Controlling the default connection pool" on page 82.
- When the WebSphere MQ base Java client finds a suitable client connection channel definition in the client channel definition table, it uses only the information extracted from this channel definition to start an MQI channel. Any channel related fields or environment properties that the application might have set in the MQEnvironment class are ignored.

  In particular, note the following points if you are using Secure Sockets Layer (SSL):
  - An MQI channel uses SSL only if the channel definition extracted from the client channel definition table specifies the name of a CipherSpec supported by the WebSphere MQ base Java client.
  - A client channel definition table also contains information about the location of Lightweight Directory Access Protocol (LDAP) servers that hold certificate revocation lists (CRLs). The WebSphere MQ base Java client uses only this information to access LDAP servers that hold CRLs.

  For more information about using SSL with a client channel definition table, see *WebSphere MQ Clients*.

  Note also the following points if you are using channel exits:
  - An MQI channel uses only the channel exits and associated user data specified by the channel definition extracted from the client channel definition table.
  - A channel definition extracted from a client channel definition table can specify channel exits that are written in Java. This means, for example, that the SCYEXIT parameter on the DEFINE CHANNEL command to create a client connection channel definition can specify the name of a class that

implements the MQSecurityExit interface. Similarly, the SENDEXIT parameter can specify the name of a class that implements the MQSendExit interface, and the RCVEXIT parameter can specify the name of a class that implements the MQReceiveExit interface. For more information about how to write a channel exit in Java, see "Using channel exits" on page 77.

The use of channel exits written in a language other than Java is also supported. For information about how to specify the SCYEXIT, SENDEXIT, and RCVEXIT parameters on the DEFINE CHANNEL command for channel exits written in another language, see the *WebSphere MQ Script (MQSC) Command Reference*.

Using the WebSphere MQ Explorer

When using the WebSphere MQ Explorer, you can connect to a remote queue manager using a channel definition table. You can also set channel exits for remote queue managers, but the following restrictions apply:

- Java exits cannot be loaded from a .jar or .zip file. They must exist uncompressed in the appropriate directory structure.
- Native (non-Java) exits can be loaded but the WebSphere MQ JMS/Java client must be installed, as well as the Explorer. If the JMS/Java client is not installed, the native exit will not load, because a native utility library from the JMS/Java client is neded to load the native exit.

Specifying a range of ports for client connections

When a WebSphere MQ base Java application attempts to connect to a WebSphere MQ queue manager in client mode, a firewall might allow only those connections that originate from specified ports or range of ports. In this situation, you can specify a port, or a range of points, that the application can bind to. You can do this in either of the following ways:

- You can set the localAddressSetting field in the MQEnvironment class. Here is an example:

```
MQEnvironment.localAddressSetting = "9.20.0.1(2000,3000)";
```

- You can set the environment property MQC.LOCAL_ADDRESS_PROPERTY. Here is an example:

```
(MQEnvironment.properties).put(MQC.LOCAL_ADDRESS_PROPERTY,
                               "9.20.0.1(2000,3000)");
```

In each of these examples, when the application connects to a queue manager subsequently, the application binds to a local IP address and port number in the range 9.20.0.1(2000) to 9.20.0.1(3000).

In a system with more than one network interface, you can also use the localAddressSetting field, or the environment property MQC.LOCAL_ADDRESS_PROPERTY, to specify which network interface must be used for a connection.

Connection errors might occur if you restrict the range of ports. If an error occurs, an MQException is thrown containing the WebSphere MQ reason code MQRC_Q_MGR_NOT_AVAILABLE and the following message:

```
Socket connection attempt refused due to LOCAL_ADDRESS_PROPERTY restrictions
```

An error might occur if all the ports in the specified range are in use, or if the specified IP address, host name, or port number is not valid (a negative port number, for example).

Accessing queues and processes

To access queues and processes, use the MQQueueManager class. The MQOD (object descriptor structure) is collapsed into the parameters of these methods. For example, to open a queue on a queue manager called queueManager, use the following code:

```
MQQueue queue = queueManager.accessQueue("qName",
                                         MQC.MQOO_OUTPUT,
                                         "qMgrName",
                                         "dynamicQName",
                                         "altUserId");
```

The *options* parameter is the same as the Options parameter in the MQOPEN call.

The accessQueue method returns a new object of class MQQueue.

When you have finished using the queue, use the close() method to close it, as in the following example:

```
queue.close();
```

With WebSphere MQ classes for Java, you can also create a queue by using the MQQueue constructor. The parameters are exactly the same as for the accessQueue method, with the addition of a queue manager parameter. For example:

```
MQQueue queue = new MQQueue(queueManager,
                            "qName",
                            MQC.MQOO_OUTPUT,
                            "qMgrName",
                            "dynamicQName",
                            "altUserId");
```

Constructing a queue object in this way enables you to write your own subclasses of MQQueue.

To access a process, use the accessProcess method in place of accessQueue. This method does not have a *dynamic queue name* parameter, because this does not apply to processes.

The accessProcess method returns a new object of class MQProcess.

When you have finished using the process object, use the close() method to close it, as in the following example:

```
process.close();
```

With WebSphere MQ classes for Java, you can also create a process by using the MQProcess constructor. The parameters are exactly the same as for the accessProcess method, with the addition of a queue manager parameter. Constructing a process object in this way enables you to write your own subclasses of MQProcess.

Handling messages

Put messages onto queues using the put() method of the MQQueue class. You get messages from queues using the get() method of the MQQueue class. Unlike the procedural interface, where MQPUT and MQGET put and get arrays of bytes, the Java programming language puts and gets instances of the MQMessage class. The

MQMessage class encapsulates the data buffer that contains the actual message data, together with all the MQMD (message descriptor) parameters that describe that message.

To build a new message, create a new instance of the MQMessage class, and use the writeXXX methods to put data into the message buffer.

When the new message instance is created, all the MQMD parameters are automatically set to their default values, as defined in the *WebSphere MQ Application Programming Reference*. The put() method of MQQueue also takes an instance of the MQPutMessageOptions class as a parameter. This class represents the MQPMO structure. The following example creates a message and puts it onto a queue:

```
// Build a new message containing my age followed by my name
MQMessage myMessage = new MQMessage();
myMessage.writeInt(25);

String name = "Charlie Jordan";
myMessage.writeInt(name.length());
myMessage.writeBytes(name);

// Use the default put message options...
MQPutMessageOptions pmo = new MQPutMessageOptions();

// put the message!
queue.put(myMessage,pmo);
```

The get() method of MQQueue returns a new instance of MQMessage, which represents the message just taken from the queue. It also takes an instance of the MQGetMessageOptions class as a parameter. This class represents the MQGMO structure.

You do not need to specify a maximum message size, because the get() method automatically adjusts the size of its internal buffer to fit the incoming message. Use the readXXX methods of the MQMessage class to access the data in the returned message.

The following example shows how to get a message from a queue:

```
// Get a message from the queue
MQMessage theMessage    = new MQMessage();
MQGetMessageOptions gmo = new MQGetMessageOptions();
queue.get(theMessage,gmo);  // has default values

// Extract the message data
int age = theMessage.readInt();
int strLen = theMessage.readInt();
byte[] strData = new byte[strLen];
theMessage.readFully(strData,0,strLen);
String name = new String(strData,0);
```

You can alter the number format that the read and write methods use by setting the *encoding* member variable.

You can alter the character set to use for reading and writing strings by setting the *characterSet* member variable.

See “MQMessage” on page 147 for more details.

Note: The writeUTF() method of MQMessage automatically encodes the length of the string as well as the Unicode bytes it contains. When your message will be read by another Java program (using readUTF()), this is the simplest way to send string information.

Handling errors

Methods in the Java interface do not return a completion code and reason code. Instead, they throw an exception whenever the completion code and reason code resulting from a WebSphere MQ call are not both zero. This simplifies the program logic so that you do not have to check the return codes after each call to WebSphere MQ. You can decide at which points in your program you want to deal with the possibility of failure. At these points, you can surround your code with `try` and `catch` blocks, as in the following example:

```
try {
    myQueue.put(messageA,putMessageOptionsA);
    myQueue.put(messageB,putMessageOptionsB);
}
catch (MQException ex) {
    // This block of code is only executed if one of
    // the two put methods gave rise to a non-zero
    // completion code or reason code.
    System.out.println("An error occurred during the put operation:" +
                           "CC = " + ex.completionCode +
                           "RC = " + ex.reasonCode);
    System.out.println("Cause exception:" + ex.getCause() );
}
```

The WebSphere MQ call reason codes reported back in Java exceptions are documented in a chapter called “Return Codes” in the *WebSphere MQ Application Programming Reference*.

Exceptions that are thrown while a WebSphere MQ base Java application is running are also written to the log. However, an application can call the MQException.logExclude() method to prevent exceptions associated with a specific reason code from being logged. You might want to do this in situations where you expect many exceptions associated with a specific reason code to be thrown, and you do not want the log to be filled with these exceptions. For example, if your application attempts to get a message from a queue each time it iterates around a loop and, for most of these attempts, you expect no suitable message to be on the queue, you might want to prevent exceptions associated the reason code MQRC_NO_MSG_AVAILABLE from being logged. If an application has previously prevented exceptions associated with a specific reason code from being logged, it can allow these exceptions to be logged again by calling the method MQException.logInclude().

Sometimes the reason code does not convey all details associated with the error. This can occur if WebSphere MQ uses services provided by another product (for example, a JSSE implementation) that throws a java.lang.Exception to WebSphere MQ Java. In this case, the method MQException.getCause() retrieves the underlying java.lang.Exception that caused the error.

Getting and setting attribute values

For many of the common attributes, the classes MQManagedObject, MQQueue, MQProcess, and MQQueueManager contain getXXX() and setXXX() methods. These methods allow you to get and set their attribute values. Note that for MQQueue, the methods work only if you specify the appropriate inquire and set flags when you open the queue.

For less common attributes, the MQQueueManager, MQQueue, and MQProcess classes all inherit from a class called MQManagedObject. This class defines the inquire() and set() interfaces.

When you create a new queue manager object by using the *new* operator, it is automatically opened for inquire. When you use the accessProcess() method to access a process object, that object is automatically opened for inquire. When you use the accessQueue() method to access a queue object, that object is *not* automatically opened for either inquire or set operations. This is because adding these options automatically can cause problems with some types of remote queues. To use the inquire, set, getXXX, and setXXX methods on a queue, you must specify the appropriate inquire and set flags in the openOptions parameter of the accessQueue() method.

The inquire and set methods take three parameters:

- selectors array
- intAttrs array
- charAttrs array

You do not need the SelectorCount, IntAttrCount, and CharAttrLength parameters that are found in MQINQ, because the length of an array in Java is always known. The following example shows how to make an inquiry on a queue:

```
// inquire on a queue
final static int MQIA_DEF_PRIORITY = 6;
final static int MQCA_Q_DESC = 2013;
final static int MQ_Q_DESC_LENGTH = 64;

int[] selectors  = new int[2];
int[] intAttrs   = new int[1];
byte[] charAttrs = new byte[MQ_Q_DESC_LENGTH]

selectors[0] = MQIA_DEF_PRIORITY;
selectors[1] = MQCA_Q_DESC;

queue.inquire(selectors,intAttrs,charAttrs);

System.out.println("Default Priority = " + intAttrs[0]);
System.out.println("Description : " + new String(charAttrs,0));
```

Multithreaded programs

Multithreaded programs are hard to avoid in Java. Consider a simple program that connects to a queue manager and opens a queue at startup. The program displays a single button on the screen. When a user presses that button, the program fetches a message from the queue.

The Java runtime environment is inherently multithreaded. Therefore, your application initialization occurs in one thread, and the code that executes in response to the button press executes in a separate thread (the user interface thread).

With the C based WebSphere MQ client, this would cause a problem, because handles cannot be shared across multiple threads. WebSphere MQ classes for Java relaxes this constraint, allowing a queue manager object (and its associated queue and process objects) to be shared across multiple threads.

The implementation of WebSphere MQ classes for Java ensures that, for a given connection (MQQueueManager object instance), all access to the target WebSphere MQ queue manager is synchronized. A thread that wants to issue a call to a queue manager is blocked until all other calls in progress for that connection are complete. If you require simultaneous access to the same queue manager from multiple threads within your program, create a new MQQueueManager object for each thread that requires concurrent access. (This is equivalent to issuing a separate MQCONN call for each thread.)

Using channel exits

WebSphere MQ classes for Java allows you to provide your own send, receive, and security exits.

To implement an exit, you define a new Java class that implements the appropriate interface. Three exit interfaces are defined in the WebSphere MQ package:

- MQSendExit
- MQReceiveExit
- MQSecurityExit

Note: Channel exits are supported for client connections only; they are not supported for bindings connections.

Any SSL encryption defined for a connection is performed *after* the send exit has been invoked. Similarly, decryption is performed *before* the receive or security exits are invoked.

The following sample defines a class that implements all three:

```
class MyMQExits implements MQSendExit, MQReceiveExit, MQSecurityExit {

  // This method comes from the send exit
  public byte[] sendExit(MQChannelExit channelExitParms,
                         MQChannelDefinition channelDefParms,
                         byte agentBuffer[])
  {
     // fill in the body of the send exit here
  }


  // This method comes from the receive exit
  public byte[] receiveExit(MQChannelExit channelExitParms,
                            MQChannelDefinition channelDefParms,
                            byte agentBuffer[])
  {
     // fill in the body of the receive exit here
  }


  // This method comes from the security exit
  public byte[] securityExit(MQChannelExit channelExitParms,
                             MQChannelDefinition channelDefParms,
                             byte agentBuffer[])
  {
```

```
    // fill in the body of the security exit here
  }

}
```

Each exit is passed an MQChannelExit and an MQChannelDefinition object instance. These objects represent the MQCXP and MQCD structures defined in the procedural interface.

For a send exit, the *agentBuffer* parameter contains the data that is about to be sent. For a receive exit or a security exit, the *agentBuffer* parameter contains the data that has just been received. You do not need a length parameter, because the expression agentBuffer.length indicates the length of the array. You cannot change the size of the *agentBuffer* in an exit .

For the send and security exits, your exit code should return the byte array that you want to send to the server. For a receive exit, your exit code must return the modified data that you want WebSphere MQ classes for Java to interpret.

The simplest possible exit body is:

```
{
  return agentBuffer;
}
```

If your program is to run as a downloaded Java applet, the security restrictions that apply mean that you cannot read or write any local files. If your exit needs a configuration file, you can place the file on the Web and use the java.net.URL class to download it and examine its contents.

Having defined a class that implements the MQSecurityExit interface, an application can use the security exit by assigning an instance of the class to the MQEnvironment.securityExit field before creating an MQQueueManager object. An application can use a send or a receive exit in a similar way. For example, the following code fragment shows you how to use the security, send, and receive exits that are implemented in the class MyMQExits, which was defined previously:

```
MyMQExits myexits = new MyMQExits();
MQEnvironment.securityExit = myexits;
MQEnvironment.sendExit = myexits;
MQEnvironment.receiveExit = myexits;
:
MQQueueManager jupiter = new MQQueueManager("JUPITER");
```

If an application connects to a queue manager by setting channel related fields or environment properties in the MQEnvironment class, no user data can be passed to channel exit classes when they are called. However, if an application uses a client channel definition table to connect to a queue manager, any user data specified in a client connection channel definition is passed to channel exit classes when they are called. For more information about using a client channel definition table, see "Using a client channel definition table" on page 70.

If you package channel exit classes in separate JAR files or class files, you must make sure that WebSphere MQ base Java can locate these files. For an application that is not running in an application server, you can do this in either of the following ways:

- Make sure that the files can be found in the class path of the JVM.
- Store the files in the directory shown in Table 13 on page 79.

For an application that is running in an application server, you must store the files in the directory shown in Table 13.

Table 13. The directory for channel exit programs

Platform	Directory
AIX, HP-UX, Linux, and Solaris	/var/mqm/exits (32-bit channel exit programs) /var/mqm/exits64 (64-bit channel exit programs)
Windows	*install_data_dir*\exits
Note: *install_data_dir* is the directory that you chose for the WebSphere MQ data files during installation. The default directory is C:\Program Files\IBM\WebSphere MQ.	

Using channel exits not written in Java

A WebSphere MQ base Java application can use channel exit programs that are written in C or C++. Three classes are provided for this purpose:

- MQExternalSecurityExit, which implements the MQSecurityExit interface
- MQExternalSendExit, which implements the MQSendExit interface
- MQExternalReceiveExit, which implements the MQReceiveExit interface

To use a security exit that is not written in Java, an application must first create an MQExternalSecurityExit object. The application specifies, as parameters on the MQExternalSecurityExit constructor, the name of the library containing the security exit, the name of the entry point for the security exit, and the user data to be passed to the security exit when it is called. The application can then assign the MQExternalSecurityExit object to the MQEnvironment.securityExit field before creating an MQQueueManager object. The MQExternalSecurityExit object contains all the information required to construct the MQCXP and MQCD structures that are passed to the security exit when it is called.

An application can use a send or a receive exit that is not written in Java in a way similar to that just described for a security exit.

If an application connects to a queue manager by setting channel related fields or environment properties in the MQEnvironment class, no user data can be passed to channel exit programs when they are called. If an application uses a client channel definition table to connect to a queue manager, any user data specified in a client connection channel definition is passed to channel exit programs when they are called. On i5/OS, however, no user data can be passed to channel exit programs, even if the application uses a client channel definition table. For more information about using a client channel definition table, see “Using a client channel definition table” on page 70.

You must store channel exit programs that are not written in Java in the directory shown in Table 13.

For information about how to write a channel exit in C or C++, see *WebSphere MQ Intercommunication*.

Using a sequence of channel send or receive exits

A WebSphere MQ base Java application can use a sequence of channel send or receive exits that are run in succession. Two classes are provided for this purpose:

- MQSendExitChain, which implements the MQSendExit interface
- MQReceiveExitChain, which implements the MQReceiveExit interface

To use a sequence of send exits, an application must create a list of objects, where each object is one of the following:

- An instance of a user defined class that implements the MQSendExit interface (for a send exit written in Java)
- An instance of the MQExternalSendExit class (for a send exit not written in Java)
- An instance of the MQSendExitChain class

The application creates an MQSendExitChain object by passing this list of objects as a parameter on the constructor. The application can then assign the MQSendExitChain object to the MQEnvironment.sendExit field before creating an MQQueueManager object.

The context of information passed to exits is solely within the domain of the exits. For example, if a Java exit and a C exit are chained, the C exit will not be aware that there is also a Java exit in effect.

An application can use a sequence of receive exits in a way similar to that just described for a sequence of send exits.

Channel compression

Compressing the data that flows on a WebSphere MQ channel can improve the performance of the channel and reduce network traffic. Using function supplied with WebSphere MQ, you can compress the data that flows on message channels and MQI channels and, on either type of channel, you can compress header data and message data independently of each other. By default, no data is compressed on a channel. For a full description of channel compression, including how it is implemented in WebSphere MQ, see *WebSphere MQ Intercommunication*, for message channels, and *WebSphere MQ Clients*, for MQI channels.

A WebSphere MQ base Java application specifies the techniques that can be used for compressing header or message data on a client connection by creating a java.util.Collection object. Each compression technique is an Integer object in the collection, and the order in which the application adds the compression techniques to the collection is the order in which the compression techniques are negotiated with the queue manager when the client connection starts. The application can then assign the collection to the hdrCompList field, for header data, or the msgCompList field, for message data, in the MQEnvironment class. When the application is ready, it can start the client connection by creating an MQQueueManager object.

The following code fragments illustrate the approach just described. The first code fragment shows you how to implement header data compression:

```
Collection headerComp = new Vector();
headerComp.add(new Integer(MQC.MQCOMPRESS_SYSTEM));
:
MQEnvironment.hdrCompList = headerComp;
:
MQQueueManager qMgr = new MQQueueManager(QM);
```

The second code fragment shows you how to implement message data compression:

```
Collection msgComp = new Vector();
msgComp.add(new Integer(MQC.MQCOMPRESS_RLE));
msgComp.add(new Integer(MQC.MQCOMPRESS_ZLIBHIGH));
:
```

```
MQEnvironment.msgCompList = msgComp;
:
MQQueueManager qMgr = new MQQueueManager(QM);
```

In the second example, the compression techniques are negotiated in the order RLE, then ZLIBHIGH, when the client connection starts. The compression technique that is selected cannot be changed during the lifetime of the MQQueueManager object.

The compression techniques for header and message data that are supported by both the client and the queue manager on a client connection are passed to a channel exit as collections in the hdrCompList and msgCompList fields respectively of an MQChannelDefinition object. The actual techniques that are currently being used for compressing header and message data on a client connection are passed to a channel exit in the CurHdrCompression and CurMsgCompression fields respectively of an MQChannelExit object.

Note that, if compression is used on a client connection, the data is compressed before any channel send exits are processed and decompressed after any channel receive exits are processed. The data passed to send and receive exits is therefore in a compressed state.

For more information about specifying compression techniques, and about which compression techniques are available, see “MQEnvironment” on page 118 and “MQC” on page 199.

Connection pooling

WebSphere MQ classes for Java provides additional support for applications that deal with multiple connections to WebSphere MQ queue managers. When a connection is no longer required, instead of destroying it, it can be pooled and later reused. This can provide a substantial performance enhancement for applications and middleware that connect serially to arbitrary queue managers.

WebSphere MQ provides a default connection pool. Applications can activate or deactivate this connection pool by registering and deregistering tokens through the MQEnvironment class. If the pool is active when WebSphere MQ base Java constructs an MQQueueManager object, it searches this default pool and reuses any suitable connection. When an MQQueueManager.disconnect() call occurs, the underlying connection is returned to the pool.

Alternatively, applications can construct an MQSimpleConnectionManager connection pool for a particular use. Then, the application can either specify that pool during construction of an MQQueueManager object, or pass that pool to MQEnvironment for use as the default connection pool.

To prevent connections from using too much resource, you can limit the total number of connections that an MQSimpleConnectionManager object can handle, and you can limit the size of the connection pool. Setting limits is useful if there are conflicting demands for connections within a JVM.

By default, the getMaxConnections() method returns the value zero, which means that there is no limit to the number of connections that the MQSimpleConnectionManager object can handle. You can set a limit by using the setMaxConnections() method. If you set a limit and the limit is reached, a request

for a further connection might cause an MQException to be thrown, with a reason code of MQRC_MAX_CONNS_LIMIT_REACHED.

Also, WebSphere MQ base Java provides a partial implementation of the Java 2 Platform, Enterprise Edition (J2EE) Connector Architecture, Version 1.0. Applications running under a Java 2 v1.3 JVM with JAAS 1.0 (Java Authentication and Authorization Service) can provide their own connection pool by implementing the javax.resource.spi.ConnectionManager interface. Again, this interface can be specified on the MQQueueManager constructor, or specified as the default connection pool.

Controlling the default connection pool

Consider the following example application, MQApp1:

```
import com.ibm.mq.*;
public class MQApp1
{
      public static void main(String[] args) throws MQException
      {
         for (int i=0; i<args.length; i++) {
            MQQueueManager qmgr=new MQQueueManager(args[i]);
            :
            : (do something with qmgr)
            :
            qmgr.disconnect();
         }
      }
}
```

MQApp1 takes a list of local queue managers from the command line, connects to each in turn, and performs some operation. However, when the command line lists the same queue manager many times, it is more efficient to connect only once, and to reuse that connection many times.

WebSphere MQ base Java provides a default connection pool that you can use to do this. To enable the pool, use one of the MQEnvironment.addConnectionPoolToken() methods. To disable the pool, use MQEnvironment.removeConnectionPoolToken().

The following example application, MQApp2, is functionally identical to MQApp1, but connects only once to each queue manager.

```
import com.ibm.mq.*;
public class MQApp2
{
      public static void main(String[] args) throws MQException
      {
         MQPoolToken token=MQEnvironment.addConnectionPoolToken();

         for (int i=0; i<args.length; i++) {
            MQQueueManager qmgr=new MQQueueManager(args[i]);
            :
            : (do something with qmgr)
            :
            qmgr.disconnect();
         }

         MQEnvironment.removeConnectionPoolToken(token);

      }
}
```

The first bold line activates the default connection pool by registering an MQPoolToken object with MQEnvironment.

The MQQueueManager constructor now searches this pool for an appropriate connection and only creates a connection to the queue manager if it cannot find an existing one. The qmgr.disconnect() call returns the connection to the pool for later reuse. These API calls are the same as the sample application MQApp1.

The second highlighted line deactivates the default connection pool, which destroys any queue manager connections stored in the pool. This is important because otherwise the application would terminate with a number of live queue manager connections in the pool. This situation could cause errors that would appear in the queue manager logs.

If an application uses a client channel definition table to connect to a queue manager, the MQQueueManager constructor first searches the table for a suitable client connection channel definition. If one is found, the constructor searches the default connection pool for a connection that can be used for the channel. If the constructor cannot find a suitable connection in the pool, it then searches the client channel definition table for the next suitable client connection channel definition, and proceeds as described previously. If the constructor completes its search of the client channel definition table and fails to find any suitable connection in the pool, the constructor starts a second search of the table. During this search, the constructor tries to create a new connection for each suitable client connection channel definition in turn, and uses the first connection that it manages to create.

The default connection pool stores a maximum of ten unused connections, and keeps unused connections active for a maximum of five minutes. The application can alter this (for details, see “Supplying a different connection pool” on page 84).

Instead of using MQEnvironment to supply an MQPoolToken, the application can construct its own:

```
MQPoolToken token=new MQPoolToken();
MQEnvironment.addConnectionPoolToken(token);
```

Some applications or middleware vendors provide subclasses of MQPoolToken in order to pass information to a custom connection pool. They can be constructed and passed to addConnectionPoolToken() in this way so that extra information can be passed to the connection pool.

The default connection pool and multiple components

MQEnvironment holds a static set of registered MQPoolToken objects. To add or remove MQPoolTokens from this set, use the following methods:

- MQEnvironment.addConnectionPoolToken()
- MQEnvironment.removeConnectionPoolToken()

An application might consist of many components that exist independently and perform work using a queue manager. In such an application, each component should add an MQPoolToken to the MQEnvironment set for its lifetime.

For example, the example application MQApp3 creates ten threads and starts each one. Each thread registers its own MQPoolToken, waits for a length of time, then connects to the queue manager. After the thread disconnects, it removes its own MQPoolToken.

The default connection pool remains active while there is at least one token in the set of MQPoolTokens, so it will remain active for the duration of this application. The application does not need to keep a master object in overall control of the threads.

```
import com.ibm.mq.*;
public class MQApp3
{
      public static void main(String[] args)
      {
         for (int i=0; i<10; i++) {
            MQApp3_Thread thread=new MQApp3_Thread(i*60000);
            thread.start();
         }
      }
}

class MQApp3_Thread extends Thread
{
      long time;

      public MQApp3_Thread(long time)
      {
         this.time=time;
      }

      public synchronized void run()
      {
         MQPoolToken token=MQEnvironment.addConnectionPoolToken();
         try {
            wait(time);
            MQQueueManager qmgr=new MQQueueManager("my.qmgr.1");
            :
            : (do something with qmgr)
            :
            qmgr.disconnect();
         }
         catch (MQException mqe) {System.err.println("Error occurred!");}
         catch (InterruptedException ie) {}

         MQEnvironment.removeConnectionPoolToken(token);
      }
}
```

Supplying a different connection pool

This section describes how to use the class **com.ibm.mq.MQSimpleConnectionManager** to supply a different connection pool. This class provides basic facilities for connection pooling, and applications can use this class to customize the behavior of the pool.

Once it is instantiated, an MQSimpleConnectionManager can be specified on the MQQueueManager constructor. The MQSimpleConnectionManager then manages the connection that underlies the constructed MQQueueManager. If the MQSimpleConnectionManager contains a suitable pooled connection, that connection is reused and returned to the MQSimpleConnectionManager after an MQQueueManager.disconnect() call.

The following code fragment demonstrates this behavior:

```
MQSimpleConnectionManager myConnMan=new MQSimpleConnectionManager();
myConnMan.setActive(MQSimpleConnectionManager.MODE_ACTIVE);
MQQueueManager qmgr=new MQQueueManager("my.qmgr.1", myConnMan);
:
: (do something with qmgr)
```

```
:
qmgr.disconnect();

MQQueueManager qmgr2=new MQQueueManager("my.qmgr.1", myConnMan);
:
: (do something with qmgr2)
:
qmgr2.disconnect();
myConnMan.setActive(MQSimpleConnectionManager.MODE_INACTIVE);
```

The connection that is forged during the first MQQueueManager constructor is stored in myConnMan after the qmgr.disconnect() call. The connection is then reused during the second call to the MQQueueManager constructor.

The second line enables the MQSimpleConnectionManager. The last line disables MQSimpleConnectionManager, destroying any connections held in the pool. An MQSimpleConnectionManager is, by default, in MODE_AUTO, which is described later in this section.

An MQSimpleConnectionManager allocates connections on a most-recently-used basis, and destroys connections on a least-recently-used basis. By default, a connection is destroyed if it has not been used for five minutes, or if there are more than ten unused connections in the pool. You can alter these values by calling MQSimpleConnectionManager.setTimeout().

You can also set up an MQSimpleConnectionManager for use as the default connection pool, to be used when no Connection Manager is supplied on the MQQueueManager constructor.

The following application demonstrates this:

```
import com.ibm.mq.*;
public class MQApp4
{
      public static void main(String []args)
      {
         MQSimpleConnectionManager myConnMan=new MQSimpleConnectionManager();
         myConnMan.setActive(MQSimpleConnectionManager.MODE_AUTO);
         myConnMan.setTimeout(3600000);
         myConnMan.setMaxConnections(75);
         myConnMan.setMaxUnusedConnections(50);
         MQEnvironment.setDefaultConnectionManager(myConnMan);
         MQApp3.main(args);
      }
}
```

The bold lines create and configure an MQSimpleConnectionManager object. The configuration does the following:

- Ends connections that are not used for an hour
- Limits the number of connections managed by myConnMan to 75
- Limits the number of unused connections in the pool to 50
- Sets MODE_AUTO, which is the default. This means that the pool is active only if it is the default connection manager, and there is at least one token in the set of MQPoolTokens held by MQEnvironment.

The new MQSimpleConnectionManager is then set as the default connection manager.

In the last line, the application calls MQApp3.main(). This runs a number of threads, where each thread uses WebSphere MQ independently. These threads use myConnMan when they forge connections.

Supplying your own ConnectionManager

Under Java 2 v1.3, with JAAS 1.0 installed, applications and middleware providers can provide alternative implementations of connection pools. WebSphere MQ base Java provides a partial implementation of the J2EE Connector Architecture. Implementations of **javax.resource.spi.ConnectionManager** can either be used as the default Connection Manager or be specified on the MQQueueManager constructor.

WebSphere MQ base Java complies with the Connection Management contract of the J2EE Connector Architecture. Read this section in conjunction with the Connection Management contract of the J2EE Connector Architecture (refer to Sun's Web site at `http://java.sun.com`).

The ConnectionManager interface defines only one method:

```
package javax.resource.spi;
public interface ConnectionManager {
      Object allocateConnection(ManagedConnectionFactory mcf,
                                ConnectionRequestInfo cxRequestInfo);
}
```

The MQQueueManager constructor calls allocateConnection on the appropriate ConnectionManager. It passes appropriate implementations of ManagedConnectionFactory and ConnectionRequestInfo as parameters to describe the connection required.

The ConnectionManager searches its pool for a javax.resource.spi.ManagedConnection object that has been created with identical ManagedConnectionFactory and ConnectionRequestInfo objects. If the ConnectionManager finds any suitable ManagedConnection objects, it creates a java.util.Set that contains the candidate ManagedConnections. Then, the ConnectionManager calls the following:

```
ManagedConnection mc=mcf.matchManagedConnections(connectionSet, subject,
cxRequestInfo);
```

The WebSphere MQ implementation of ManagedConnectionFactory ignores the subject parameter. This method selects and returns a suitable ManagedConnection from the set, or returns null if it does not find a suitable ManagedConnection. If there is not a suitable ManagedConnection in the pool, the ConnectionManager can create one by using:

```
ManagedConnection mc=mcf.createManagedConnection(subject, cxRequestInfo);
```

Again, the subject parameter is ignored. This method connects to a WebSphere MQ queue manager and returns an implementation of javax.resource.spi.ManagedConnection that represents the newly-forged connection. Once the ConnectionManager has obtained a ManagedConnection (either from the pool or freshly created), it creates a connection handle using:

```
Object handle=mc.getConnection(subject, cxRequestInfo);
```

This connection handle can be returned from allocateConnection().

A ConnectionManager must register an interest in the ManagedConnection through:

```
mc.addConnectionEventListener()
```

The ConnectionEventListener is notified if a severe error occurs on the connection, or when MQQueueManager.disconnect() is called. When MQQueueManager.disconnect() is called, the ConnectionEventListener can do either of the following:

- Reset the ManagedConnection using the mc.cleanup() call, then return the ManagedConnection to the pool
- Destroy the ManagedConnection using the mc.destroy() call

If the ConnectionManager is the default ConnectionManager, it can also register an interest in the state of the MQEnvironment-managed set of MQPoolTokens. To do so, first construct an MQPoolServices object, then register an MQPoolServicesEventListener object with the MQPoolServices object:

```
MQPoolServices mqps=new MQPoolServices();
mqps.addMQPoolServicesEventListener(listener);
```

The listener is notified when an MQPoolToken is added or removed from the set, or when the default ConnectionManager changes. The MQPoolServices object also provides a way to query the current size of the set of MQPoolTokens.

JTA/JDBC coordination using WebSphere MQ base Java

WebSphere MQ base Java supports the MQQueueManager.begin() method, which allows WebSphere MQ to act as a coordinator for a database which provides a JDBC type 2 or JDBC type 4 compliant driver. Currently this support is available on AIX, HP-UX, Solaris, and Windows with DB2® or Oracle databases.

Configuring JTA/JDBC coordination

In order to use the XA-JTA support, you must use the special JTA switch library. The method for using this library varies depending on whether you are using Windows or one of the other platforms.

Configuring on Windows

On Windows systems, the new XA library is supplied as a complete DLL. The name of this DLL is `jdbcxxx.dll` where `xxx` indicates the database for which the switch library has been compiled. This library is in the `Java\lib\jdbc` directory of your WebSphere MQ base Java installation.

Configuring on the other platforms

For each database management system, WebSphere MQ provides two object files. You must link one object file to create a 32-bit switch library, and link the other object file to create a 64-bit switch library. For DB2, the name of each object file is jdbcdb2.o and, for Oracle, the name of each object file is jdbcora.o.

You must link each object file using the appropriate makefile supplied with WebSphere MQ. A switch library requires other libraries, which might be stored in different locations on different systems. However, a switch library cannot use the library path environment variable to locate these libraries because the switch library is loaded by the queue manager, which runs in a setuid environment. The supplied makefile therefore ensures that a switch library contains the fully qualified path names of these libraries.

To create a switch library, enter a **make** command with the following format. To create a 32-bit switch library, enter the command in the /java/lib/jdbc directory of

your WebSphere MQ installation. To create a 64-bit switch library, enter the command in the /java/lib64/jdbc directory.

```
make DBMS
```

where *DBMS* is the database management system for which you are creating the switch library. The valid values are `db2` for DB2 and `oracle` for Oracle.

Here is an example of a **make** command:

```
make db2
```

Note the following points:

- To run 32-bit applications, you must create both a 32-bit and a 64-bit switch library for each database management system that you are using. To run 64-bit applications, you need create only a 64-bit switch library. For DB2, the name of each switch library is jdbcdb2 and, for Oracle, the name of each switch library is jdbcora. The makefiles ensure that 32-bit and 64-bit switch libraries are stored in different WebSphere MQ directories. A 32-bit switch library is stored in the /var/mqm/exits directory, and a 64-bit switch library is stored in the /var/mqm/exits64 directory.
- Because you can install Oracle anywhere on a system, the makefiles use the ORACLE_HOME environment variable to locate where Oracle is installed.

After you have created the switch libraries for DB2, Oracle, or both, you must declare them to your queue manager. If the queue manager configuration file (qm.ini) already contains XAResourceManger stanzas for DB2 or Oracle databases, you must replace the SwitchFile entry in each stanza by one of the following:

For a DB2 database

```
SwitchFile=jdbcdb2
```

For an Oracle database

```
SwitchFile=jdbcora
```

Do not specify the fully qualified path name of either the 32-bit or 64-bit switch library. Specify only the name of the library.

If the queue manager configuration file does not already contain XAResourceManager stanzas for DB2 or Oracle databases, or if you want to add additional XAResourceManager stanzas, see the *WebSphere MQ System Administration Guide* for information about how to construct an XAResourceManager stanza. However, each SwitchFile entry in a new XAResourceManger stanza must be exactly as described previously for a DB2 or Oracle database. You must also include the entry `ThreadOfControl=PROCESS`.

After you have updated the queue manager configuration file, and made sure that all appropriate database environment variables have been set, you can restart the queue manager.

Using JTA/JDBC coordination

The basic sequence of API calls for a user application is:

```
qMgr = new MQQueueManager("QM1")
Connection con = qMgr.getJDBCConnection( xads );
qMgr.begin()

< Perform MQ and DB operations to be grouped in a unit of work >
```

```
qMgr.commit() or qMgr.backout();
con.close()
qMgr.disconnect()
```

xads in the getJDBCConnection call is a database-specific implementation of the XADataSource interface, which defines the details of the database to connect to. See the documentation for your database to determine how to create an appropriate XADataSource object to pass into getJDBCConnection.

You must also update your CLASSPATH with the appropriate database-specific jar files for performing JDBC work.

If you need to connect to multiple databases, you might have to call getJDBCConnection several times to perform the transaction across several different connections.

There are two forms of the getJDBCConnection, reflecting the two forms of XADataSource.getXAConnection:

```
public java.sql.Connection getJDBCConnection(javax.sql.XADataSource xads)
  throws MQException, SQLException, Exception

public java.sql.Connection getJDBCConnection(XADataSource dataSource,
                                             String userid, String password)
  throws MQException, SQLException, Exception
```

These methods declare Exception in their throws clauses to avoid problems with the JVM verifier for customers who are not using the JTA functionality. The actual exception thrown is javax.transaction.xa.XAException. which requires the jta.jar file to be added to the classpath for programs that did not previously require it.

To use the JTA/JDBC support, you must include the following statement in your application:

```
MQEnvironment.properties.put(MQC.THREAD_AFFINITY_PROPERTY, new Boolean(true));
```

Known problems and limitations with JTA/JDBC coordination

Because this support makes calls to JDBC drivers, the implementation of those JDBC drivers can have significant impact on the system behavior. In particular, tested JDBC drivers behave differently when the database is shut down while an application is running. **Always** avoid abruptly shutting down a database while there are applications holding open connections to it.

Multiple XAResourceManager stanzas

The use of more than one XAResourceManager stanza in a queue manager configuration file, qm.ini, is not supported. Any XAResourceManager stanza other than the first is ignored.

DB2

Sometimes DB2 returns a SQL0805N error. This problem can be resolved with the following CLP command:

```
DB2 bind @db2cli.lst blocking all grant public
```

Refer to the DB2 documentation for more information.

The XAResourceManager stanza must be configured to use ThreadOfControl=PROCESS. For DB2 version 8.1 and higher this does not match the default thread of control setting for DB2, so toc=p must be

specified in the XA Open String. An example XAResourceManager stanza for DB2 with JTA/JDBC coordination is as follows:

```
XAResourceManager:
     Name=jdbcdb2
     SwitchFile=jdbcdb2
     XAOpenString=uid=userid,db=dbalias,pwd=password,toc=p
     ThreadOfControl=PROCESS
```

This does not prevent the Java applications that use JTA/JDBC coordination from being multithreaded themselves.

Oracle 8.1.7
Calling the JDBC Connection.close() method after MQQueueManager.disconnect() generates an SQLException. Either call Connection.close() before MQQueueManager.disconnect(), or omit the call to Connection.close().

Secure Sockets Layer (SSL) support

WebSphere MQ base Java client applications and WebSphere MQ JMS connections using TRANSPORT(CLIENT) support Secure Sockets Layer (SSL) encryption. SSL provides communication encryption, authentication, and message integrity. It is typically used to secure communications between any two peers on the Internet or within an intranet.

WebSphere MQ classes for Java uses Java Secure Socket Extension (JSSE) to handle SSL encryption, and so requires a JSSE provider. J2SE v1.4 JVMs have a JSSE provider built in. Details of how to manage and store certificates can vary from provider to provider. For information about this, refer to your JSSE provider's documentation.

This section assumes that your JSSE provider is correctly installed and configured, and that suitable certificates have been installed and made available to your JSSE provider.

If your WebSphere MQ base Java client application uses a client channel definition table to connect to a queue manager, see "Using a client channel definition table" on page 70.

Enabling SSL

SSL is supported only for client connections. To enable SSL, you must specify the CipherSuite to use when communicating with the queue manager, and this must match the CipherSpec set on the target channel. Additionally, the named CipherSuite must be supported by your JSSE provider. However, CipherSuites are distinct from CipherSpecs and so have different names. Appendix D, "SSL CipherSpecs and CipherSuites," on page 645 contains a table mapping the CipherSpecs supported by WebSphere MQ to their equivalent CipherSuites as known to JSSE.

To enable SSL, specify the CipherSuite using the sslCipherSuite static member variable of MQEnvironment. The following example attaches to a SVRCONN channel named SECURE.SVRCONN.CHANNEL, which has been set up to require SSL with a CipherSpec of RC4_MD5_EXPORT:

```
MQEnvironment.hostname       = "your_hostname";
MQEnvironment.channel        = "SECURE.SVRCONN.CHANNEL";
MQEnvironment.sslCipherSuite = "SSL_RSA_EXPORT_WITH_RC4_40_MD5";
MQQueueManager qmgr = new MQQueueManager("your_Q_manager");
```

Note that, although the channel has a CipherSpec of RC4_MD5_EXPORT, the Java application must specify a CipherSuite of SSL_RSA_EXPORT_WITH_RC4_40_MD5. For more information about CipherSpecs and CipherSuites, see the *WebSphere MQ Security* book. See Appendix D, "SSL CipherSpecs and CipherSuites," on page 645 for a list of mappings between CipherSpecs and CipherSuites.

An application can also specify a CipherSuite by setting the environment property MQC.SSL_CIPHER_SUITE_PROPERTY.

If you require a client connection to use a CipherSuite that is supported by the IBM Java JSSE FIPS provider (IBMJSSEFIPS), an application can set the sslFipsRequired field in the MQEnvironment class to `true`. Alternatively, the application can set the environment property MQC.SSL_FIPS_REQUIRED_PROPERTY. The default value is `false`, which means that a client connection can use any CipherSuite that is not supported by IBMJSSEFIPS.

If an application uses more than one client connection, the value of the sslFipsRequired field that is used when the application creates the first client connection determines the value that is used when the application creates any subsequent client connection. This means that, when the application creates a subsequent client connection, the value of the sslFipsRequired field is ignored. You must restart the application if you want to use a different value for the sslFipsRequired field.

To connect successfully using SSL, the JSSE truststore must be set up with Certificate Authority root certificates from which the certificate presented by the queue manager can be authenticated. Similarly, if SSLClientAuth on the SVRCONN channel has been set to MQSSL_CLIENT_AUTH_REQUIRED, the JSSE keystore must contain an identifying certificate that is trusted by the queue manager.

Using the distinguished name of the queue manager

The queue manager identifies itself using an SSL certificate, which contains a *Distinguished Name* (DN). A WebSphere MQ base Java client application can use this DN to ensure that it is communicating with the correct queue manager. A DN pattern is specified using the sslPeerName variable of MQEnvironment. For example, setting:

```
MQEnvironment.sslPeerName = "CN=QMGR.*, OU=IBM, OU=WEBSPHERE";
```

allows the connection to succeed only if the queue manager presents a certificate with a Common Name beginning QMGR., and at least two Organizational Unit names, the first of which must be IBM and the second WEBSPHERE.

If sslPeerName is set, connections succeed only if it is set to a valid pattern and the queue manager presents a matching certificate.

An application can also specify the distinguished name of the queue manager by setting the environment property MQC.SSL_PEER_NAME_PROPERTY. For more information about distinguished names, see *WebSphere MQ Security*.

Using certificate revocation lists

A certificate revocation list (CRL) is a set of certificates that have been revoked, either by the issuing Certificate Authority or by the local organization. CRLs are typically hosted on LDAP servers. With Java 2 v1.4, a CRL server can be specified at connect-time and the certificate presented by the queue manager is checked

against the CRL before the connection is allowed. For more information about certificate revocation lists and WebSphere MQ, see *WebSphere MQ Security*.

Note: To use a CertStore successfully with a CRL hosted on an LDAP server, make sure that your Java Software Development Kit (SDK) is compatible with the CRL. Some SDKs require that the CRL conforms to RFC 2587, which defines a schema for LDAP v2. Most LDAP v3 servers use RFC 2256 instead.

The CRLs to use are specified through the java.security.cert.CertStore class. Refer to documentation on this class for full details of how to obtain instances of CertStore. To create a CertStore based on an LDAP server, first create an LDAPCertStoreParameters instance, initialized with the server and port settings to use. For example:

```
import java.security.cert.*;
CertStoreParameters csp = new LDAPCertStoreParameters("crl_server", 389);
```

Having created a CertStoreParameters instance, use the static constructor on CertStore to create a CertStore of type LDAP:

```
CertStore cs = CertStore.getInstance("LDAP", csp);
```

Other CertStore types (for example, Collection) are also supported. Commonly there are several CRL servers set up with identical CRL information to give redundancy. Once you have a CertStore object for each of these CRL servers, place them all in a suitable Collection. The following example shows the CertStore objects placed in an ArrayList:

```
import java.util.ArrayList;
Collection crls = new ArrayList();
crls.add(cs);
```

This Collection can be set into the MQEnvironment static variable, sslCertStores, before connecting to enable CRL checking:

```
MQEnvironment.sslCertStores = crls;
```

The certificate presented by the queue manager when a connection is being set up is validated as follows:

1. The first CertStore object in the Collection identified by sslCertStores is used to identify a CRL server.
2. An attempt is made to contact the CRL server.
3. If the attempt is successful, the server is searched for a match for the certificate.
   a. If the certificate is found to be revoked, the search process is over and the connection request fails with reason code MQRC_SSL_CERTIFICATE_REVOKED.
   b. If the certificate is not found, the search process is over and the connection is allowed to proceed.
4. If the attempt to contact the server is unsuccessful, the next CertStore object is used to identify a CRL server and the process repeats from step 2.

   If this was the last CertStore in the Collection, or if the Collection contains no CertStore objects, the search process has failed and the connection request fails with reason code MQRC_SSL_CERT_STORE_ERROR.

The Collection object determines the order in which CertStores are used.

The Collection of CertStores can also be set using the MQC.SSL_CERT_STORE_PROPERTY. As a convenience, this property also allows a single CertStore to be specified without needing to be a member of a Collection.

If sslCertStores is set to null, no CRL checking is performed. This property is ignored if sslCipherSuite is not set.

Renegotiating the secret key used for encryption

A WebSphere MQ base Java client application can control when the secret key that is used for encryption on a client connection is renegotiated. The application can do this in any of the following ways:

- By setting the sslResetCount field in the MQEnvironment class.
- By setting the environment property MQC.SSL_RESET_COUNT_PROPERTY in a Hashtable object. The application then assigns the hashtable to the `properties` field in the MQEnvironment class, or passes the hashtable to an MQQueueManager object on its constructor.

If the application uses more than one of these ways, the usual precedence rules apply. See “MQEnvironment” on page 118 for the precedence rules.

The value of the sslResetCount field or environment property MQC.SSL_RESET_COUNT_PROPERTY represents the total number of bytes sent and received by the WebSphere MQ base Java client code before the secret key is renegotiated. The number of bytes sent is the number before encryption, and the number of bytes received is the number after decryption. The number of bytes also includes control information sent and received by the WebSphere MQ base Java client.

If the reset count is zero, which is the default value, the secret key is never renegotiated. The reset count is ignored if no CipherSuite is specified.

If you are using an HP or Sun Java 2 Software Development Kit (SDK) or Java Runtime Environment (JRE), do not set the reset count to a value other than zero. If you do set the reset count to a value other than zero, a client connection fails when it attempts to renegotiate the secret key.

For more information about the secret key that is used for encryption on an SSL enabled channel, see *WebSphere MQ Security*.

Supplying a customized SSLSocketFactory

Different JSSE implementations can provide different features. For example, a specialized JSSE implementation could allow configuration of a particular model of encryption hardware. Additionally, some JSSE providers allow customization of keystores and truststores by program, or allow the choice of identity certificate from the keystore to be altered. In JSSE, all these customizations are abstracted into a factory class, javax.net.ssl.SSLSocketFactory.

Refer to your JSSE documentation for details of how to create a customized SSLSocketFactory implementation. The details vary from provider to provider, but a typical sequence of steps might be:

1. Create an SSLContext object using a static method on SSLContext
2. Initialize this SSLContext with appropriate KeyManager and TrustManager implementations (created from their own factory classes)
3. Create an SSLSocketFactory from the SSLContext

When you have an SSLSocketFactory object, set the MQEnvironment.sslSocketFactory to the customized factory object. For example:

```
javax.net.ssl.SSLSocketFactory sf = sslContext.getSocketFactory();
MQEnvironment.sslSocketFactory = sf;
```

WebSphere MQ classes for Java then use this SSLSocketFactory to connect to the WebSphere MQ queue manager. This property can also be set using the MQC.SSL_SOCKET_FACTORY_PROPERTY. If sslSocketFactory is set to null, the JVM's default SSLSocketFactory is used. This property is ignored if sslCipherSuite is not set.

Making changes to the JSSE keystore or truststore

If you change the contents of the JSSE keystore or truststore, or change the location of the keystore or truststore file, WebSphere MQ base Java applications that are running at the time do not automatically pick up the changes. For the changes to take effect, the following actions must be performed:

- The applications must close all their connections, and destroy any unused connections in connection pools.
- If your JSSE provider caches information from the keystore and truststore, this information must be refreshed.

After these actions have been performed, the applications can then recreate their connections.

Depending on how you design your applications, and on the function provided by your JSSE provider, it might be possible to perform these actions without stopping and restarting your applications. However, stopping and restarting the applications might be the simplest solution.

Error handling when using SSL

The following reason codes can be issued by WebSphere MQ classes for Java when connecting to a queue manager using SSL:

MQRC_SSL_NOT_ALLOWED
The sslCipherSuite property was set, but bindings connect was used. Only client connect supports SSL.

MQRC_JSSE_ERROR
The JSSE provider reported an error that could not be handled by WebSphere MQ. This could be caused by a configuration problem with JSSE, or because the certificate presented by the queue manager could not be validated. The exception produced by JSSE can be retrieved using the getCause() method on MQException.

MQRC_SSL_PEER_NAME_MISMATCH
The DN pattern specified in the sslPeerName property did not match the DN presented by the queue manager.

MQRC_SSL_PEER_NAME_ERROR
The DN pattern specified in the sslPeerName property was not valid.

MQRC_UNSUPPORTED_CIPHER_SUITE
The CipherSuite named in sslCipherSuite was not recognized by the JSSE provider. A full list of CipherSuites supported by the JSSE provider can be obtained by a program using the SSLSocketFactory.getSupportedCipherSuites() method. A list of CipherSuites that can be used to communicate with WebSphere MQ can be found in Appendix D, "SSL CipherSpecs and CipherSuites," on page 645.

MQRC_SSL_CERTIFICATE_REVOKED
The certificate presented by the queue manager was found in a CRL specified with the sslCertStores property. Update the queue manager to use trusted certificates.

MQRC_SSL_CERT_STORE_ERROR
None of the supplied CertStores could be searched for the certificate presented by the queue manager. The MQException.getCause() method returns the error that occurred while searching the first CertStore attempted. If the causal exception is NoSuchElementException, ClassCastException, or NullPointerException, check that the Collection specified on the sslCertStores property contains at least one valid CertStore object.

Running WebSphere MQ base Java applications

If you write an application (a class that contains a main() method), using either the client or the bindings mode, run your program using the Java interpreter. Use the command:

```
java -Djava.library.path=library_path MyClass
```

where *library_path* is the path to the WebSphere MQ Java libraries (see "The WebSphere MQ Java libraries" on page 10).

Tracing WebSphere MQ base Java programs

WebSphere MQ base Java includes a trace facility, which you can use to produce diagnostic messages if you suspect that there might be a problem with the code. (You normally need to use this facility only at the request of IBM service.)

Tracing is controlled by the enableTracing and disableTracing methods of the MQEnvironment class. For example:

```
MQEnvironment.enableTracing(2);   // trace at level 2
 ...                              // these commands will be traced
MQEnvironment.disableTracing();   // turn tracing off again
```

The trace is written to the Java console (`System.err`).

If your program is an application, or if you run it from your local disk using the appletviewer command, you can also redirect the trace output to a file of your choice. The following code fragment shows an example of how to redirect the trace output to a file called `myapp.trc`:

```
import java.io.*;

try {
  FileOutputStream
  traceFile = new FileOutputStream("myapp.trc");
  MQEnvironment.enableTracing(2,traceFile);
}
catch (IOException ex) {
  // couldn't open the file,
  // trace to System.err instead
  MQEnvironment.enableTracing(2);
}
```

There are five different levels of tracing:

1. Provides entry, exit, and exception tracing

2. Provides parameter information in addition to 1
3. Provides transmitted and received WebSphere MQ headers and data blocks in addition to 2
4. Provides transmitted and received user message data in addition to 3
5. Provides tracing of methods in the Java Virtual Machine in addition to 4

To trace methods in the Java Virtual Machine with trace level 5:

- For an application, run it by issuing the command `java_g` (instead of `java`)
- For an applet, run it by issuing the command `appletviewer_g` (instead of `appletviewer`)

Note: `java_g` is not supported on i5/OS, but similar function is provided by using `OPTION(*VERBOSE)` on the RUNJVA command.

Chapter 8. Environment-dependent behavior

WebSphere MQ classes for Java allow you to create applications that can run against different versions of WebSphere MQ and MQSeries. This chapter describes the behavior of the Java classes dependent on these different versions.

WebSphere MQ classes for Java provides a core of classes, which provide consistent function and behavior in all the environments. Features outside this core depend on the capability of the queue manager to which the application is connected.

Except where noted here, the behavior exhibited is as described in the Application Programming Reference book appropriate to the queue manager.

Core details

WebSphere MQ classes for Java contains the following core set of classes, which can be used in all environments with only the minor variations listed in "Restrictions and variations for core classes" on page 98.

- MQEnvironment
- MQException
- MQGetMessageOptions

  Excluding:
  - MatchOptions
  - GroupStatus
  - SegmentStatus
  - Segmentation
- MQManagedObject

  Excluding:
  - inquire()
  - set()
- MQMessage

  Excluding:
  - groupId
  - messageFlags
  - messageSequenceNumber
  - offset
  - originalLength
- MQPoolServices
- MQPoolServicesEvent
- MQPoolServicesEventListener
- MQPoolToken
- MQPutMessageOptions

  Excluding:
  - knownDestCount
  - unknownDestCount

  - invalidDestCount
  - recordFields
- MQProcess
- MQQueue
- MQQueueManager

  Excluding:
  - begin()
  - accessDistributionList()
- MQSimpleConnectionManager
- MQC

Note:

1. Some constants are not included in the core (see "Restrictions and variations for core classes" for details); do not use them in completely portable programs.
2. Some platforms do not support all connection modes. On these platforms, you can use only the core classes and options that relate to the supported modes. (See Table 1 on page 4.)

Restrictions and variations for core classes

The core classes generally behave consistently across all environments, even if the equivalent MQI calls normally have environment differences. The behavior is as if a Windows or UNIX WebSphere MQ queue manager is used, except for the following minor restrictions and variations.

MQGMO_* values

The following MQGMO_* values are not supported by all queue managers, and their use might throw MQException from an MQQueue.get():

MQGMO_SYNCPOINT_IF_PERSISTENT
MQGMO_MARK_SKIP_BACKOUT
MQGMO_BROWSE_MSG_UNDER_CURSOR
MQGMO_LOCK
MQGMO_UNLOCK
MQGMO_LOGICAL_ORDER
MQGMO_COMPLETE_MESSAGE
MQGMO_ALL_MSGS_AVAILABLE
MQGMO_ALL_SEGMENTS_AVAILABLE
MQGMO_UNMARKED_BROWSE_MSG
MQGMO_MARK_BROWSE_HANDLE
MQGMO_MARK_BROWSE_CO_OP
MQGMO_UNMARK_BROWSE_HANDLE
MQGMO_UNMARK_BROWSE_CO_OP

Additionally, MQGMO_SET_SIGNAL is not supported when used from Java.

MQPMRF_* values

These are used only when putting messages to a distribution list, and are supported only by queue managers supporting distribution lists. For example, z/OS queue managers do not support distribution lists.

MQPMO_* values

The following MQPMO_* values are not supported by all queue managers, and their use might throw MQException from an MQQueue.put() or an MQQueueManager.put():

MQPMO_LOGICAL_ORDER
MQPMO_NEW_CORREL_ID
MQPMO_NEW_MESSAGE_ID
MQPMO_RESOLVE_LOCAL_Q

MQCNO_FASTPATH_BINDING

This value is ignored on queue managers that do not support it, or when using a TCP/IP client connection.

MQRO_* values

The following report options can be set but are ignored by some queue managers. This can affect applications connected to a queue manager that honors the report options when the report message is generated by a remote queue manager that does not. Avoid relying on these options if there is a possibility that a queue manager involved does not support them.

MQRO_EXCEPTION_WITH_FULL_DATA
MQRO_EXPIRATION_WITH_FULL_DATA
MQRO_COA_WITH_FULL_DATA
MQRO_COD_WITH_FULL_DATA
MQRO_DISCARD_MSG
MQRO_PASS_DISCARD_AND_EXPIRY

Miscellaneous differences with z/OS

Message priority
When a message is put with a priority greater than MaxPriority, a z/OS queue manager rejects the put with MQCC_FAILED and MQRC_PRIORITY_ERROR. Other platforms complete the put with MQCC_WARNING and MQRC_PRIORITY_EXCEEDS_MAXIMUM, and treat the message as if it were put with MaxPriority.

BackoutCount
A z/OS queue manager returns a maximum BackoutCount of 255, even if the message has been backed out more than 255 times.

Default dynamic queue prefix
When connected to a z/OS queue manager using a bindings connection, the default dynamic queue prefix is CSQ.*. Otherwise, the default dynamic queue prefix is AMQ.*.

MQQueueManager constructor
Client connect is not supported on z/OS. Attempting to connect with client options results in an MQException with MQCC_FAILED and MQRC_ENVIRONMENT_ERROR. The MQQueueManager constructor might also fail with MQRC_CHAR_CONVERSION_ERROR (if it fails to initialize conversion between the IBM-1047 and ISO8859-1 code pages), or MQRC_UCS2_CONVERSION_ERROR (if it fails to initialize conversion between the queue manager's code page and Unicode). If your application fails with one of these reason codes, ensure that the National Language Resources component of Language Environment® is installed, and ensure that the correct conversion tables are available.

Conversion tables for Unicode are installed as part of the z/OS C/C++ optional feature. See the *z/OS C/C++ Programming Guide*, SC09-4765, for more information about enabling UCS-2 conversions.

Features outside the core

The WebSphere MQ classes for Java contain the following functions that are specifically designed to use API extensions that are not supported by all queue managers. This section describes how they behave when using a queue manager that does *not* support them.

MQQueueManager constructor option

Some of the MQQueueManager constructors include an optional integer argument. This maps onto the MQI's MQCNO options field, and is used to switch between normal and fast path connection. This extended form of the constructor is accepted in all environments, provided that the only options used are MQCNO_STANDARD_BINDING or MQCNO_FASTPATH_BINDING. Any other options cause the constructor to fail with MQRC_OPTIONS_ERROR. The fast path option MQC.MQCNO_FASTPATH_BINDING is honored only when with a bindings connection to a queue manager that supports it. In other environments, it is ignored.

MQQueueManager.begin() method

This can be used only against a WebSphere MQ queue manager on UNIX or Windows systems in bindings mode. Otherwise, it fails with MQRC_ENVIRONMENT_ERROR. See "JTA/JDBC coordination using WebSphere MQ base Java" on page 87 for more details.

MQGetMessageOptions fields

When using a queue manager that does not support the Version 2 MQGMO structure, leave the following fields set to their default values:

GroupStatus
SegmentStatus
Segmentation

Also, the MatchOptions field support only MQMO_MATCH_MSG_ID and MQMO_MATCH_CORREL_ID. If you put unsupported values into these fields, the subsequent MQQueue.get() fail with MQRC_GMO_ERROR. If the queue manager does not support the Version 2 MQGMO structure, these fields are not updated after a successful MQQueue.get().

Distribution lists

The following classes are used to create distribution lists:

MQDistributionList
MQDistributionListItem
MQMessageTracker

You can create and populate MQDistributionLists and MQDistributionListItems in any environment, but not all queue managers allow you to open an MQDistributionList. In particular, z/OS queue managers do not support distribution lists. Attempting to open an MQDistributionList when using such a queue manager results in MQRC_OD_ERROR.

MQPutMessageOptions fields

Four fields in the MQPMO are rendered as the following member variables in the MQPutMessageOptions class:

knownDestCount
unknownDestCount
invalidDestCount
recordFields

These fields are primarily intended for use with distribution lists. However, a queue manager that supports distribution lists also fills in the DestCount fields after an MQPUT to a single queue. For example, if the queue resolves to a local queue, knownDestCount is set to 1 and the other two count fields are set to 0.

If the queue manager does not support distribution lists, these values are simulated as follows:

- If the put() succeeds, unknownDestCount is set to 1, and the others are set to 0.
- If the put() fails, invalidDestCount is set to 1, and the others are set to 0.

The recordFields variable is used with distribution lists. A value can be written into recordFields at any time, regardless of the environment. It is ignored if the MQPutMessageOptions object is used on a subsequent MQQueue.put() or MQQueueManager.put(), rather than MQDistributionList.put().

MQMD fields

The following MQMD fields are largely concerned with message segmentation:

GroupId
MsgSeqNumber
Offset
MsgFlags
OriginalLength

If an application sets any of these MQMD fields to values other than their defaults, and then does a put() or get() on a queue manager that does not support these, the put() or get() raises an MQException with MQRC_MD_ERROR. A successful put() or get() with such a queue manager always leaves the MQMD fields set to their default values. Do not send a grouped or segmented message to a Java application that runs against a queue manager that does not support message grouping and segmentation.

If a Java application attempts to get() a message from a queue manager that does not support these fields, and the physical message to be retrieved is part of a group of segmented messages (that is, it has non-default values for the MQMD fields), it is retrieved without error. However, the MQMD fields in the MQMessage are not updated, the MQMessage format property is set to MQFMT_MD_EXTENSION, and the true message data is prefixed with an MQMDE structure that contains the values for the new fields.

Restrictions under CICS Transaction Server

In the CICS Transaction Server for OS/390 or CICS Transaction Server for z/OS environment, only the main (first) thread is allowed to issue CICS or WebSphere MQ calls. It is therefore not possible to share MQQueueManager or MQQueue objects between threads in this environment, or to create a new MQQueueManager on a child thread.

Restrictions under CICS Transaction Server

"Miscellaneous differences with z/OS" on page 99 identifies some restrictions and variations that apply to the WebSphere MQ classes for Java when running against a z/OS queue manager. Additionally, when running under CICS, the transaction control methods on MQQueueManager are not supported. Instead of issuing MQQueueManager.commit() or MQQueueManager.backout(), applications use the JCICS task synchronization methods, Task.commit() and Task.rollback(). The Task class is supplied by JCICS in the com.ibm.cics.server package.

Part 3. WebSphere MQ base Java API reference

Introduction

This part documents the WebSphere MQ base Java application programming interface. The same information is provided in the file mqjmsapi.jar, which contains the HTML pages generated by the Javadoc tool.

Chapter 9. Package com.ibm.mq

This is the main package of Java classes and interfaces for WebSphere MQ classes for Java. com.ibm.mq.jms is the main package used for WebSphere MQ classes for JMS and it is described separately.

MQChannelDefinition

public class **MQChannelDefinition**
extends Object

```
java.lang.Object
   |
   +----com.ibm.mq.MQChannelDefinition
```

Use the MQChannelDefinition class to pass information concerning the connection with the queue manager to the send, receive, and security exits.

Note: This class does not apply when connecting directly to WebSphere MQ in bindings mode.

Fields

channelName
```
public java.lang.String
```

The name of the channel through which the connection is established.

connectionName
```
public java.lang.String
```

The TCP/IP hostname of the machine on which the queue manager resides.

hdrCompList
```
public java.util.Collection
```

The list of supported header data compression techniques.

localAddress
```
public java.lang.String
```

The actual TCP/IP address in use.

maxMessageLength
```
public int
```

The maximum length of message that can be sent to the queue manager.

msgCompList
```
public java.util.Collection
```

The list of supported message data compression techniques.

queueManagerName
```
public java.lang.String
```

The name of the queue manager to which the connection is made.

receiveUserData
```
public java.lang.String
```

A storage area for the receive exit to use. Information placed here is preserved across invocations of the receive exit, and is also available to the send and security exits.

remotePassword

```
public java.lang.String
```

The password used to establish the connection.

remoteUserId

```
public java.lang.String
```

The user ID used to establish the connection.

securityUserData

```
public java.lang.String
```

A storage area for the security exit to use. Information placed here is preserved across invocations of the security exit and is also available to the send and receive exits.

sendUserData

```
public java.lang.String
```

A storage area for the send exit to use. Information placed here is preserved across invocations of the send exit and is also available to the security and receive exits.

sslPeerName

```
public java.lang.String
```

The SSL peer name used for matching. If SSL is used to encrypt data, the name is set to the Distinguished Name presented by the queue manager during connection. If SSL is not used, it is left at null.

MQChannelExit

public class **MQChannelExit**
extends Object

```
java.lang.Object
   |
   +----com.ibm.mq.MQChannelExit
```

This class defines context information passed to the send, receive and security exits when they are invoked. The exit must set the exitResponse member variable to indicate what action the WebSphere MQ Client for Java must take next.

Note: This class is not used when connecting directly to WebSphere MQ in bindings mode.

Fields

capabilityFlags

```
public int
```

Capabilities of the queue manager.

Only the MQC.MQCF_DIST_LISTS flag is supported.

CurHdrCompression

```
public int
```

The technique currently being used to compress header data.

CurMsgCompression

```
public int
```

The technique currently being used to compress message data.

exitID

```
public int
```

The type of exit that has been invoked. Possible values are:

- MQC.MQXT_CHANNEL_SEC_EXIT
- MQC.MQXT_CHANNEL_SEND_EXIT
- MQC.MQXT_CHANNEL_RCV_EXIT

exitReason

```
public int
```

The reason for invoking the exit. Possible values are:

- MQC.MQXR_INIT
- MQC.MQXR_TERM
- MQC.MQXR_XMIT
- MQC.MQXR_SEC_MSG
- MQC.MQXR_INIT_SEC
- MQC.MQXR_SEC_PARMS

exitResponse

```
public int
```

Set by the exit to indicate the action that the WebSphere MQ Client for Java must take next. Valid values are:

- MQC.MQXCC_OK
- MQC.MQXCC_SUPPRESS_FUNCTION
- MQC.MQXCC_SEND_AND_REQUEST_SEC_MSG
- MQC.MQXCC_SEND_SEC_MSG
- MQC.MQXCC_SUPPRESS_EXIT
- MQC.MQXCC_CLOSE_CHANNEL

exitUserArea

```
public byte[]
```

A storage area available for the exit to use. Any data placed here is preserved by the WebSphere MQ Client for Java across exit invocations with the same exitID. That is, each send, receive or security exit has its own independent user area.

fapLevel

```
public int
```

The negotiated Format and Protocol (FAP) level. The default level is 8.

maxSegmentLength

```
public int
```

The maximum length for a simple transmission to a queue manager. If the exit returns data which is to be sent to the queue manager, the length of the returned data must not exceed this value.

MQXCC_CLOSE_CHANNEL

```
public final static int
```

Deprecated

use MQC.MQXCC_CLOSE_CHANNEL instead.

MQXCC_OK

```
public final static int
```

Deprecated

use MQC.MQXCC_OK instead.

MQXCC_SEND_AND_REQUEST_SEC_MSG

```
public final static int
```

Deprecated

use MQC.MQXCC_SEND_AND_REQUEST_SEC_MSG instead.

MQXCC_SEND_SEC_MSG
```
public final static int
```

Deprecated

use MQC.MQXCC_SEND_SEC_MSG instead.

MQXCC_SUPPRESS_EXIT
```
public final static int
```

Deprecated

use MQC.MQXCC_SUPPRESS_EXIT instead.

MQXCC_SUPPRESS_FUNCTION
```
public final static int
```

Deprecated

use MQC.MQXCC_SUPPRESS_FUNCTION instead.

MQXR_INIT
```
public final static int
```

Deprecated

use MQC.MQXR_INIT instead.

MQXR_INIT_SEC
```
public final static int
```

Deprecated

use MQC.MQXR_INIT_SEC instead.

MQXR_SEC_MSG
```
public final static int
```

Deprecated

use MQC.MQXR_SEC_MSG instead.

MQXR_TERM
```
public final static int
```

Deprecated

use MQC.MQXR_TERM instead.

MQXR_XMIT
```
public final static int
```

Deprecated

use MQC.MQXR_XMIT instead.

MQXT_CHANNEL_RCV_EXIT

```
public final static int
```

Deprecated

use MQC.MQXT_CHANNEL_RCV_EXIT instead.

MQXT_CHANNEL_SEC_EXIT

```
public final static int
```

Deprecated

use MQC.MQXT_CHANNEL_SEC_EXIT instead.

MQXT_CHANNEL_SEND_EXIT

```
public final static int
```

Deprecated

use MQC.MQXT_CHANNEL_SEND_EXIT instead.

Methods

getMQCSP

```
public MQConnectionSecurityParameters getMQCSP();
```

Gets a MQConnectionSecurityParameters object. If no such object has been created, this method will return null.

Returns

- the MQConnectionSecurityParameters object.

setMQCSP

```
public void setMQCSP(MQConnectionSecurityParameters mqcsp);
```

Sets a MQConnectionSecurityParameters object. If this object is created and set when a security exit is invoked with MQC.MQXR_SEC_PARMS then any supplied information will be sent to the Queue Manager.

This applies to channel security exits only.

Parameters

- mqcsp - the MQConnectionSecurityParameters object.

MQConnectionSecurityParameters

public class **MQConnectionSecurityParameters**
extends Object

```
java.lang.Object
   |
   +----com.ibm.mq.MQConnectionSecurityParameters
```

This class is a representation of the MQCSP structure. It is used to enable the Object Authority Manager (OAM) to authenticate a user and change appropriate identity context fields.

In the WebSphere MQ Java client, this field can be set only from within a Security channel exit. When the exit is invoked, the reference to this class in MQChannel Exit will be null. The exit can replace this with an MQConnectionSecurityParameters object defined by the exit. For example:

```
  public byte[] securityExit(MQChannelExit channelExitParms,
           MQChannelDefinition channelDefinition,
           byte[] agentBuffer)
{
      // ... other code ...
      MQConnectionSecurityParameters csp = new MQConnectionSecurityParameters();
     csp.setCSPUserId("myID");
     csp.setCSPPassword("myPassword");
     csp.setAuthenticationType(MQC.MQCSP_AUTH_USER_ID_AND_PWD);
      channelExitParms.setMQCSP(csp);
}
```

If the reference is not null when the exit completes, then the information in the MQConnectionSecurityParameters object created by the Exit will be sent to the queue manager.

Data must be in the character set and encoding of the local queue manager; these are given by the CodedCharSetId queue-manager attribute and MQENC_NATIVE, respectively.

Methods

getAuthenticationType

```
public int getAuthenticationType();
```

This method returns the authentication method to be used by the Object Authority Manager (OAM). It will be either MQCSP_AUTH_NONE or MQCSP_AUTH_USER_ID_AND_PWD.

The initial value of this field is MQCSP_AUTH_NONE

Returns

- int authenticationType

getCSPPassword

```
public String getCSPPassword();
```

This method returns the defined MQCSP password.

Returns

- the MQCSP password

getCSPUserId

```
public String getCSPUserId();
```

This method returns the defined MQCSP user ID.

Returns

- the MQCSP user ID

setAuthenticationType

```
public void setAuthenticationType(int i);
```

Sets the authentication method to be used by the Object Authority Manager (OAM). It can be either MQCSP_AUTH_NONE or MQCSP_AUTH_USER_ID_AND_PWD. Any other value is interpreted as MQCSP_AUTH_NONE.

The initial value of this field is MQCSP_AUTH_NONE .

setCSPPassword

```
public void setCSPPassword(String pass);
```

Sets a String to be used as the MQCSP password. If the authentication type is set to MQCSP_AUTH_USER_ID_AND_PWD then this will be passed to the Object Authority Manager (OAM) for authentication.

The initial value of this field is null.

Parameters

- pass - CSPPassword

setCSPUserId

```
public void setCSPUserId(String id);
```

Sets a String to be used as the MQCSP user ID. If the authentication type is set to MQCSP_AUTH_USER_ID_AND_PWD then this will be passed to the Object Authority Manager (OAM) for authentication.

The initial value of this field is null.

Parameters

- id - the user id

MQDistributionList

public class **MQDistributionList**
extends MQManagedObject

```
java.lang.Object
   |
   +----com.ibm.mq.MQManagedObject
           |
           +----com.ibm.mq.MQDistributionList
```

Create a distribution list using the MQDistributionList() constructor or the MQQueueManager.accessDistributionList() method. A distribution list represents a set of open queues to which messages can be sent using a single call to the put method.

Constructors

MQDistributionList

```
public MQDistributionList(MQQueueManager qMgr,
                          MQDistributionListItem[] litems,
                          int openOptions, String alternateUserId)
             throws MQException;
```

Creates a new distribution list and opens the queues.

Parameters

- qMgr - the queue manager where the list is to be opened.
- litems - the items to be included in the distribution list.
- openOptions - options which control the opening of the distribution list.
- alternateUserId - the alternative user identifier used to check the authorization for opening queues if MQOO_ALTERNATE_USER_AUTHORITY is specified in openOptions. Otherwise this parameter can be left blank (or null).

Exceptions

- MQException - is only thrown if the call fails completely. The constructor completes if at least one queue opens successfully.

Methods

close

```
public void close() throws MQException;
```

Closes the distribution list.

Exceptions

- MQException - if the close fails.

getFirstDistributionListItem

```
public MQDistributionListItem getFirstDistributionListItem();
```

Gets the first item in the distribution list, or null if the list is empty.

Returns

- the first item.

getInvalidDestinationCount

```
public int getInvalidDestinationCount();
```

Gets the number of items in the distribution list that failed to open successfully.

Returns

- the number of items.

getValidDestinationCount

```
public int getValidDestinationCount();
```

Gets the number of items in the distribution list that were opened successfully.

Returns

- the number of items.

put

```
public void put(MQMessage message, MQPutMessageOptions putMessageOptions)
             throws MQException;
```

Puts a message to the queues on the distribution list.

Parameters

- message - the message descriptor information and the returned message data.
- putMessageOptions - controls the action of MQPUT.

Exceptions

- MQException - if the put fails.

MQDistributionListItem

public class **MQDistributionListItem**
extends MQMessageTracker

```
java.lang.Object
   |
   +----com.ibm.mq.MQMessageTracker
           |
           +----com.ibm.mq.MQDistributionListItem
```

Represents a single item (queue) within a distribution list.

Constructors

MQDistributionListItem

```
public MQDistributionListItem();
```

Public constructor.

Fields

completionCode

```
public int
```

The completion code resulting from the most recent operation on this item. If this was the construction of a distribution list, the completion code relates to the opening of the queue. If it was a put operation, the completion code relates to the attempt to put a message onto this queue.

The initial value is 0.

queueManagerName

```
public java.lang.String
```

The name of the queue manager on which the queue is defined. The initial value is "" (empty string).

queueName

```
public java.lang.String
```

The name of a queue to be uses with a distribution list. It cannot be the name of a model queue.

The initial value is "" (empty string).

reasonCode

```
public int
```

The reason code resulting from the last operation on this item. If this was the construction of a distribution list, the reason code relates to the opening of the queue. If it was a put operation, the reason code relates to the attempt to put a message onto this queue.

The initial value is 0.

Methods

getNextDistributedItem

```
public MQDistributionListItem getNextDistributedItem();
```

Gets the next item in the chain.

Returns

- next item, or null if none.

getPreviousDistributedItem

```
public MQDistributionListItem getPreviousDistributedItem();
```

Gets the previous item in chain.

Returns

- previous item, or null if none.

MQEnvironment

public class **MQEnvironment**
extends Object

```
java.lang.Object
   |
   +----com.ibm.mq.MQEnvironment
```

MQEnvironment contains static fields that control the environment in which an MQQueueManager object (and its corresponding connection to WebSphere MQ) is constructed. As values set in MQEnvironment class take effect when the MQQueueManager constructor is called, you must set the values in the MQEnvironment class before you construct a MQQueueManager object.

Note: All the methods and attributes of this class apply to the WebSphere MQ classes for Java client connections, but only enableTracing(), disableTracing(), properties, and version_notice apply to bindings connections.

Fields

CCSID

```
public static int
```

The CCSID used by the client. It does not apply when connecting directly to WebSphere MQ in bindings mode.

Changing this value affects the way that the queue manager you connect to translates information in the WebSphere MQ headers. All data in WebSphere MQ headers is drawn from the invariant part of the ASCII codeset, except for the data in the MQMessage.applicationIdData and MQMessage.putApplicationName fields.

If you avoid using characters from the variant part of the ASCII codeset for these two fields, then the CCSID can be changed from 819 to any other ASCII codeset.

If you change the client CCSID to be the same as that of the queue manager to which you are connecting, you gain a performance benefit at the queue manager because it does not attempt to translate the message headers.

The default value is 819.

channel

```
public static java.lang.String
```

The name of the channel to connect to on the target queue manager. It does not apply when connecting directly to WebSphere MQ in bindings mode. You *must* set this field, or the corresponding property, before constructing an MQQueueManager instance for use in client mode.

connOptions

```
public static int
```

The queue manager connection options. Possible values are:

- MQC.MQCNO_STANDARD_BINDING
- MQC.MQCNO_FASTPATH_BINDING
- MQC.MQCNO_ISOLATED_BINDING

- MQC.MQCNO_SHARED_BINDING
- MQC.MQCNO_RESTRICT_CONN_TAG_Q_MGR
- MQC.MQCNO_RESTRICT_CONN_TAG_QSG
- MQC.MQCNO_SERIALIZE_CONN_TAG_Q_MGR
- MQC.MQCNO_SERIALIZE_CONN_TAG_QSG

connTag

```
public static byte[]
```

The connection tag which allows users to serialize access to the resources they are using on a z/OS queue manager. The connTag `String` is truncated to 128 bytes. connTag is ignored if connOptions is not set.

hdrCompList

```
public static java.util.Collection
```

The list of supported compressors for header compression. Possible values are:

- MQC.MQCOMPRESS_NONE
- MQC.MQCOMPRESS_SYSTEM

hostname

```
public static java.lang.String
```

The TCP/IP hostname of the machine on which the WebSphere MQ server resides. If the hostname is not set, and no overriding properties are set, bindings mode is used to connect to the local queue manager.

localAddressSetting

```
public static java.lang.String
```

The local address, including a range of ports, used when connecting to a WebSphere MQ queue manager through a firewall. The format is `[ip-addr][(low-port[,high-port])]`.

Here are some examples:

9.20.4.98
The channel binds to address 9.20.4.98 locally

9.20.4.98(1000)
The channel binds to address 9.20.4.98 locally and uses port 1000

9.20.4.98(1000,2000)
The channel binds to address 9.20.4.98 locally and uses a port in the range 1000 to 2000

(1000) The channel binds to port 1000 locally

(1000,2000)
The channel binds to a port in the range 1000 to 2000 locally

You can specify a host name instead of an IP address. The variable is initialized from system property com.ibm.mq.localAddress when you start the JVM. The default value is null.

msgCompList

```
public static java.util.Collection
```

The list of supported compressors for message compression. Possible values are:

- MQC.MQCOMPRESS_NONE, the default.
- MQC.MQCOMPRESS_RLE.
- MQC.MQCOMPRESS_ZLIBFAST.
- MQC.MQCOMPRESS_ZLIBHIGH.

password

```
public static java.lang.String
```

The password used to verify the identity of the WebSphere MQ Client. It is equivalent to the MQEnvironment variable MQ_PASSWORD .

If a security exit is not defined for this client, the value of `password` is transmitted to the server and is available to the server security exit when it is invoked.

The default value is "" (empty string).

port

```
public static int
```

The port to be used. This is the port on which WebSphere MQ listens for connection requests.

The default value is 1414.

properties

```
public static java.util.Hashtable
```

A Hashtable that defines the WebSphere MQ environment.

This Hashtable allows you to set environment properties as key/value pairs rather than as individual variables.

The properties can also be passed as a Hashtable in a parameter on the MQQueueManager constructor. Properties passed on the constructor take precedence over values set with this properties variable, but they are otherwise interchangeable. The order of precedence of finding properties is:

1. properties parameter on MQQueueManager constructor
2. MQEnvironment.properties
3. Other MQEnvironment variables
4. Constant default values

The property key names are:

- MQC.CCSID_PROPERTY overrides CCSID.
- MQC.CHANNEL_PROPERTY overrides `channel`.
- MQC.CONNECT_OPTIONS_PROPERTY overrides connOptions.
- MQC.CONNTAG_PROPERTY overrides connTag.
- MQC.HEADER_COMPRESSION_PROPERTY overrides hdrCompList.
- MQC.MESSAGE_COMPRESSION_PROPERTY overrides msgCompList.

- MQC.HOST_NAME_PROPERTY overrides hostname.
- MQC.LOCAL_ADDRESS_PROPERTY overrides localAddressSetting.
- MQC.PASSWORD_PROPERTY overrides `password`.
- MQC.PORT_PROPERTY overrides `port`.
- MQC.RECEIVE_EXIT_PROPERTY overrides receiveExit.
- MQC.SECURITY_EXIT_PROPERTY overrides securityExit.
- MQC.SEND_EXIT_PROPERTY overrides sendExit.
- MQC.SSL_CERT_STORE_PROPERTY overrides sslCertStores.
- MQC.SSL_CIPHER_SUITE_PROPERTY overrides sslCipherSuite.
- MQC.SSL_FIPS_REQUIRED overrides sslFipsRequired.
- MQC.SSL_RESET_COUNT_PROPERTY . overrides sslResetCount.
- MQC.SSL_PEER_NAME_PROPERTY . overrides sslPeerName.
- MQC.SSL_SOCKET_FACTORY_PROPERTY overrides sslSocketFactory.
- MQC.TRANSPORT_PROPERTY forces MQC.TRANSPORT_MQSERIES_BINDINGS or MQC.TRANSPORT_MQSERIES_CLIENT.
- MQC.USER_ID_PROPERTY overrides userID.

receiveExit

```
public static com.ibm.mq.MQReceiveExit
```

The receive exit used when receiving messages from a queue manager. It allows you to examine, and possibly alter, data and is normally used in conjunction with a corresponding send exit at the queue manager.

If you want to provide your own receive exit, define a class that implements the MQReceiveExit interface, and assign receiveExit to an instance of that class.

If you set this field to null no receive exit is called.

securityExit

```
public static com.ibm.mq.MQSecurityExit
```

The security exit used when connecting to a queue manager. It allows you to customize the security flows that occur when an attempt is made to connect to a queue manager.

If you want to provide your own security exit, define a class that implements the MQSecurityExit interface and assign securityExit to an instance of that class.

If you set this field to null no security exit is called.

sendExit

```
public static com.ibm.mq.MQSendExit
```

The send exit used when sending messages to a queue manager. It allows you to examine, and possibly alter, data and is normally used in conjunction with a corresponding receive exit at the queue manager.

If you want to provide your own send exit, define a class that implements the MQSendExit interface, and assign sendExit to an instance of that class.

If you set this field to null no send exit is called.

sslCertStores

```
public static java.util.Collection
```

Collection of SSL CertStores. The collection of CertStores (J2SE v1.4 only) is used to enable WebSphere MQ Java clients to check certificates for revocation in a Certificate Revocation List (CRL).

sslCipherSuite

```
public static java.lang.String
```

The name of the Cipher Suite to be used by SSL. SSL is only valid for a client connection, and is triggered by setting sslCipherSuite . If sslCipherSuite is not set, all of the other values are irrelevant and a standard non-SSL connection is used to connect to the server.

sslFipsRequired

```
public static boolean
```

When this is set to true, the only Cipher Suites which can be used on an SSL connection from this client process are those which are FIPS-enabled. If this has been set to true and a customized sslSocketFactory has been specified the customized sslSocketFactory will not be used, as it cant be guaranteed that the sslSocketFactory is FIPS compliant.

sslPeerName

```
public static java.lang.String
```

The Distinguished Name (DN) of the queue manager to be used by SSL. The peer name is set to indicate that connections will only be allowed where the server is successfully authenticated as a specific DN.

sslResetCount

```
public static int
```

The total number of unencrypted bytes that are sent and received by the initiating channel MCA before the secret key is reset. The number of bytes includes control information sent by the message channel agent. A value of 0 disables secret key reset from occurring.

sslSocketFactory

```
public static java.lang.Object
```

The factory to use when connecting with SSL encryption. If sslCipherSuite is set, this variable can be used to customize all aspects of the SSL connection. For more information on constructing and customizing SSLSocketFactory instances, refer to your JSSE provider; for information regarding the use of this variable. If set to null (default) and SSL encryption is requested, the default SSLSocketFactory is used. This variable is ignored if sslCipherSuite is null.

userID

```
public static java.lang.String
```

The ID used to identify the WebSphere MQ client. It is equivalent to the WebSphere MQ environment variable MQ_USER_ID.

If no security exit is defined for this client, the value of userID is transmitted to the server and is available for use by the server security exit.

The default value is "" (empty string).

version_notice

```
public final static java.lang.String
```

The current version of the WebSphere MQ Java Classes.

Methods

addConnectionPoolToken

```
public static MQPoolToken addConnectionPoolToken();
```

Constructs an MQPoolToken and adds it to the set of tokens. The token is returned to the application to be passed later into removeConnectionPoolToken() method.

Returns

- the token which has been added.

addConnectionPoolToken

```
public static void addConnectionPoolToken(MQPoolToken token);
```

Adds a given MQPoolToken to the connection pool. A default ConnectionManager can use this as a hint; typically, it is enabled only while there is at least one token in the connection pool.

Parameters

- token - the token to be added.

disableTracing

```
public static void disableTracing();
```

This method turns off the WebSphere MQ Client for Java trace facility.

enableTracing

```
public static void enableTracing(int level);
```

Turns on tracing at the specified trace level (traces to system.err). Levels are:

0 entry/exit.

1 as above plus error and exceptions.

2 as above plus some parameter and flow information.

3 as above plus data trace for headers flowing between client and queue manager.

4 as above plus full data trace of all data sent between client and queue manager.

5 as above plus Java VM method tracing.

Parameters

- level - the level of trace.

enableTracing

```
public static void enableTracing(int level, OutputStream stream);
```

This method turns on tracing at the specified trace level. See enableTracing(int) for details of levels.

Specifying a FileOutputStream means that in a WebSphere Application Server version 5 environment, the WebSphere MQ Base Java classes trace is not redirected to the WebSphere Application Server trace adapter which expects a JMS context.

If the trace string `JMSApi=all=enabled` is specified within the WebSphere Application Server version 5 environment at the same time as this is run, it takes priority and this tracing is disabled.

Parameters

- level - the level of trace.
- stream - the stream to which output is sent.

getDefaultConnectionManager

```
public static ConnectionManager getDefaultConnectionManager();
```

Gets the default ConnectionManager.

Returns

- ConnectionManager or null if the default connection manager is an MQConnectionManager rather than a ConnectionManager.

getQueueManagerReference

```
public static MQQueueManager getQueueManagerReference(int scope);
```

Returns an MQQueueManager object reference if one is available within the specified scope. The scope must be one of MQC.ASSOCIATE_ALL or MQC.ASSOCIATE_THREAD, and a queue manager must already have been created with MQC.MQ_QMGR_ASSOCIATION_PROPERTY set to the scope requested.

If no queue manager has been created within the specified scope, or if MQC.ASSOCIATE_NONE is specified, this method will return null.

A call to this method is the same as calling MQEnvironment.getQueueManagerReference(int, Object) with a null Object.

Parameters

- scope - the association scope

Returns

- MQQueueManager, or null if no reference is available.

getQueueManagerReference

```
public static MQQueueManager getQueueManagerReference(int scope,
                                                      Object context);
```

Returns an MQQueueManager object reference if one is available within the specified scope. The scope must be one of MQC.ASSOCIATE_ALL or MQC.ASSOCIATE_THREAD, and a queue manager must already have been created with MQC.MQ_QMGR_ASSOCIATION_PROPERTY set to the scope requested. The supplied Object gives information necessary to identify the MQQueueManager within the scope; for MQC.ASSOCIATE_ALL and MQC.ASSOCIATE_THREAD this Object must be a String containing the name of the queue manager

If no queue manager identified by the supplied Object has been created within the specified scope, or if MQC.ASSOCIATE_NONE is specified, this method will return null.

An MQQueueManager object returned by this method will refer to the same underlying HConn as the MQQueueManager created with MQC.MQ_QMGR_ASSOCIATION_PROPERTY set, and both will therefore share the same transaction context. If an attempt is made to create a second MQQueueManager object on the same context to a different queue manager, then a separate HConn will be made, and the first and second object will have independent transaction contexts. These contexts will extend to WebSphere MQ coordinated JDBC transactions by using MQQueueManager.getJDBCConnection(XADataSource) on the appropriate queue manager.

Parameters

- scope - the association scope
- context - an object containing context. Currently this must be a String specifying a WebSphere MQ queue manager name

Returns

- MQQueueManager, or null if no reference is available.

getVersionNotice

```
public final static String getVersionNotice();
```

Gets the current version of the WebSphere MQ Java Classes.

Returns

- the version.

removeConnectionPoolToken

```
public static void removeConnectionPoolToken(MQPoolToken token);
```

Removes a token from the connection pool.

Parameters

- token - the token to be removed.

setDefaultConnectionManager

```
public static void setDefaultConnectionManager(ConnectionManager cxMan);
```

Sets the default ConnectionManager, and empties the set of MQPoolTokens. The default ConnectionManager is used no ConnectionManager is specified on the MQQueueManager constructor. This method requires a JVM at Java 2 V1.3 or later, with JAAS 1.0 or later installed.

Parameters

- cxMan - the supplied ConnectionManager.

setDefaultConnectionManager

```
public static void setDefaultConnectionManager(MQConnectionManager mqCxMan);
```

Sets the supplied MQConnectionManager to be the default ConnectionManager. The default ConnectionManager is used when no ConnectionManager is specified on the MQQueueManager constructor. This method also empties the set of MQPoolTokens.

Parameters

- mqCxMan - the supplied MQConnectionManager .

traceSystemProperties

```
public static void traceSystemProperties();
```

Outputs the current System Properties to trace. Constructs a StringBuffer containing all the data and traces the entire contents in a single statement. This means that the results are all traced without the usual preceding timestamp and class hashcode. (although these will appear on the line above the properties block).

MQExitChain

public class **MQExitChain**
extends Object

```
java.lang.Object
   |
   +----com.ibm.mq.MQExitChain
```

Class used for chaining send and receive user exits. It is not used directly but is used by MQReceiveExitChain and MQSendExitChain .

Constructors

MQExitChain

```
public MQExitChain();
```

Constructor creates an object with no exits defined.

Methods

getExitChain

```
public List getExitChain();
```

Gets the exits which have been chained by this object.

Returns

- the exits as a List object.

getReasonCode

```
public int getReasonCode();
```

Gets the reason code created in the most recent use of this object.

Returns

- the reason code. See MQException .

MQExternalReceiveExit

public class **MQExternalReceiveExit**
extends MQExternalUserExit
implements MQReceiveExit

```
java.lang.Object
   |
   +----com.ibm.mq.MQExternalUserExit
           |
           +----com.ibm.mq.MQExternalReceiveExit
```

Enables Java code to call a non-Java receive exit. Chaining of exits is implemented by MQReceiveExitChain.

An MQExternalReceiveExit object holds all the information required to construct the MQCXP and MQCD structures which are required when calling the non-Java receive exit.

Constructors

MQExternalReceiveExit

```
public MQExternalReceiveExit();
```

The default constructor.

MQExternalReceiveExit

```
public MQExternalReceiveExit(String libraryName, String entryPointName,
                             String userData);
```

Constructs an object with an exit already defined.

Parameters

- libraryName - the name of the library module which contains the exit.
- entryPointName - the name of the entry point in `libraryName` used by the exit.
- userData - the data defined by the user.

Methods

receiveExit

```
public byte[] receiveExit(MQChannelExit exitParms,
                          MQChannelDefinition channelParms,
                          byte[] data);
```

Calls the external user exit.

Parameters

- exitParms - the data on the exit.
- channelParms - the data on the channel.
- data - the raw message data.

Returns

- the raw message data after processing by the exit.

MQExternalSecurityExit

public class **MQExternalSecurityExit**
extends MQExternalUserExit
implements MQSecurityExit

```
java.lang.Object
   |
   +----com.ibm.mq.MQExternalUserExit
           |
           +----com.ibm.mq.MQExternalSecurityExit
```

Enables Java code to call a non-Java security exit.

An MQExternalSecurityExit object holds all the information required to construct the MQCXP and MQCD objects which are required when calling the non-Java security exit.

Constructors

MQExternalSecurityExit

```
public MQExternalSecurityExit();
```

The default constructor.

MQExternalSecurityExit

```
public MQExternalSecurityExit(String libraryName, String entryPointName,
                              String userData);
```

Constructs an object with an exit already defined.

Parameters

- libraryName - the name of the library module which contains the exit.
- entryPointName - the name of the entry point in `libraryName` used by the exit.
- userData - the data defined by the user.

Methods

securityExit

```
public byte[] securityExit(MQChannelExit exitParms,
                           MQChannelDefinition channelParms,
                           byte[] data);
```

Calls the external user exit.

Parameters

- exitParms - the data on the exit.
- channelParms - the data on the channel.
- data - the raw message data.

Returns

- the raw message data after processing by the exit.

MQExternalSendExit

public class **MQExternalSendExit**
extends MQExternalUserExit
implements MQSendExit

```
java.lang.Object
   |
   +----com.ibm.mq.MQExternalUserExit
           |
           +----com.ibm.mq.MQExternalSendExit
```

Enables Java code to call a non-Java send exit. Chaining of exits is implemented by MQSendExitChain.

An MQExternalSendExit object holds all the information required to construct the MQCXP and MQCD structures which are required when calling the non-Java send exit.

Constructors

MQExternalSendExit

```
public MQExternalSendExit();
```

The default constructor.

MQExternalSendExit

```
public MQExternalSendExit(String libraryName, String entryPointName,
                          String userData);
```

Constructs an object with an exit already defined.

Parameters

- libraryName - the name of the library module which contains the exit.
- entryPointName - the name of the entry point in `libraryName` used by the exit.
- userData - the data defined by the user.

Methods

sendExit

```
public byte[] sendExit(MQChannelExit exitParms,
                       MQChannelDefinition channelParms,
                       byte[] data);
```

Calls the external user exit.

Parameters

- exitParms - the data on the exit.
- channelParms - the data on the channel.
- data - the raw message data.

Returns

- the raw message data after processing by the exit.

MQExternalUserExit

public class **MQExternalUserExit**
extends Object

```
java.lang.Object
   |
   +----com.ibm.mq.MQExternalUserExit
```

The MQExternalUserExit class is a superclass for MQExternalReceiveExit, MQExternalSecurityExit and MQExternalSendExit. You cannot create it directly.

Methods

getReasonCode

```
public int getReasonCode();
```

Gets the latest reason code.

Returns

- the reason code. See MQException .

getUserData

```
public String getUserData();
```

Gets the user data for the exit.

Returns

- the user data.

setEntryPointName

```
public void setEntryPointName(String entryPointName);
```

Sets the name of the entry point for an exit.

Parameters

- entryPointName - the name of the entry point.

setLibraryName

```
public void setLibraryName(String libraryName);
```

Sets the name of the library module which contains the exit.

Parameters

- libraryName - the library name.

setUserData

```
public void setUserData(String userData);
```

Sets the user data for the exit.

Parameters

- userData - the user data.

MQGetMessageOptions

public class **MQGetMessageOptions**
extends Object

```
java.lang.Object
   |
   +----com.ibm.mq.MQGetMessageOptions
```

This class contains options which control the behavior of MQQueue.get().

Constructors

MQGetMessageOptions

```
public MQGetMessageOptions();
```

Constructs an MQGetMessageOptions object with options set to MQC.MQGMO_MO_WAIT, a wait interval of zero, and a blank resolved queue name.

MQGetMessageOptions

```
public MQGetMessageOptions(boolean noReadBack);
```

Constructs an MQGetMessageOptions object with an option on reading the `options` field. You can use this constructor to save some overheads if your application never needs to read back the options field.

Parameters

- noReadBack - if **true**, prevents the options MQGMO field from being read back. This means that the overhead of converting it is avoided.

Fields

groupStatus

```
public char
```

Whether the retrieved message is in a group, and if it is, whether it is the last in the group. Possible values are:

- MQC.MQGS_LAST_MSG_IN_GROUP
- MQC.MQGS_MSG_IN_GROUP
- MQC.MQGS_NOT_IN_GROUP

matchOptions

```
public int
```

Selection criteria which determine which message is retrieved. The following match options can be set:

- MQC.MQMO_MATCH_CORREL_ID
- MQC.MQMO_MATCH_GROUP_ID
- MQC.MQMO_MATCH_MSG_ID
- MQC.MQMO_MATCH_MSG_SEQ_NUMBER
- MQC.MQMO_NONE

The default value is MQC.MQMO_MATCH_MSG_ID |
MQC.MQMO_MATCH_CORREL_ID .

msgToken

```
public byte[]
```

A token for use when getting messages. It is set either by the queue manager or by the application in combination with MQMO_MATCH_MSG_TOKEN . The token is truncated if its size is greater than MQC.MQ_MSG_TOKEN_LENGTH. It is ignored if it has been set without the corresponding matchOption being set. If matchOption is set for a platform other than z/OS an attempted get will fail.

options

```
public int
```

Options which control the action of MQQueue.get() . Any or none of the following values can be specified. If more than one option is required the values can combined using either '+' or '|'.

- MQC.MQGMO_WAIT
- MQC.MQGMO_NO_WAIT
- MQC.MQGMO_SYNCPOINT
- MQC.MQGMO_NO_SYNCPOINT - default
- MQC.MQGMO_BROWSE_FIRST
- MQC.MQGMO_BROWSE_NEXT
- MQC.MQGMO_BROWSE_MSG_UNDER_CURSOR
- MQC.MQGMO_MSG_UNDER_CURSOR
- MQC.MQGMO_LOCK
- MQC.MQGMO_UNLOCK
- MQC.MQGMO_ACCEPT_TRUNCATED_MSG
- MQC.MQGMO_FAIL_IF_QUIESCING
- MQC.MQGMO_CONVERT

resolvedQueueName

```
public java.lang.String
```

The local name of the queue from which the message was retrieved. This is the resolved name as set by the queue manager. It is different from the name used to open the queue, if an alias queue or model queue was opened.

returnedLength

```
public int
```

The length in bytes of the message data. It is set by the queue manager to the value returned by the MQGET call. If the queue manager does not support this capability, the value is set to MQC.MQRL_UNDEFINED .

segmentation

```
public char
```

Whether segmentation is allowed for the retrieved message. Possible values are:

- MQC.MQSEG_INHIBITED
- MQC.MQSEG_ALLOWED

segmentStatus

```
public char
```

Whether the retrieved message is a segment of a logical message. If the message is a segment, indicates whether or not it is the last segment. Possible values are:

- MQC.MQSS_LAST_SEGMENT
- MQC.MQSS_NOT_A_SEGMENT
- MQC.MQSS_SEGMENT

waitInterval

```
public int
```

The maximum time (in milliseconds) that an MQQueue.get() call waits for a suitable message to arrive. It is used in conjunction with MQC.MQGMO_WAIT. A value of MQC.MQWI_UNLIMITED indicates that an unlimited wait is required.

Methods

MQJavaLevel

public class **MQJavaLevel**
extends Object

```
java.lang.Object
   |
   +----com.ibm.mq.MQJavaLevel
```

Displays information about the currently installed version of WebSphere MQ Classes for Java.

Add this class to the CLASSPATH, and run it using the command java com.ibm.mq.MQJavaLevel. You can modify the output with the following parameters:

`-b` - basic format (no titles)

`-f n` - fields to display

where *n* is one, or a combination of, the following digits:

1 Name

2 Version

4 Build ID

8 Build Type

You can add these numbers together (for example, '3' displays both the Name and Version fields). If `-f` is not specified, the default is to display all fields.

MQManagedObject

public class **MQManagedObject**
extends Object

```
java.lang.Object
   |
   +----com.ibm.mq.MQManagedObject
```

MQManagedObject is a superclass for MQDistributionList, MQProcess, MQQueue, MQQueueManager . It provides the ability to inquire and set attributes of these objects.

Fields

alternateUserId

```
public java.lang.String
```

The alternative user ID specified (if any) when this resource was opened. Setting this attribute has no effect.

closeOptions

```
public int
```

Controls the way the resource is closed. Permitted values are:

- MQC.MQCO_NONE - the default value
- MQC.MQCO_DELETE - permanent dynamic queues only
- MQC.MQCO_DELETE_PURGE - permanent dynamic queues only

connectionReference

```
public com.ibm.mq.MQQueueManager
```

The queue manager to which this resource belongs. Setting this attribute has no effect.

isOpen

```
public boolean
```

Deprecated

use the isOpen() method instead.

Indicates whether this resource is currently open. Do not set this attribute.

name

```
public java.lang.String
```

The name of this resource. This is either the name supplied by the access method, or the name allocated by the queue manager for a dynamic queue. Setting this attribute has no effect.

openOptions

```
public int
```

The options specified when this resource was opened. Setting this attribute has no effect.

Methods

close

```
public void close() throws MQException;
```

Closes the object. No further operations on this object are permitted after it is closed. The behavior of the close method can be altered by setting closeOptions.

Exceptions

- MQException - if the WebSphere MQ call fails.

getAttributeString

```
public final String getAttributeString(int aSelector, int length)
             throws MQException;
```

Gets an attribute string.

Parameters

- aSelector - indicates which attribute is being queried. Suitable selectors for character attributes are shown in MQC.MQCA_*.
- length - the length of string required.

Exceptions

- MQException - if the call fails.

getDescription

```
public String getDescription() throws MQException;
```

Gets the description of this resource as held at the queue manager.

Returns

- the description.

Exceptions

- MQException - if this method is called after the resource has been closed, to indicate that the resource is no longer available.

inquire

```
public void inquire(int[] selectors, int[] intAttrs, byte[] charAttrs)
             throws MQException;
```

Queries requested attributes of the object.

Many of the common attribute values can be queried using the getXXX() methods defined in MQManagedObject, MQQueue, MQQueueManager and MQProcess.

Parameters

- selectors - indicates which attributes are being queried. Suitable selectors for character attributes are shown in MQC.MQCA_*. Suitable selectors for integer attributes are shown in MQC.MQIA_*.
- intAttrs - the requested attribute values in the same order as in `selectors`.
- charAttrs - the requested character attributes, concatenated together and in the same order as in `selectors`.

Exceptions

- MQException - if the inquire fails.

isOpen

```
public boolean isOpen();
```

Indicates whether this object is open.

Returns

- **true** if the object is open.

set

```
public void set(int[] selectors, int[] intAttrs, byte[] charAttrs)
             throws MQException;
```

Sets requested attributes of the object.

Note that many of the more common attribute values can be set using the setXXX() methods defined in MQQueue .

Parameters

- selectors - indicates which attributes are being set. Suitable selectors for character attributes are shown in MQC.MQCA_*. Suitable selectors for integer attributes are shown in MQC.MQIA_*.
- intAttrs - the requested attribute values in the same order as in *selectors*.
- charAttrs - the requested character attributes, concatenated together and in the same order as in *selectors*.

Exceptions

- MQException - if the inquire fails.

setAttributeString

```
public final void setAttributeString(int aSelector, String aValue,
                                     int length) throws MQException;
```

Sets an attribute string.

Parameters

- aSelector - integer which indicates which attribute is being set. Suitable selectors for character attributes are shown in MQC.MQCA_*. Please refer to *WebSphere MQ Application Programming Reference* for further details.
- aValue - the value of the attribute.
- length - the number of characters of aValue to set.

Exceptions

- MQException - if the call fails.

MQMD

public class **MQMD**
extends Object

```
java.lang.Object
   |
   +----com.ibm.mq.MQMD
```

The MQMD class contains the control information that accompanies the application data when a message travels between the sending and receiving applications.

Character data in the message descriptor is in the character set of the queue manager to which the application is connected; this is given by the CodedCharSetId queue manager attribute. Numeric data in the message descriptor is in the native machine encoding (given by MQENC_NATIVE).

If the sending and receiving queue managers use different character sets or encodings, the data in the message descriptor is converted automatically - it is not necessary for the receiving application to perform these conversions.

You can write an exit to convert an application's message data, which is invoked when an MQGET call retrieves the message.

Fields

accountingToken

```
public byte[]
```

The accounting token. This is part of the identity of the message and it allows work done as a result of the message to be appropriately charged.

The default value is an array of zeros.

applicationIdData

```
public java.lang.String
```

Application ID data. It is part of the identity context of the message - information defined by the application suite; it can be used to provide additional information about the message or its originator.

The default value is "" (empty string).

applicationOriginData

```
public java.lang.String
```

Data about the originating application. This can be used by the application to provide additional information about the origin of the message.

The default value is "" (empty string).

backoutCount

```
public int
```

The number of times the message has been backed out. This is the number of times the message has been returned by MQQueue.get(), as part of a unit of work, and subsequently backed out.

The default value is zero.

characterSet

```
public int
```

The coded character set identifier of character data in the application message data. It alters the behavior of MQMessage.readString(), MQMessage.readLine() and MQMessage.writeString() .

The default value for this field is MQC.MQCCSI_Q_MGR . The following additional character set values are supported:

850 commonly used ASCII codeset

819 the ISO standard ASCII codeset

37 the American EBCDIC codeset

1200 Unicode

1208 UTF-8

correlationId

```
public byte[]
```

Specifies the correlation identifier of the message to be retrieved. This applies to MQQueue.get(). Normally the queue manager returns the first message whose message identifier and correlation identifier match those specified. The special value MQC.MQCI_NONE allows *any* correlation identifier to match.

For MQQueue.get() this specifies the correlation identifier to use.

The default value is MQC.MQCI_NONE.

encoding

```
public int
```

Specifies the representation used for numeric values in the application message data. This applies to binary, packed decimal and floating point data. The behavior of the read and write methods for these numeric formats is altered accordingly.

The following encodings are defined:

	binary	**packed decimal**	**floating point**
big-endian	MQC.MQENC_INTEGER_NORMAL	MQC.MQENC_DECIMAL_NORMAL	MQC.MQENC_FLOAT_IEEE_NORMAL
little-endian	MQC.MQENC_INTEGER_REVERSED	MQC.MQENC_DECIMAL_REVERSED	MQC.MQENC_FLOAT_IEEE_REVERSED
zSeries	MQC.MQENC_INTEGER_NORMAL	MQENC_DECIMAL_NORMAL	MQC.MQENC_FLOAT_S390

You can construct a value for the encoding field by combining one value from each row of the table by use of '+' or '|' operators. The default value is MQC.MQENC_INTEGER_NORMAL | MQC.MQENC_DECIMAL_NORMAL | MQC.MQENC_FLOAT_IEEE_NORMAL. For convenience this value is also represented by MQC.MQENC_NATIVE. This setting causes MQMessage.writeInt() to write, for example, a big-endian integer, and MQMessage.readInt() to read a big-endian integer.

A loss in precision can occur when converting from IEEE format floating point values to zSeries format floating point values.

expiry

```
public int
```

The expiry time (in tenths of a second). It is set by the application which puts the message. After a message's expiry time has elapsed, it is eligible to be discarded by the queue manager. If the message specified one of the MQC.MQRO_EXPIRATION flags, then a report is generated when the message is discarded.

The default value is MQC.MQEI_UNLIMITED, which means that the message never expires.

feedback

```
public int
```

The nature of the feedback report. It is used with a message of type MQC.MQMT_REPORT to indicate the nature of the report. The following feedback codes are defined:

- MQC.MQFB_EXPIRATION
- MQC.MQFB_COA
- MQC.MQFB_COD
- MQC.MQFB_QUIT
- MQFB_NONE
- MQC.MQFB_SYSTEM_FIRST
- MQC.MQFB_APPL_CANNOT_BE_STARTED
- MQC.MQFB_TM_ERROR
- MQC.MQFB_APPL_TYPE_ERROR
- MQC.MQFB_STOPPED_BY_MSG_EXIT
- MQC.MQFB_XMIT_Q_MSG_ERROR
- MQC.MQFB_SYSTEM_LAST
- MQC.MQFB_ACTIVITY
- MQC.MQFB_MAX_ACTIVITIES
- MQC.MQFB_NOT_FORWARDED
- MQC.MQFB_NOT_DELIVERED
- MQC.MQFB_UNSUPPORTED_FORWARDING
- MQC.MQFB_UNSUPPORTED_DELIVERY

Application-defined feedback values in the range MQC.MQFB_APPL_FIRST to MQC.MQFB_APPL_LAST can also be used.

The default value of this field is MQC.MQFB_NONE , indicating that no feedback is provided.

format

```
public java.lang.String
```

A name which indicates the nature of the data in the message. It is set by the sender. You can use your own format names, but names beginning with the letters "MQ" have meanings that are defined by the queue manager. The queue manager built-in formats are:

- MQC.MQFMT_NONE
- MQC.MQFMT_ADMIN
- MQC.MQFMT_COMMAND_1
- MQC.MQFMT_COMMAND_2
- MQC.MQFMT_DEAD_LETTER_HEADER
- MQC.MQFMT_EVENT
- MQC.MQFMT_MD_EXTENSION
- MQC.MQFMT_PCF
- MQC.MQFMT_STRING
- MQC.MQFMT_TRIGGER
- MQC.MQFMT_XMIT_Q_HEADER

The default value is MQC.MQFMT_NONE.

groupId

```
public byte[]
```

The ID of the message group. This identifies the message group to which the message belongs.

messageFlags

```
public int
```

Flags controlling the segmentation and status of the message. Possible values are:

- MQC.MQMF_NONE, the default
- MQC.MQMF_SEGMENTATION_INHIBITED
- MQC.MQMF_SEGMENTATION_ALLOWED
- MQC.MQMF_SEGMENT
- MQC.MQMF_LAST_SEGMENT
- MQC.MQMF_MSG_IN_GROUP
- MQC.MQMF_LAST_MSG_IN_GROUP

messageId

```
public byte[]
```

Specifies the message identifier of the message to be retrieved. This applies to MQQueue.get(). Normally the queue manager returns the first message whose message identifier and correlation identifier match those specified. The special value MQC.MQMI_NONE allows *any* message identifier to match.

For MQQueue.put() this specifies the message identifier to use. If MQC.MQMI_NONE is specified, the queue manager generates a unique message identifier when the message is put. The value of this field is updated after the put to indicate the message identifier that was used.

The default value is MQC.MQMI_NONE.

messageSequenceNumber

```
public int
```

Sequence number of logical message within group.

messageType
```
public int
```

Indicates the type of the message. The following values are currently defined:
- MQC.MQMT_DATAGRAM
- MQC.MQMT_REQUEST
- MQC.MQMT_REPLY
- MQC.MQMT_REPORT

Application defined values can also be used; these can be in the range MQMT_APPL_FIRST to MQMT_APPL_LAST .

The default value of this field is MQC.MQMT_DATAGRAM .

offset
```
public int
```

Offset of data in the physical message from the start of the logical message.

originalLength
```
public int
```

Original length of a segmented message.

persistence
```
public int
```

The message persistence. The following values are defined:
- MQC.MQPER_PERSISTENT
- MQC.MQPER_NOT_PERSISTENT
- MQC.MQPER_PERSISTENCE_AS_Q_DEF

The default value is MQC.MQPER_PERSISTENCE_AS_Q_DEF

priority
```
public int
```

The message priority. The default value is MQC.MQPRI_PRIORITY_AS_Q_DEF .

putApplicationName
```
public java.lang.String
```

The name of the application that put the message. The default value is "" (empty string).

putApplicationType
```
public int
```

The type of application that put the message. The value can be defined by the system or by the user. The following values are defined by the system:
- MQC.MQAT_AIX
- MQC.MQAT_CICS
- MQC.MQAT_DOS
- MQC.MQAT_IMS

- MQC.MQAT_MVS
- MQC.MQAT_OS2
- MQC.MQAT_OS400
- MQC.MQAT_QMGR
- MQC.MQAT_UNIX
- MQC.MQAT_WINDOWS
- MQC.MQAT_JAVA
- MQC.MQAT_UNKNOWN
- MQC.MQAT_NO_CONTEXT
- MQC.MQAT_CICS_VSE
- MQC.MQAT_VMS
- MQC.MQAT_GUARDIAN
- MQC.MQAT_VOS
- MQC.MQAT_DEFAULT
- MQC.MQAT_NSK
- MQC.MQAT_CICS_BRIDGE
- MQC.MQAT_NOTES_AGENT
- MQC.MQAT_WINDOWS_NT
- MQC.MQAT_IMS_BRIDGE
- MQC.MQAT_XCF

The default value is the special value MQC.MQAT_NO_CONTEXT, which indicates that no context information is present in the message.

putDateTime

```
public java.util.GregorianCalendar
```

The time and date when the message was put.

replyToQueueManagerName

```
public java.lang.String
```

The name of the queue manager to which reply or report messages can be sent.

The default value is "" (empty string).

replyToQueueName

```
public java.lang.String
```

The name of the queue to which a reply can be sent. The application that issued the get request for the message can send MQC.MQFMT_REPLY and MQC.MQFMT_REPORT messages to this queue.

The default value is "" (empty string).

report

```
public int
```

A report message about another message. This field enables the application sending the original message to specify which report messages are required, whether the application message data is to be included in them, and also how the message and correlation ID in the report or reply are to be set. It comprises one or

more constants from the MQC class combined by means of the '+' or '|' operators. You can select one type from each row of the following table:

	basic	with data	with full data
Exception	MQC.MQRO_EXCEPTION	MQC.MQRO_EXCEPTION_WITH_DATA	MQC.MQRO_EXCEPTION_WITH_FULL_DATA
Expiration	MQC.MQRO_EXPIRATION	MQC.MQRO_EXPIRATION_WITH_DATA	MQC.MQRO_EXPIRATION_WITH_FULL_DATA
Confirm on arrival	MQC.MQRO_COA	MQC.MQRO_COA_WITH_DATA	MQC.MQRO_COA_WITH_FULL_DATA
Confirm on delivery	MQC.MQRO_COD	MQC.MQRO_COD_WITH_DATA	MQC.MQRO_COD_WITH_FULL_DATA

You can specify how the message ID is generated for the report or reply message:

- MQC.MQRO_NEW_MSG_ID
- MQC.MQRO_PASS_MSG_ID

You can specify one of the following to control how to set the correlation ID of the report or reply message:

- MQC.MQRO_COPY_MSG_ID_TO_CORREL_ID
- MQC.MQRO_PASS_CORREL_ID

You can specify the following to pass the discard option and expiry time of the original message to the report or reply message:

- MQC.MQRO_PASS_DISCARD_AND_EXPIRY

You can specify one of the following to control the disposition of the original message when it cannot be delivered to the destination queue:

- MQC.MQRO_DEAD_LETTER_Q
- MQC.MQRO_DISCARD_MSG
- MQC.MQRO_PASS_DISCARD_AND_EXPIRY

If no report options are specified, the default is MQC.MQRO_NEW_MSG_ID | MQC.MQRO_COPY_MSG_ID_TO_CORREL_ID | MQC.MQRO_DEAD_LETTER_Q .

userId

```
public java.lang.String
```

The user ID. It is part of the identity of the message and identifies which user originated it.

The default value is "" (empty string).

Methods

getVersion

```
public int getVersion();
```

Gets the version of the message.

Returns

- the version.

setVersion

```
public void setVersion(int version) throws MQException;
```

Sets the version of the message.

Parameters

- version - the value to be set.

Exceptions

- MQException - if the value of the version is inconsistent.

MQMessage

public class **MQMessage**
extends MQMD
implements DataInputDataOutput

```
java.lang.Object
   |
   +----com.ibm.mq.MQMD
           |
           +----com.ibm.mq.MQMessage
```

MQMessage represents both the message descriptor and the data for a WebSphere MQ message. It has a group of methods for reading data from a message and a group of methods for writing data into a message. The format of numbers and strings used by these read and write methods is controlled by the `encoding` and `characterSet` fields. The remaining fields contain control information which accompanies the application message data when a message travels between sending and receiving applications.

Constructors

MQMessage

```
public MQMessage();
```

The default constructor. This creates a message with default message descriptor information and an empty message buffer.

Methods

clearMessage

```
public void clearMessage() throws IOException;
```

Discards any data in the message buffer and sets the cursor position to zero.

Exceptions

- IOException - if there is a problem with IO.

getDataLength

```
public int getDataLength() throws IOException;
```

Gets the number of bytes of message data remaining to be read.

getDataOffset

```
public int getDataOffset() throws IOException;
```

Returns the current cursor position within the message data (the point at which read and write operations take effect).

Returns

- the cursor position.

Exceptions

- IOException - if there is a problem with IO.

getMessageLength

```
public int getMessageLength() throws IOException;
```

Gets the number of bytes of message data in this message.

Returns

- the number of bytes.

Exceptions

- IOException - if there is a problem with IO.

getTotalMessageLength

```
public int getTotalMessageLength();
```

Gets the total number of bytes in the message as stored on the message queue on which this message was held. This method reports the total size of the message on the queue when an MQQueue.get() method fails with an error code which indicates that the message has been truncated.

readBoolean

```
public boolean readBoolean() throws IOException, EOFException;
```

Reads a boolean from the current position in the message buffer.

Returns

- the boolean.

Exceptions

- IOException - if there is a problem with IO.
- EOFException - if the read goes beyond the end of file.

readByte

```
public byte readByte() throws IOException, EOFException;
```

Reads a byte from the current position in the message buffer.

Returns

- the (signed) byte.

Exceptions

- IOException - if there is a problem with IO.
- EOFException - if the read goes beyond the end of file.

readChar

```
public char readChar() throws IOException, EOFException;
```

Reads a character from the current position in the message buffer.

Returns

- the Unicode character.

Exceptions

- IOException - if there is a problem with IO.
- EOFException - if the read goes beyond the end of file.

readDecimal2

```
public short readDecimal2() throws IOException;
```

Reads a 2-byte packed decimal number in the range -999 to 999.

Returns

- a big-endian short if `encoding` equals MQC.MQENC_DECIMAL_NORMAL or a little-endian short if it equals MQC.MQENC_DECIMAL_REVERSED.

Exceptions

- IOException - if there is a problem with IO.

readDecimal4

```
public int readDecimal4() throws IOException;
```

Reads a 4-byte packed decimal number in the range -9,999,999 to 9,999,999.

Returns

- a big-endian int if `encoding` equals MQC.MQENC_DECIMAL_NORMAL or a little-endian int if it equals MQC.MQENC_DECIMAL_REVERSED.

Exceptions

- IOException - if there is a problem with IO.

readDecimal8

```
public long readDecimal8() throws IOException;
```

Reads an 8-byte packed decimal number in the range -999,999,999,999,999 to 999,999,999,999,999.

Returns

- a big-endian long if `encoding` equals MQC.MQENC_DECIMAL_NORMAL or a little-endian long if it equals MQC.MQENC_DECIMAL_REVERSED.

Exceptions

- IOException - if there is a problem with IO.

readDouble

```
public double readDouble() throws IOException, EOFException;
```

Reads a double from the current position in the message buffer.

Returns

- a big-endian double if `encoding` is equal to MQC.MQENC_INTEGER_NORMAL, or a little-endian double if it is equal to MQC.MQENC_INTEGER_REVERSED .

Exceptions

- IOException - if `encoding` is not equal to either of these values.
- EOFException - if the read goes beyond the end of file.

readFloat

```
public float readFloat() throws IOException, EOFException;
```

Reads a double from the current position in the message buffer.

Returns

- a big-endian float if `encoding` equals MQC.MQENC_INTEGER_NORMAL, a little-endian float if it equals MQC.MQENC_INTEGER_REVERSED, or a zSeries format floating point number if it equals MQC.MQENC_FLOAT_S390.

Exceptions

- IOException - if `encoding` is none of these.
- EOFException - if the read goes beyond the end of file.

readFully

```
public void readFully(byte[] b) throws IOException;
```

Fills a byte array with data from the message buffer.

Parameters

- b - the byte array.

Exceptions

- IOException - if there is a problem with IO.

readFully

```
public void readFully(byte[] b, int off, int len) throws IOException;
```

Partly fills a byte array with data from the message buffer.

Parameters

- b - the byte array.
- off - the offset into the message buffer where the reading starts.
- len - the number of bytes to be read.

Exceptions

- IOException - if there is a problem with IO.

readInt

```
public int readInt() throws IOException;
```

Reads an integer from the current position in the message buffer.

Returns

- a big-endian integer if `encoding` is equal to MQC.MQENC_INTEGER_NORMAL, or a little-endian integer if it is equal to MQC.MQENC_INTEGER_REVERSED .

Exceptions

- IOException - if there is a problem with IO.

readInt2

```
public short readInt2() throws IOException;
```

Identical to readShort(). Provided for cross-language WebSphere MQ API compatibility.

Returns

- a big-endian short if `encoding` is equal to MQC.MQENC_INTEGER_NORMAL, or a little-endian short if it is equal to MQC.MQENC_INTEGER_REVERSED .

Exceptions

- IOException - if there is a problem with IO.

readInt4

```
public int readInt4() throws IOException;
```

Synonym for readInt(), provided for cross-language WebSphere MQ API compatibility.

Exceptions

- IOException - if there is a problem with IO.

readInt8

```
public long readInt8() throws IOException;
```

Identical to readLong(). Provided for cross-language WebSphere MQ API compatibility.

Returns

- a big-endian long if `encoding` is equal to MQC.MQENC_INTEGER_NORMAL, or a little-endian long if it is equal to MQC.MQENC_INTEGER_REVERSED .

Exceptions

- IOException - if encoding is not equal to either of these values.

readLine

```
public String readLine() throws IOException;
```

Reads a line of text from the message. Converts from the codeset identified in characterSet, and then reads in a line that has been terminated by \n, \r, \r\n, EOF or the end of a UTF string.

Returns

- the returned string, in Unicode.

Exceptions

- IOException - if there is a problem with IO.

readLong

```
public long readLong() throws IOException;
```

Reads an integer from the current position in the message buffer.

Returns

- a big-endian long if `encoding` is equal to MQC.MQENC_INTEGER_NORMAL, or a little-endian long if it is equal to MQC.MQENC_INTEGER_REVERSED .

Exceptions

- IOException - if encoding is not equal to either of these values.

readMQMDE

```
public void readMQMDE() throws MQException, IOException;
```

Reads an imbedded extended MQMD object. It uses extended MQMD information to update encoding, characterSet, format, groupId, messageSequenceNumber, offset, messageFlags and originalLength fields. Only call this method if format is MQC.MQFMT_MD_EXTENSION .

Exceptions

- IOException - if there is a problem with IO.
- EOFException - if the read goes beyond the end of file.

readObject

```
public Object readObject() throws ClassNotFoundException,
           InvalidClassException, StreamCorruptedException,
           OptionalDataException, IOException;
```

Reads an object carried in the message.

Returns

- the Object.

Exceptions

- ClassNotFoundException -
- InvalidClassException -
- StreamCorruptedException -
- OptionalDataException -
- IOException -

readShort

```
public short readShort() throws IOException;
```

Reads a short from the current position in the message buffer.

Returns

- a big-endian short if encoding is equal to MQC.MQENC_INTEGER_NORMAL, or a little-endian short if it is equal to MQC.MQENC_INTEGER_REVERSED .

Exceptions

- IOException - if encoding is not equal to either of these values.

readString

```
public String readString(int length) throws IOException;
```

Reads a string in the codeset identified by characterSet and converts it into Unicode.

Parameters

- length - The number of characters to read (which might differ from the number of bytes according to the codeset, since some codesets use more than one byte per character).

readStringOfByteLength

```
public String readStringOfByteLength(int numberOfBytes)
             throws IOException, EOFException;
```

Reads a specified number of bytes and uses them to construct a new string using the character set specified by characterSet. When the given bytes are not valid in the given charset, the behavior of this method is dependant on the implementation of the JRE.

Where the byte length of a string is known, read the entire String in a single invocation of this method. This will avoid problems where byte and char boundaries do not coincide.

Parameters

- numberOfBytes - The number of bytes to read.

Returns

- the string.

Exceptions

- IOException - if there is a problem with IO.

readStringOfCharLength

```
public String readStringOfCharLength(int numberOfChars)
              throws IOException, EOFException;
```

Reads a string in the codeset identified by characterSet and converts it into Unicode.

Parameters

- numberOfChars - The number of characters to read (which might differ from the number of bytes according to the codeset, because some codesets use more than one byte per character).

Returns

- the string.

Exceptions

- IOException - if there is a problem with IO.

readUInt2

```
public int readUInt2() throws IOException;
```

Identical to readUnsignedShort(), provided for cross-language WebSphere MQ API compatibility.

Returns

- an int which contains a big-endian short if `encoding` is equal to MQC.MQENC_INTEGER_NORMAL, or a little-endian short if it is equal to MQC.MQENC_INTEGER_REVERSED.

Exceptions

- IOException - if there is a problem with IO.

readUnsignedByte

```
public int readUnsignedByte() throws IOException;
```

Reads an unsigned byte from the current position in the message buffer.

Returns

- an int which contains the value.

Exceptions

- IOException - if there is a problem with IO.

readUnsignedShort

```
public int readUnsignedShort() throws IOException;
```

Reads an unsigned short from the current position in the message buffer.

Returns

- an int which contains a big-endian short if `encoding` is equal to MQC.MQENC_INTEGER_NORMAL, or a little-endian short if it is equal to MQC.MQENC_INTEGER_REVERSED.

Exceptions

- IOException - if encoding is not equal to either of these values.

readUTF

```
public String readUTF() throws IOException;
```

Reads a UTF format String from the current position in the message buffer.

Returns

- the String.

Exceptions

- IOException - if there is a problem with IO.

resizeBuffer

```
public void resizeBuffer(int size) throws IOException;
```

Indicates to the MQMessage class the size of buffer that might be required. If the message currently contains message data and the new size is less than the current size, the message data is truncated. If this message is subsequently used with MQQueue.get(), then this is the size of buffer allocated for the get request.

Exceptions

- IOException - if there is a problem with IO.

seek

```
public void seek(int offset) throws EOFException;
```

Moves the cursor to a new absolute position in the message buffer. Subsequent reads and writes will start from this position in the buffer.

Parameters

- offset - the new value of the cursor position.

Exceptions

- EOFException - if offset takes cursor outside the message data.

setDataOffset

```
public void setDataOffset(int offset) throws EOFException;
```

Moves the cursor to a new absolute position in the message buffer. This method is identical to seek(), and is provided for cross-language compatibility with the other WebSphere MQ APIs.

Parameters

- offset - the new value of the cursor position.

Exceptions

- EOFException - if offset takes cursor outside the message data.

skipBytes

```
public int skipBytes(int n) throws IOException, EOFException;
```

Moves the cursor forward in the message buffer.

Parameters

- n - the number of bytes to move.

Returns

- the number of bytes actually moved.

Exceptions

- IOException - if there is a problem with IO.
- EOFException - if the skip goes beyond the end of file.

write

```
public void write(byte[] b) throws IOException;
```

Writes an array of bytes into the message buffer at the current position.

Parameters

- b - the array to be written.

Returns

- the array to be written.

Exceptions

- IOException - if there is a problem with IO.

write

```
public void write(byte[] b, int off, int len) throws IOException;
```

Writes a series of bytes into the message buffer at the current position.

Parameters

- b - the array from which the bytes are written.
- off - the offset to the first byte in the array to be written.
- len - the number of bytes to be written.

Exceptions

- IOException - if there is a problem with IO.

write

```
public void write(int b) throws IOException;
```

Writes a byte into the message buffer at the current position.

Parameters

- b - the byte to be written

Exceptions

- IOException - if there is a problem with IO.

writeBoolean

```
public void writeBoolean(boolean v) throws IOException;
```

Writes a boolean into the message buffer at the current position.

Parameters

- v - the boolean to be written.

Exceptions

- IOException - if there is a problem with IO.

writeByte

```
public void writeByte(int v) throws IOException;
```

Writes a byte into the message buffer at the current position.

Parameters

- v - the byte to be written.

Exceptions

- IOException - if there is a problem with IO.

writeBytes

```
public void writeBytes(String s) throws IOException;
```

Writes a String as a sequence of bytes into the message buffer at the current position.

Parameters

- s - the String to be written.

Exceptions

- IOException - if there is a problem with IO.

writeChar

```
public void writeChar(int v) throws IOException;
```

Writes a Unicode character into the message buffer at the current position.

Parameters

- v - the character to be written, expressed as an int.

Exceptions

- IOException - if there is a problem with IO.

writeChars

```
public void writeChars(String s) throws IOException;
```

Writes a String as a sequence of Unicode characters into the message buffer at the current position.

Parameters

- s - the String to be written.

Exceptions

- IOException - if there is a problem with IO.

writeDecimal2

```
public void writeDecimal2(short v) throws IOException;
```

Writes a 2-byte packed decimal format number into the message buffer at the current position. The behavior of this method is determined by `encoding`. A value of MQC.MQENC_DECIMAL_NORMAL writes a big-endian packed decimal and a value of MQC.MQENC_DECIMAL_REVERSED writes a little-endian packed decimal.

Parameters

- v - is the number to be written in the range -999 to 999.

Exceptions

- IOException - if there is a problem with IO.

writeDecimal4

```
public void writeDecimal4(int v) throws IOException;
```

Writes a 4-byte packed decimal format number into the message buffer at the current position. The behavior of this method is determined by `encoding`. A value of MQC.MQENC_DECIMAL_NORMAL writes a big-endian packed decimal and a value of MQC.MQENC_DECIMAL_REVERSED writes a little-endian packed decimal.

Parameters

- v - is the number to be written in the range -9,999,999 to 9,999,999.

Exceptions

- IOException - if there is a problem with IO.

writeDecimal8

```
public void writeDecimal8(long v) throws IOException;
```

Writes an 8-byte packed decimal format number into the message buffer at the current position. The behavior of this method is determined by `encoding`. A value of MQC.MQENC_DECIMAL_NORMAL writes a big-endian packed decimal and a value of MQC.MQENC_DECIMAL_REVERSED writes a little-endian packed decimal.

Parameters

- v - is the number to be written int the range -999,999,999,999,999 to 999,999,999,999,999.

Exceptions

- IOException - if there is a problem with IO.

writeDouble

```
public void writeDouble(double v) throws IOException;
```

Writes a double into the message buffer at the current position. The behavior of this method is determined by `encoding`.

Values of MQC.MQENC_FLOAT_IEEE_NORMAL and MQENC_IEEE_FLOAT_REVERSED write IEEE standard floats in big-endian and little-endian formats respectively.

A value of MQC.MQENC_FLOAT_S390 writes a zSeries format floating point number. Note that the range of IEEE doubles is greater than the range of zSeries double precision floating point numbers, and that very large numbers cannot be converted.

Parameters

- v - the double to be written.

Exceptions

- IOException - if there is a problem with IO.

writeFloat

```
public void writeFloat(float v) throws IOException;
```

Writes a float into the message buffer at the current position. The behavior of this method is determined by `encoding`.

Values of MQC.MQENC_FLOAT_IEEE_NORMAL and MQENC_IEEE_FLOAT_REVERSED write IEEE standard floats in big-endian and little-endian formats respectively.

A value of MQC.MQENC_FLOAT_S390 writes a zSeries format floating point number. Note that the range of IEEE doubles is greater than the range of zSeries double precision floating point numbers, and that very large numbers cannot be converted.

Parameters

- v - the float to be written.

Exceptions

- IOException - if there is a problem with IO.

writeInt

```
public void writeInt(int v) throws IOException;
```

Writes an int into the message buffer at the current position. The behavior of this method is determined by `encoding`.

Values of MQC.MQENC_INTEGER_NORMAL and MQC.MQENC_INTEGER_REVERSED write integers in big-endian and little-endian formats respectively.

Parameters

- v - the int to be written.

Exceptions

- IOException - if there is a problem with IO.

writeInt2

```
public void writeInt2(int v) throws IOException;
```

Identical to writeShort(), provided for cross-language WebSphere MQ API compatibility.

Parameters

- v - the long to be written.

Exceptions

- IOException - if there is a problem with IO.

writeInt4

```
public void writeInt4(int v) throws IOException;
```

Synonym for writeInt(), provided for cross-language WebSphere MQ API compatibility.

Exceptions

- IOException - if there is a problem with IO.

writeInt8

```
public void writeInt8(long v) throws IOException;
```

Synonym for writeLong(), provided for cross-language WebSphere MQ API compatibility.

Exceptions

- IOException - if there is a problem with IO.

writeLong

```
public void writeLong(long v) throws IOException;
```

Writes a long into the message buffer at the current position. The behavior of this method is determined by `encoding`.

Values of MQC.MQENC_INTEGER_NORMAL and MQC.MQENC_INTEGER_REVERSED write longs in big-endian and little-endian formats respectively.

Parameters

- v - the long to be written.

Exceptions

- IOException - if there is a problem with IO.

writeMQMDE

```
public void writeMQMDE() throws IOException, MQException;
```

Writes an extended MQMD object into the message at the current position. Values for the MQMDE are drawn from the field values: `encoding`, characterSet, `format`, groupId, messageSequenceNumber, `offset`, messageFlags and originalLength fields. The current value of the format field is written into the MDE, and the format field is then set to MQFMT_MD_EXTENSION.

Exceptions

- IOException - if there is a problem with IO.
- EOFException - if the read goes beyond the end of file.

writeObject

```
public void writeObject(Object obj) throws IOException;
```

Writes an Object into the message.

Parameters

- obj - the Object to be written.

Exceptions

- IOException - if there is a problem with IO.

writeShort

```
public void writeShort(int v) throws IOException;
```

Writes a short into the message buffer at the current position. The behavior of this method is determined by `encoding`.

Values of MQC.MQENC_INTEGER_NORMAL and MQC.MQENC_INTEGER_REVERSED write shorts in big-endian and little-endian formats respectively.

A value of MQC.MQENC_FLOAT_S390 writes a zSeries format floating point number.

Parameters

- v - the long to be written.

Exceptions

- IOException - if there is a problem with IO.

writeString

```
public void writeString(String s) throws IOException;
```

Writes a String into the message buffer at the current position, converting it to the codeset identified by characterSet.

Parameters

- s - the String to be written.

Exceptions

- IOException - if there is a problem with IO.

writeUTF

```
public void writeUTF(String str) throws IOException;
```

Writes a String in UTF format into the message buffer at the current position.

Parameters

- str - the String to be written.

Exceptions

- IOException - if there is a problem with IO.

MQPoolToken

public class **MQPoolToken**
extends Object

```
java.lang.Object
   |
   +----com.ibm.mq.MQPoolToken
```

The MQPoolToken is used in conjunction with MQEnvironment to allow application components to exercise control over the default connection manager. Typically, an application component constructs an MQPoolToken and registers it with MQEnvironment prior to using WebSphere MQ, and removes the MQPoolToken when it has finished using WebSphere MQ.

The default connection manager can keep track of the number of registered tokens via an MQPoolServices object. It typically destroys any MQManagedConnections in the pool, when the number of registered MQPoolTokens falls to zero.

A connection manager intended for use as the default connection manager can provide a subclass of MQPoolToken. Application components can optionally instantiate the subclass and use this to pass information to the connection manager.

Constructors

MQPoolToken

```
public MQPoolToken();
```

The default constructor.

MQProcess

public class **MQProcess**
extends MQManagedObject

```
java.lang.Object
   |
   +----com.ibm.mq.MQManagedObject
           |
           +----com.ibm.mq.MQProcess
```

MQProcess provides inquire operations for WebSphere MQ processes. Use MQQueueManager.accessProcess() to create an MQProcess object.

Constructors

MQProcess

```
public MQProcess(MQQueueManager qMgr, String processName, int openOptions,
                 String queueManagerName,
                 String alternateUserId) throws MQException;
```

Establishes access to a WebSphere MQ process on the specified queue manager in to inquire about the process attributes. It has been made public to permit subclassing.

Parameters

- qMgr - the queue manager which is running the process.
- processName - the name of process to open.
- openOptions - options which control the opening of the process. As inquire is automatically added to the options specified there is no need to specify it explicitly. Valid options are:
  - MQC.MQOO_ALTERNATE_USER_AUTHORITY
  - MQC.MQOO_FAIL_IF_QUIESCING

  If more than one option is required, the values can be combined using either the '+' or '|' operator.
- queueManagerName - the name of queue manager qMgr.
- alternateUserId - if MQC.MQOO_ALTERNATE_USER_AUTHORITY is specified in the openOptions parameter, this parameter specifies the alternative user ID to be used to check the authorization for the open. Otherwise this parameter can be blank or null.

Exceptions

- MQException - if the open fails.

Methods

close

```
public void close() throws MQException;
```

Closes the process.

Exceptions

- MQException - if the WebSphere MQ call fails.

getApplicationId

```
public String getApplicationId() throws MQException;
```

Gets the character string which identifies the application to be started. This information is used by a trigger monitor application which processes messages on the initiation queue; the information is sent to the initiation queue as part of the trigger message.

Returns

- the application ID.

Exceptions

- MQException - if you call this method after you have closed the process, to indicate that the process is no longer accessible or if the underlying inquire() call fails.

getApplicationType

```
public int getApplicationType() throws MQException;
```

Identifies the nature of the process to be started in response to a trigger message.

Returns

- the following standard types have already been defined but others can be used:
  - MQC.MQAT_AIX
  - MQC.MQAT_CICS
  - MQC.MQAT_IMS
  - MQC.MQAT_MVS
  - MQC.MQAT_OS400
  - MQC.MQAT_UNIX
  - MQC.MQAT_WINDOWS
  - MQC.MQAT_JAVA
  - MQC.MQAT_USER_FIRST
  - MQC.MQAT_USER_LAST

Exceptions

- MQException - if underlying inquire call fails.

getEnvironmentData

```
public String getEnvironmentData() throws MQException;
```

Gets information on the environment of the application that is to be started.

Returns

- the information as a String.

Exceptions

- MQException - if an internal error occurs.

getUserData

```
public String getUserData() throws MQException;
```

Gets information pertaining to the application to be started.

Returns

- the information as a String.

Exceptions

- MQException - if an internal error occurs.

MQPutMessageOptions

public class **MQPutMessageOptions**
extends Object

```
java.lang.Object
   |
   +----com.ibm.mq.MQPutMessageOptions
```

This class contains options that control the behavior of MQQueue.put().

Constructors

MQPutMessageOptions

```
public MQPutMessageOptions();
```

Constructs an object with no options set, and blank resolvedQueueName and resolvedQueueManagerName.

MQPutMessageOptions

```
public MQPutMessageOptions(boolean noReadBack);
```

Constructs an MQPutMessageOptions object; reading the `options` field is optional. You can use this constructor to save some overheads if your application never needs to read back the options field.

Parameters

- noReadBack - if **true** this disables the reading back the options MQPMO field. This avoids the overhead of converting it.

Fields

contextReference

```
public com.ibm.mq.MQQueue
```

An input field that indicates the source of the context information. If the options field includes MQC.MQPMO_PASS_IDENTITY_CONTEXT or MQC.MQPMO_PASS_ALL_CONTEXT, set this field to refer to the MQQueue from which to take the context information. The initial value of this field is null.

contextReferenceHandle

```
public int
```

Handle to the context reference

invalidDestCount

```
public int
```

Number of messages that could not be sent.

knownDestCount

```
public int
```

Number of messages successfully sent to local queues.

options

```
public int
```

Options that control the action of MQQueue.put(). Any or none of the following values can be specified. If more than one option is required, the values can be combined using '+' or '|'.

- MQC.MQPMO_SYNCPOINT
- MQC.MQPMO_NO_SYNCPOINT - default
- MQC.MQPMO_NO_CONTEXT
- MQC.MQPMO_DEFAULT_CONTEXT
- MQC.MQPMO_SET_IDENTITY_CONTEXT
- MQC.MQPMO_SET_ALL_CONTEXT
- MQC.MQPMO_FAIL_IF_QUIESCING
- MQC.MQPMO_NEW_MSG_ID
- MQC.MQPMO_NEW_CORREL_ID
- MQC.MQPMO_LOGICAL_ORDER
- MQC.MQPMO_ALTERNATE_USER_AUTHORITY
- MQC.MQPMO_RESOLVE_LOCAL_Q

recordFields

```
public int
```

Flag which controls the behavior of MQPUT when used with distribution lists.

resolvedQueueManagerName

```
public java.lang.String
```

An output field set by the queue manager to the name of the queue manager that owns the remote queue. This might be different from the name of the queue manager from which the queue was accessed if the queue is a remote queue.

resolvedQueueName

```
public java.lang.String
```

The output field set by the queue manager to the name of the queue on which the message is placed. This might be different from the name used to open the queue, if the opened queue was an alias or model queue.

unknownDestCount

```
public int
```

Number of messages successfully sent to remote queues.

Methods

updateDistributionListItems

```
public void updateDistributionListItems();
```

Copies updates in the response records into the distribution list items. Called from the bindings.

MQQueue

public class **MQQueue**
extends MQManagedObject

```
java.lang.Object
   |
   +----com.ibm.mq.MQManagedObject
           |
           +----com.ibm.mq.MQQueue
```

MQQueue provides inquire, set, put and get operations for WebSphere MQ queues. The inquire and set capabilities are inherited from MQManagedObject.

Use MQQueueManager.accessQueue() to gain access to an MQQueue object.

Constructors

MQQueue

```
public MQQueue(MQQueueManager qMgr, String queueName, int openOptions,
               String queueManagerName,
               String dynamicQueueName, String alternateUserId)
              throws MQException;
```

Public constructor which allows users to create MQQueue subclasses.

Parameters

- qMgr - the object which represents the queue manager on which the queue resides. Valid options are:
  - MQC.MQOO_ALTERNATE_USER_AUTHORITY
  - MQC.MQOO_BIND_AS_Q_DEF
  - MQC.MQOO_BIND_NOT_FIXED
  - MQC.MQOO_BIND_ON_OPEN
  - MQC.MQOO_BROWSE
  - MQC.MQOO_FAIL_IF_QUIESCING
  - MQC.MQOO_INPUT_AS_Q_DEF
  - MQC.MQOO_INPUT_SHARED
  - MQC.MQOO_INPUT_EXCLUSIVE
  - MQC.MQOO_INQUIRE
  - MQC.MQOO_OUTPUT
  - MQC.MQOO_PASS_ALL_CONTEXT
  - MQC.MQOO_PASS_IDENTITY_CONTEXT
  - MQC.MQOO_SAVE_ALL_CONTEXT
  - MQC.MQOO_SET
  - MQC.MQOO_SET_ALL_CONTEXT
  - MQC.MQOO_SET_IDENTITY_CONTEXT
  - MQC.MQOO_RESOLVE_LOCAL_Q

  If more than one option is required, the values can be combined using either the '+' or '|' operator.
- queueName - name of the queue to open.
- openOptions - options which control the opening of the queue.

- queueManagerName - name of the queue manager on which the queue is defined. If it is blank or null the queue manager to which this MQQueueManager object is connected is used.
- dynamicQueueName - specifies the name of the dynamic queue to be created. It is ignored unless queueName specifies the name of a model queue, in which case it must not be blank or null. If the last non-blank character in the name is an asterisk (*), the queue manager replaces the asterisk with a string of characters that guarantees that the name generated for the queue is unique on this queue manager.
- alternateUserId - the alternative user ID used to check the authorization for the open if MQC.MQOO_ALTERNATE_USER_AUTHORITY is specified in openOptions.

Exceptions

- MQException - if the queue cannot be opened.

Methods

close

```
public void close() throws MQException;
```

Closes the object. No further operations on this object are permitted after it is closed. The behavior of the close method can be altered by setting closeOptions.

Exceptions

- MQException - if the WebSphere MQ call fails.

get

```
public void get(MQMessage message) throws MQException;
```

Retrieves a message from the queue, using default get message options. It uses an option setting of MQC.MQGMO_NO_WAIT.

Parameters

- message - an input/output parameter which contains the message descriptor information and the returned message data. On input, some of the fields are used as input parameters, in particular messageId and correlationId. It is important to ensure that these are set as required.

Exceptions

- MQException - if the get fails.

get

```
public void get(MQMessage message, MQGetMessageOptions getMessageOptions)
             throws MQException;
```

Retrieves a message from the queue, regardless of the size of the message. For large messages, this might require two calls to WebSphere MQ, One to establish the required buffer size and one to get the message data itself.

If the call fails, the MQMessage object is unchanged. If it succeeds, the message descriptor fields and message data portions of the MQMessage are completely overwritten by the fields and data from the incoming message.

Note that if you perform a get with wait, all other threads using the same MQQueueManager are blocked until the call completes. If you need multiple threads to access WebSphere MQ simultaneously, then each thread must create its own MQQueueManager object.

Parameters

- message - an input/output parameter which contains the message descriptor information and the returned message data. On input some of the fields are used as input parameters, in particular messageId and correlationId. It is important to ensure that these are set as required.
- getMessageOptions - options which control the action of the get. See MQGetMessageOptions.options for details.

Exceptions

- MQException - if the get fails.

get

```
public void get(MQMessage message, MQGetMessageOptions getMessageOptions,
                int MaxMsgSize) throws MQException;
```

Retrieves a message from the queue, up to a maximum specified message size.

If the call fails, the MQMessage object is unchanged. If it succeeds, the message descriptor fields and message data portions of the MQMessage are completely overwritten by the fields and data from the incoming message.

If you perform a get with wait, all other threads using the same MQQueueManager are blocked until the call completes. If you need multiple threads to access WebSphere MQ simultaneously, then each thread must create its own MQQueueManager object.

Parameters

- message - an input/output parameter which contains the message descriptor information and the returned message data. On input, some of the fields are used as input parameters, in particular messageId and correlationId. It is important to ensure that these are set as required.
- getMessageOptions - options which control the action of the get. See MQGetMessageOptions.options for details.
- MaxMsgSize - the largest message this call can to receive. If the message on the queue is larger than this, the result depends on whether the MQC.MQGMO_ACCEPT_TRUNCATED_MSG flag has been selected in MQGetMessageOptions.options.

  If the flag has been set, the message is filled with as much of the message data as will fit in the specified buffer size, and an MQException is thrown with completion code MQCC_WARNING and reason code MQRC_TRUNCATED_MSG_ACCEPTED.

  If the flag has not been set, the message is left on the queue and an MQException is thrown with completion code MQCC_FAILED and reason code MQRC_TRUNCATED_MSG_FAILED.

Exceptions

- MQException - if the get fails.

getCreationDateTime

```
public GregorianCalendar getCreationDateTime() throws MQException;
```

Gets the date and time that this queue was created.

Returns

- the date and time.

Exceptions

- MQException - if you call this method after you have closed the queue, to indicate that the queue is no longer accessible.

getCurrentDepth

```
public int getCurrentDepth() throws MQException;
```

Gets the number of messages currently on the queue. This value is incremented during a put call and during backout of a get call. It is decremented during a non-browse get and during backout of a put call.

Returns

- the number of messages.

Exceptions

- MQException - if you call this method after you have closed the queue, to indicate that the queue is no longer accessible.

getDefinitionType

```
public int getDefinitionType() throws MQException;
```

Indicates how the queue was defined.

Returns

- one of the following:
  - MQC.MQQDT_PREDEFINED
  - MQC.MQQDT_PERMANENT_DYNAMIC
  - MQC.MQQDT_TEMPORARY_DYNAMIC

Exceptions

- MQException - if you call this method after you have closed the queue, to indicate that the queue is no longer accessible.

getInhibitGet

```
public int getInhibitGet() throws MQException;
```

Indicates whether get operations are allowed for this queue.

Returns

- one of the following:
  - MQC.MQQA_GET_INHIBITED
  - MQC.MQQA_GET_ALLOWED

Exceptions

- MQException - if you call this method after you have closed the queue, to indicate that the queue is no longer accessible.

getInhibitPut

```
public int getInhibitPut() throws MQException;
```

Indicates whether put operations are allowed for this queue.

Returns

- one of the following:
  - MQC.MQQA_PUT_INHIBITED
  - MQC.MQQA_PUT_ALLOWED

Exceptions

- MQException - if you call this method after you have closed the queue, to indicate that the queue is no longer accessible.

getMaximumDepth

```
public int getMaximumDepth() throws MQException;
```

Gets the maximum number of messages that can exist on the queue at any one time. An attempt to put a message to a queue that already contains this many messages fails with reason code MQException.MQRC_Q_FULL .

Returns

- the maximum number of messages.

Exceptions

- MQException - if you call this method after you have closed the queue, to indicate that the queue is no longer accessible.

getMaximumMessageLength

```
public int getMaximumMessageLength() throws MQException;
```

Gets the maximum length of the application data of a message on this queue. An attempt to put a message larger than this value fails with reason code MQException.MQRC_MSG_TOO_BIG_FOR_Q.

Returns

- the maximum length.

Exceptions

- MQException - if you call this method after you have closed the queue, to indicate that the queue is no longer accessible.

getOpenInputCount

```
public int getOpenInputCount() throws MQException;
```

Gets the number of currently valid handles for removing messages from the queue.

Returns

- the *total* number of valid handles known to the local queue manager, not just those created by the WebSphere MQ Client for Java (using accessQueue()).

Exceptions

- MQException - if you call this method after you have closed the queue, to indicate that the queue is no longer accessible.

getOpenOutputCount

```
public int getOpenOutputCount() throws MQException;
```

Gets the number of currently valid handles for adding messages to the queue.

Returns

- the *total* number of valid handles known to the local queue manager, not just those created by the WebSphere MQ Client for Java (using accessQueue()).

Exceptions

- MQException - if you call this method after you have closed the queue, to indicate that the queue is no longer accessible.

getQueueType

```
public int getQueueType() throws MQException;
```

Gets the type of this queue.

Returns

- one of the following:
  - MQC.MQQT_ALIAS
  - MQC.MQQT_LOCAL
  - MQC.MQQT_MODEL
  - MQC.MQQT_REMOTE

Exceptions

- MQException - if you call this method after you have closed the queue, to indicate that the queue is no longer accessible.

getShareability

```
public int getShareability() throws MQException;
```

Indicates whether the queue can be opened multiple times for input.

Returns

- one of the following:
  - MQC.MQQA_SHAREABLE
  - MQC.MQQA_NOT_SHAREABLE

Exceptions

- MQException - if you call this method after you have closed the queue, to indicate that the queue is no longer accessible.

getTriggerControl

```
public int getTriggerControl() throws MQException;
```

Indicates whether trigger messages are written to an initiation queue. This starts an application to service the queue.

Returns

- the possible values are:
  - MQC.MQTC_OFF
  - MQC.MQTC_ON

Exceptions

- MQException - if you call this method after you have closed the queue, to indicate that the queue is no longer accessible.

getTriggerData

```
public String getTriggerData() throws MQException;
```

Gets the data for the trigger message that is written to the initiation queue. The trigger message is written to the initiation queue when a message arrives on this one.

Returns

- the data in free format.

Exceptions

- MQException - if you call this method after you have closed the queue, to indicate that the queue is no longer accessible.

getTriggerDepth

```
public int getTriggerDepth() throws MQException;
```

Gets the number of messages that have to be on the queue to generate a trigger message. This applies when the trigger type is MQC.MQTT_DEPTH.

Returns

- the number of messages.

Exceptions

- MQException - if you call this method after you have closed the queue, to indicate that the queue is no longer accessible.

getTriggerMessagePriority

```
public int getTriggerMessagePriority() throws MQException;
```

Gets the message priority below which messages do not cause trigger messages. That is, the queue manager ignores these messages when deciding whether to generate a trigger.

Returns

- the message priority. Zero means that all messages contribute to the generation of trigger messages.

Exceptions

- MQException - if you call this method after you have closed the queue, to indicate that the queue is no longer accessible.

getTriggerType

```
public int getTriggerType() throws MQException;
```

Indicates the conditions under which trigger messages are written. Trigger messages are written as a result of messages arriving on this queue.

Returns

- the possible values are:
  - MQC.MQTT_NONE
  - MQC.MQTT_FIRST
  - MQC.MQTT_EVERY
  - MQC.MQTT_DEPTH

Exceptions

- MQException - if you call this method after you have closed the queue, to indicate that the queue is no longer accessible.

put

```
public void put(MQMessage message) throws MQException;
```

Puts a message onto the queue using default put message options.

Parameters

- message - an object which contains the message descriptor data and the message data to be sent.

Exceptions

- MQException - if the put fails.

put

```
public void put(MQMessage message, MQPutMessageOptions putMessageOptions)
             throws MQException;
```

Puts a message onto the queue.

Modifications to the MQMessage object after the put has completed do not affect the actual message on the queue.

If you use the same MQMessage object to make further calls, then performing a put updates MQMessage.messageId and MQMessage.correlationId.

Note also that calling put does not clear the message data. For example:

```
msg.writeString("a");
q.put(msg,pmo);
msg.writeString("b");
q.put(msg,pmo);
```

results in ″ab″ being put by the second call.

Parameters

- message - an object which contains the message descriptor data and the message data to be sent. On completion the values are set to those of the message which was put on the queue.
- putMessageOptions - Options controlling the action of the put. See MQPutMessageOptions.options for details.

Exceptions

- MQException - if the put fails.

setInhibitGet

```
public void setInhibitGet(int inhibit) throws MQException;
```

Controls whether get operations are allowed for this queue.

Parameters

- inhibit - the permissible values are:
  - MQC.MQQA_GET_INHIBITED
  - MQC.MQQA_GET_ALLOWED

Exceptions

- MQException - if you call this method after you have closed the queue, to indicate that the queue is no longer accessible.

setInhibitPut

```
public void setInhibitPut(int inhibit) throws MQException;
```

Controls whether put operations are allowed for this queue.

Parameters

- inhibit - the permissible values are:
  - MQC.MQQA_PUT_INHIBITED
  - MQC.MQQA_PUT_ALLOWED

Exceptions

- MQException - if you call this method after you have closed the queue, to indicate that the queue is no longer accessible.

setTriggerControl

```
public void setTriggerControl(int trigger) throws MQException;
```

Controls whether trigger messages are written to an initiation queue. This starts an application to service the queue.

Parameters

- trigger - the permissible values are:
  - MQC.MQTC_OFF
  - MQC.MQTC_ON

Exceptions

- MQException - if you call this method after you have closed the queue, to indicate that the queue is no longer accessible.

setTriggerData

```
public void setTriggerData(String data) throws MQException;
```

Sets the data for the trigger message that is written to the initiation queue. The trigger message is written to the initiation queue when a message arrives on this one.

Parameters

- data - sets the data in free-format. The maximum permissible length of the string is given by MQC.MQ_TRIGGER_DATA_LENGTH.

Exceptions

- MQException - if you call this method after you have closed the queue, to indicate that the queue is no longer accessible.

setTriggerDepth

```
public void setTriggerDepth(int depth) throws MQException;
```

Sets the number of messages that have to be on the queue to generate a trigger message. This applies when trigger type is MQC.MQTT_DEPTH .

Parameters

- depth - the number of messages.

Exceptions

- MQException - if you call this method after you have closed the queue, to indicate that the queue is no longer accessible.

setTriggerMessagePriority

```
public void setTriggerMessagePriority(int priority) throws MQException;
```

Sets the message priority below which messages do not cause trigger messages. That is, the queue manager ignores these messages when deciding whether to generate a trigger.

Parameters

- priority - the message priority. Zero means that all messages contribute to the generation of trigger messages.

Exceptions

- MQException - if you call this method after you have closed the queue, to indicate that the queue is no longer accessible.

setTriggerType

```
public void setTriggerType(int type) throws MQException;
```

Sets the conditions under which trigger messages are written. Trigger messages are written as a result of messages arriving on this queue.

Parameters

- type - the possible values are:
  - MQC.MQTT_NONE
  - MQC.MQTT_FIRST
  - MQC.MQTT_EVERY
  - MQC.MQTT_DEPTH

Exceptions

- MQException - if you call this method after you have closed the queue, to indicate that the queue is no longer accessible.

MQQueueManager

public class **MQQueueManager**
extends MQManagedObject

```
java.lang.Object
   |
   +----com.ibm.mq.MQManagedObject
           |
           +----com.ibm.mq.MQQueueManager
```

The MQQueueManager class provides a connection to a WebSphere MQ queue manager.

An MQQueueManager object (and any queues or processes accessed through it) can be shared between multiple threads, but be aware that access to the WebSphere MQ queue manager itself is `synchronized`, so that only one thread can communicate with it at any one time. A call to MQQueue.get() specifying MQC.MQGMO_WAIT (for example) will therefore block any other threads attempting to make WebSphere MQ calls using the same MQQueueManager until the get completes.

Constructors

MQQueueManager

```
public MQQueueManager(String queueManagerName) throws MQException;
```

Creates a connection to the named queue manager.

The host name, channel name and port to use during the connection request are specified in the MQEnvironment class. This must be done *before* calling this constructor.

The following example shows a connection to a queue manager ″MYQM″, running on a system with host name ″fred.mq.com″.

```
MQEnvironment.hostname = "fred.mq.com";  // host to connect to
MQEnvironment.port     = -1  ;           // port to connect to.  If not set,
                                         // this defaults to 1414 for WebSphere MQ
                                         // client connections.
MQEnvironment.channel  = "channel.name"; // the CASE-SENSITIVE name of the SVRCONN
                                         // channel on the queue manager
MQQueueManager qMgr    = new MQQueueManager("MYQM");
```

If the queue manager name is null or blank, then a connection is made to the default queue manager.

Parameters

- queueManagerName - the name of the queue manager to which to connect.

Exceptions

- MQException - if there are connection problems.

MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      ConnectionManager connectionManager)
              throws MQException;
```

Creates a connection to the named queue manager specifying a connection manager.

Parameters

- queueManagerName - the name of the queue manager.
- connectionManager - the connection manager which will handle this connection.

Exceptions

- MQException - if there are connection problems.

MQQueueManager

```
public MQQueueManager(String queueManagerName, Hashtable properties)
             throws MQException;
```

Creates a connection to the named queue manager using a Hashtable. The Hashtable overrides the specification held in MQEnvironment.

Parameters

- queueManagerName - the name of the queue manager.
- properties - connection properties.

Exceptions

- MQException - if there are connection problems.

MQQueueManager

```
public MQQueueManager(String queueManagerName, Hashtable properties,
                      ConnectionManager connectionManager)
             throws MQException;
```

Creates a connection to the named queue manager using given Hashtable and connection manager. The given properties override those held in MQEnvironment.

Parameters

- queueManagerName - the name of the queue manager.
- properties - connection properties.
- connectionManager - the ConnectionManager which handles this connection.

Exceptions

- MQException - if there are connection problems.

MQQueueManager

```
public MQQueueManager(String queueManagerName, Hashtable properties,
                      MQConnectionManager connectionManager)
             throws MQException;
```

Creates a connection to a queue manager which overrides the settings in MQEnvironment class with those in the given Hashtable.

Parameters

- queueManagerName - the name of the queue manager.
- properties - connection properties.
- connectionManager - the connection manager which will handle this connection.

Exceptions

- MQException - raised if there are connection problems.

MQQueueManager

```
public MQQueueManager(String queueManagerName, Hashtable properties,
                      URL url) throws MQException;
```

Creates a connection to the named queue manager using a channel definition table. Using a client channel definition table enables alternative channel definitions to be defined. The constructor selects a set of definitions from the table and these are used instead of any settings held in MQEnvironment class when opening a channel. Properties other than those defined by the client channel definition table can be supplied with this constructor.

Parameters

- queueManagerName - the queue manager which is used when selecting a channel definition. This can be in of the following forms:
  - ″qMgrName″, where the actual name of the required queue manager is passed in. The channel must connect to a queue manager of this name.
  - ″*qMgrName″, where ″*″ followed by the actual name of the required queue manager is passed in. The channel definition that is used must specify this queue manager name. This full name is passed onto the queueManager during a connect, but the queue manager that is ultimately connected to might not have the same name as specified here after the ′*′.
  - ″*″ or ″″ or a name which consists entirely of blanks is passed in. The actual queue manager name is disregarded when a channel definition is being selected.
- properties - A Hashtable of properties to be used to establish the connection with those defined in the client channel definition that is actually used. Any properties that are not valid for this type of connection will be ignored.
- url - the URL which specifies the channel definition file to be used in connecting to the queue manager.

Exceptions

- MQException - raised if there are connection problems.

MQQueueManager

```
public MQQueueManager(String queueManagerName, int options)
             throws MQException;
```

Creates a connection to the named queue manager specifying binding options.

Parameters

- queueManagerName - the name of the queue manager.
- options - binding options. Possible values are:
  - MQC.MQCNO_FASTPATH_BINDING
  - MQC.MQCNO_STANDARD_BINDING
  - MQC.MQCNO_SHARED_BINDING
  - MQC.MQCNO_ISOLATED_BINDING

Exceptions

- MQException - if there are connection problems.

MQQueueManager

```
public MQQueueManager(String queueManagerName, int options,
                      ConnectionManager connectionManager)
              throws MQException;
```

Creates a connection to the named queue manager specifying bindings options and a connection manager.

Parameters

- queueManagerName - the name of the queue manager.
- options - binding options. Possible values are:
  - MQC.MQCNO_FASTPATH_BINDING
  - MQC.MQCNO_STANDARD_BINDING
  - MQC.MQCNO_SHARED_BINDING
  - MQC.MQCNO_ISOLATED_BINDING
- connectionManager - the Connection manager which handles this connection

Exceptions

- MQException - if there are connection problems.

MQQueueManager

```
public MQQueueManager(String queueManagerName, int options,
                      MQConnectionManager connectionManager)
              throws MQException;
```

Creates a connection to a queue manager allowing binding options to be specified. It also allows a connection manager to be specified.

Parameters

- queueManagerName - the name of the queue manager.
- options - binding options. Possible values are:
  - MQC.MQCNO_FASTPATH_BINDING
  - MQC.MQCNO_STANDARD_BINDING
  - MQC.MQCNO_SHARED_BINDING
  - MQC.MQCNO_ISOLATED_BINDING
- connectionManager - the connection manager which will handle this connection.

Exceptions

- MQException - if there are connection problems.

MQQueueManager

```
public MQQueueManager(String queueManagerName,
                      MQConnectionManager connectionManager)
              throws MQException;
```

Creates a connection to a named queue manager using a connection manager.

Parameters

- queueManagerName - the name of the queue manager.
- connectionManager - the connection manager which will handle this connection.

Exceptions

- MQException - if there are connection problems.

MQQueueManager

```
public MQQueueManager(String queueManagerName, URL url)
             throws MQException;
```

Creates a connection to the named queue manager using a channel definition table. Using a client channel definition table enables alternative channel definitions to be defined. The constructor selects a set of definitions from the table and these are used instead of any settings held in MQEnvironment class when opening a channel.

Parameters

- queueManagerName - the queue manager which is used when selecting a channel definition. This can be in of the following forms:
  - ″qMgrName″, where the actual name of the required queue manager is passed in. The channel must connect to a queue manager of this name.
  - ″*qMgrName″, where ″*″ followed by the actual name of the required queue manager is passed in. The channel definition that is used must specify this queue manager name. This full name is passed onto the queue manager during a connect, but the queue manager that is ultimately connected to might not have the same name as specified here after the ′*′.
  - ″*″ or ″″ or a name which consists entirely of blanks is passed in. The actual queue manager name is disregarded when a channel definition is being selected.
- url - the URL which specifies the channel definition file to be used in connecting to the queue manager.

Exceptions

- MQException - raised if there are connection problems.

MQQueueManager

```
public MQQueueManager(String queueManagerName, URL url,
                      ConnectionManager connectionManager)
             throws MQException;
```

Creates a connection to the named queue manager using a client channel definition table. Using a client channel definition table enables alternative channel definitions to be defined. The constructor selects a set of definitions from the table and these are used instead of any settings held in MQEnvironment class when opening a channel. The channel definition that is used must specify this queue manager name. This full name is passed onto the queue manager during a connect, but the queue manager that is ultimately connected to might not have the same name as specified here after the “*” ″*″ or ″″ or a name which consists entirely of blanks is passed in. The actual queue manager name is disregarded when a channel definition is being selected.

Parameters

- queueManagerName - the queue manager which is used when selecting a channel definition. This can be in of the following forms:
  - ″qMgrName″, where the actual name of the required queue manager is passed in. The channel must connect to a queue manager of this name.

- ″*qMgrName″, where ″*″ followed by the actual name of the required queue manager is passed in. The channel definition that is used must specify this queue manager name. This full name is passed onto the queue manager during a connect, but the queue manager that is ultimately connected to might not have the same name as specified here after the “*”.
- ″*″ or ″″ or a name which consists entirely of blanks is passed in. The actual queue manager name is disregarded when a channel definition is being selected.

- url - the URL which specifies the channel definition table to be used in connecting to the queue manager.
- connectionManager - the connection manager.

Exceptions

- MQException - raised if there are connection problems.

MQQueueManager

```
public MQQueueManager(String queueManagerName, URL url,
                      MQConnectionManager connectionManager)
              throws MQException;
```

Creates a connection to the named queue manager using a client channel definition table. Using a client channel definition table enables alternative channel definitions to be defined. The constructor selects a set of definitions from the table and these are used instead of any settings held in MQEnvironment class when opening a channel.

Parameters

- queueManagerName - the queue manager which is used when selecting a channel definition. This can be in of the following forms:
  - ″qMgrName″, where the actual name of the required queue manager is passed in. The channel must connect to a queue manager of this name.
  - ″*qMgrName″, where ″*″ followed by the actual name of the required queue manager is passed in. The channel definition that is used must specify this queue manager name. This full name is passed onto the queue manager during a connect, but the queue manager that is ultimately connected to might not have the same name as specified here after the “*”.
  - ″*″ or ″″ or a name which consists entirely of blanks is passed in. The actual queue manager name is disregarded when a channel definition is being selected.
- url - the URL which specifies the channel definition file to be used in connecting to the queue manager.
- connectionManager - the connection manager.

Exceptions

- MQException - raised if there are connection problems.

Fields

isConnected

```
public boolean
```

Deprecated

Use the isConnected() method instead.

Indicates whether this MQQueueManager object is currently connected to a WebSphere MQ queue manager. Use the disconnect method to disconnect from a queue manager.

Methods

accessDistributionList

```
public MQDistributionList accessDistributionList(MQDistributionListItem[] items,

                                                 int options)
             throws MQException;
```

Creates a distribution list using the default alternative user ID.

Parameters

- items - the elements of the distribution list.
- options - the open options for the distribution list.

Returns

- the distribution list.

Exceptions

- MQException - if there is a problem opening the distribution list.

accessDistributionList

```
public MQDistributionList accessDistributionList(MQDistributionListItem[] items,

                                                 int options,
                                                 String id)
             throws MQException;
```

Creates a distribution list.

Parameters

- items - the elements of the distribution list.
- options - the open options for the distribution list.
- id - the alternative user ID.

Returns

- the distribution list.

Exceptions

- MQException - if there is a problem opening the distribution list.

accessProcess

```
public MQProcess accessProcess(String processName, int openOptions)
             throws MQException;
```

Accesses a WebSphere MQ process on this queue manager using default queue manager name and alternative user ID values.

Parameters

- processName - name of process to open.
- openOptions - see openOptions for details.

Returns

- MQProcess which has been successfully opened.

Exceptions

- MQException - if the open fails.

accessProcess

```
public MQProcess accessProcess(String processName, int openOptions,
                               String queueManagerName,
                               String alternateUserId)
            throws MQException;
```

Establishes access to a WebSphere MQ process on this queue manager in order to inquire about the process attributes.

Parameters

- processName - name of process to open
- openOptions - options which control the opening of the process. As inquire is automatically added to the options specified there is no need to specify it explicitly. Valid options are:
  - MQC.MQOO_ALTERNATE_USER_AUTHORITY
  - MQC.MQOO_FAIL_IF_QUIESCING

  If more than one option is required the values can be combined using either the '+' or '|' operator.
- queueManagerName - name of the queue manager on which the process is defined. A name which is entirely blank or null denotes the queue manager to which this object is connected.
- alternateUserId - if MQC.MQOO_ALTERNATE_USER_AUTHORITY is specified in the openOptions parameter, this parameter specifies the alternative user ID to be used to check the authorization for the open. Otherwise this parameter can be blank or null.

Returns

- MQProcess which has been successfully opened.

Exceptions

- MQException - if the open fails.

accessQueue

```
public MQQueue accessQueue(String queueName, int openOptions)
            throws MQException;
```

Establishes access to an WebSphere MQ queue on this queue manager using default queue manager name and alternative user ID values.

Parameters

- queueName - name of queue to open.
- openOptions - options which control the opening of the queue. See accessQueue(String, int, String, String, String) for more information.

Returns

- MQQueue which has been successfully opened.

Exceptions

- MQException - if the open fails.

accessQueue

```
public MQQueue accessQueue(String queueName, int openOptions,
                           String queueManagerName,
                           String dynamicQueueName,
                           String alternateUserId)
             throws MQException;
```

Establishes access to a WebSphere MQ queue on this queue manager in order to get or browse messages, put messages, inquire about the attributes of the queue, or set the attributes of the queue.

If the queue named is a model queue, then a dynamic local queue is created. The name of the created queue is held in the `name` field of the returned MQQueue object.

Parameters

- queueName - name of queue to open.
- openOptions - options which control the opening of the queue. As inquire and set options are automatically added to the options provided, there is no need to specify these explicitly. The valid options are:
  - MQC.MQOO_BROWSE
  - MQC.MQOO_INPUT_AS_Q_DEF
  - MQC.MQOO_INPUT_SHARED
  - MQC.MQOO_INPUT_EXCLUSIVE
  - MQC.MQOO_OUTPUT
  - MQC.MQOO_SAVE_ALL_CONTEXT
  - MQC.MQOO_PASS_IDENTITY_CONTEXT
  - MQC.MQOO_PASS_ALL_CONTEXT
  - MQC.MQOO_SET_IDENTITY_CONTEXT
  - MQC.MQOO_SET_ALL_CONTEXT
  - MQC.MQOO_ALTERNATE_USER_AUTHORITY
  - MQC.MQOO_FAIL_IF_QUIESCING

  If more than one option is required the values can be combined using either the '+' or '|' operator. See *WebSphere MQ Application Programming Reference* for a fuller description of these options.
- queueManagerName - name of the queue manager on which the queue is defined. A name which is blank, or which is null, denotes the queue manager to which this object is connected.
- dynamicQueueName - name of the dynamic queue to be created. This parameter is ignored unless queueName specifies the name of a model queue. If it does, this parameter specifies the name of the dynamic queue to be created. A blank or null name is not valid if queueName specifies the name of a model queue. If the last non-blank character in the name is an asterisk (*), the queue manager replaces it with a string of characters which guarantees that the name generated for the queue is unique at the local queue manager. Asterisk is only valid in positions 1 to 33 of the dynamicQueueName parameter.
- alternateUserId - if MQC.MQOO_ALTERNATE_USER_AUTHORITY is specified in the openOptions parameter this parameter specifies the alternate user ID to be used to check the authorization for the open. If MQC.MQOO_ALTERNATE_USER_AUTHORITY is not specified, this parameter can be left blank (or null).

Returns

- the MQQueue which has been successfully opened.

Exceptions

- MQException - if the open fails.

backout

```
public void backout() throws MQException;
```

Indicates to the queue manager that all the message gets and puts that have occurred since the last syncpoint are to be backed out. Messages sent as part of a unit of work (with the MQC.MQPMO_SYNCPOINT flag set in the options field of MQPutMessageOptions) are deleted. Messages retrieved as part of a unit of work (with the MQC.MQGMO_SYNCPOINT flag set in MQGetMessageOptions.options) are reinstated on the queue.

Exceptions

- MQException - if the call fails.

begin

```
public void begin() throws MQException;
```

Begins a new unit of work. This method is only supported by WebSphere MQ in a bindings connection. It signals to the queue manager that a new unit of work is to begin.

Exceptions

- MQException - if the call fails

commit

```
public void commit() throws MQException;
```

Indicates to the queue manager that the application has reached a syncpoint. All the message gets and puts that have occurred since the last syncpoint are to be made permanent. Messages sent as part of a unit of work (with the MQC.MQPMO_SYNCPOINT flag set in MQPutMessageOptions.options) are made available to other applications. Messages retrieved as part of a unit of work (with the MQC.MQGMO_SYNCPOINT flag set in MQGetMessageOptions.options) are deleted.

Exceptions

- MQException - if the call fails.

disconnect

```
public void disconnect() throws MQException;
```

Ends the connection to the queue manager. All open queues and processes accessed through this queue manager are closed and become unusable. The only way to reconnect is to create a new MQQueueManager object.

Exceptions

- MQException - if the MQ disconnect call fails.

getCCDTURL

```
public URL getCCDTURL();
```

Returns the URL of the channel definition file, or null if it does not exist. Java URLs support various protocols, which normally include file, HTTP, FTP and LDAP. The URL class has several constructors, one of which is: URL(String spec).

Returns

- The URL of the channel definition file.

getCharacterSet

```
public int getCharacterSet() throws MQException;
```

Gets the CCSID (Coded Character Set Identifier) of the queue manager's codeset. This defines the character set used by the queue manager for all character string fields in the application programming interface.

Returns

- the CCSID.

Exceptions

- MQException - if you call this method after disconnecting from the queue manager to indicate that the connection is no longer valid.

getCommandInputQueueName

```
public String getCommandInputQueueName() throws MQException;
```

Gets the name of the command input queue defined on the queue manager. This is a queue to which applications can send commands, if authorized to do so.

Returns

- the name of the command input queue.

Exceptions

- MQException - if you call this method after disconnecting from the queue manager to indicate that the connection is no longer valid.

getCommandLevel

```
public int getCommandLevel() throws MQException;
```

Indicates the level of system control commands supported by the queue manager. The set of system control commands corresponding to a particular command level varies according to the platform on which the queue manager is running. See *WebSphere MQ Application Programming Reference*.

Returns

- values between MQC.MQCMDL_LEVEL_1 and MQC.MQCMDL_LEVEL_600.

Exceptions

- MQException - if you call this method after disconnecting from the queue manager to indicate that the connection is no longer valid.

getDistributionListCapable

```
public boolean getDistributionListCapable();
```

Indicates whether the queue manager supports distribution lists.

Returns

- **true**if distribution lists are supported.

getJDBCConnection

```
public Connection getJDBCConnection(XADataSource xads) throws MQException,
           SQLException, Exception;
```

Returns a Connection object for use with the JTA-JDBC support.

Parameters

- xads - A database-specific implementation of the XADataSource interface that defines the details of the database to connect to. See the documentation for your database to determine how to create an appropriate XADataSource object to pass into this method.

Returns

- A connection for use with the JTA-JDBC support.

Exceptions

- MQException - if there is a WebSphere MQ failure.
- SQLException - if there are problems getting the Connection object.
- Exception - thrown to avoid problems with the JVM verifier for customers who are not using the JTA functionality. The actual exception thrown is javax.transaction.xa.XAException, which requires the jta.jar file to be added to the CLASSPATH for programs that did not previously require it.

getJDBCConnection

```
public Connection getJDBCConnection(XADataSource xads, String userid,
                                    String password)
              throws MQException, SQLException, Exception;
```

Registers a database for coordination. Used to create a JDBC Connection which is coordinated by the queue manager after a call to begin.

Parameters

- xads - database-specific implementation of the XADataSource interface that defines the details of the database. See the documentation for your database to determine how to create an appropriate XADataSource object to pass into this method.
- userid - the user ID for connecting to the database.
- password - the password for connecting to the database.

Returns

- connection for use with the JTA-JDBC support.

Exceptions

- MQException - if there is a WebSphere MQ failure.
- SQLException - if there are problems getting the Connection object.
- Exception - thrown to avoid problems with the JVM verifier for customers who are not using the JTA functionality. The actual exception thrown is javax.transaction.xa.XAException, which requires the jta.jar file to be added to the CLASSPATH for programs that did not previously require it.

getMaximumMessageLength

```
public int getMaximumMessageLength() throws MQException;
```

Gets the maximum length of a message that the queue manager can handle. No queue can be defined with a maximum message length greater than this.

Returns

- The maximum message length in bytes.

Exceptions

- MQException - if you call this method after disconnecting from the queue manager to indicate that the connection is no longer valid.

getMaximumPriority

```
public int getMaximumPriority() throws MQException;
```

Gets the maximum message priority supported by the queue manager. Priorities range from zero (lowest) to this value.

Returns

- the maximum message priority.

Exceptions

- MQException - if you call this method after disconnecting from the queue manager to indicate that the connection is no longer valid.

getProperties

```
public static Hashtable getProperties(Object key, MQChannelHeader header,
                                      URL url) throws IOException,
           MQException;
```

Reads properties concerning a channel in the channel definition table.

Parameters

- key - the name of the channel.
- header - helps to find the channel information in the table.
- url - the URL of the channel definition table.

Returns

- the relevant properties in a Hashtable.

Exceptions

- IOException - raised if there are connection problems.
- MQException - raised if there are WebSphere MQ problems.

getSyncpointAvailability

```
public int getSyncpointAvailability() throws MQException;
```

Indicates whether the queue manager supports units of work and syncpointing with the MQQueue.get() and MQQueue.put() methods.

Returns

- MQC.MQSP_AVAILABLE if syncpointing is available, or MQC.MQSP_NOT_AVAILABLE if not.

Exceptions

- MQException - if you call this method after disconnecting from the queue manager to indicate that the connection is no longer valid.

isConnected

```
public boolean isConnected();
```

Indicates whether this object is currently connected to a WebSphere MQ queue manager. Use disconnect() to disconnect from a queue manager.

Returns

- **true** if connected.

put

```
public void put(String qName, MQMessage msg) throws MQException;
```

Puts a single message on to a (possibly unopened) queue. If a send exit has been specified it processes the message before it is sent.

Parameters

- qName - the name of the queue to which the message is put.
- msg - the message to be sent.

Exceptions

- MQException - if the WebSphere MQ put call fails.

put

```
public void put(String qName, MQMessage msg, MQPutMessageOptions pmo)
             throws MQException;
```

Puts a single message on to a (possibly unopened) queue. If a send exit has been specified it processes the message before it is sent.

Parameters

- qName - the name of the queue to which the message is put.
- msg - the message to be sent.
- pmo - the put message options to use.

Exceptions

- MQException - if the WebSphere MQ put call fails.

put

```
public void put(String qName, String qmName, MQMessage msg)
             throws MQException;
```

Puts a single message on to a (possibly unopened) queue. If a send exit has been specified it processes the message before it is sent.

Parameters

- qName - the name of the queue to which the message is put.
- qmName - the name of the queue manager which holds the queue.
- msg - the message to be sent.

Exceptions

- MQException - if the WebSphere MQ put call fails.

put

```
public void put(String qName, String qmName, MQMessage msg,
               MQPutMessageOptions pmo) throws MQException;
```

Puts a single message on to a (possibly unopened) queue. If a send exit has been specified it processes the message before it is sent.

Parameters

- qName - the name of the queue to which the message is put.

- qmName - the name of the queue manager which holds the queue.
- msg - the message to be sent.
- pmo - the put message options to use.

Exceptions

- MQException - if the WebSphere MQ put call fails.

put

```
public void put(String qName, String qmName, MQMessage msg,
                MQPutMessageOptions pmo, String altUserId)
              throws MQException;
```

Puts a single message onto a (possibly unopened) queue. If a send exit has been specified, it processes the message before it is sent. See the description of MQPUT1 in *WebSphere MQ Application Programming Reference* for more information.

Parameters

- qName - the name of the queue to which the message is put.
- qmName - the name of the queue manager which holds the queue.
- msg - the message to be sent.
- pmo - the put message options to use.
- altUserId - alternative user ID to use when putting the message.

Exceptions

- MQException - if the WebSphere MQ put call fails.

MQReceiveExitChain

public class **MQReceiveExitChain**
extends MQExitChain
implements MQReceiveExit

```
java.lang.Object
   |
   +----com.ibm.mq.MQExitChain
           |
           +----com.ibm.mq.MQReceiveExitChain
```

Chains receive exits together. The exits are of class MQReceiveExit ; as well as exits written in Java, this includes non-Java receive exits made available by means of the MQExternalReceiveExit class.

Constructors

MQReceiveExitChain

```
public MQReceiveExitChain();
```

The default constructor. Creates a Receive Exit Chain.

MQReceiveExitChain

```
public MQReceiveExitChain(List collection);
```

Constructor.

Parameters

- collection - a List object which defines the receive exits which are to be chained.

Methods

receiveExit

```
public byte[] receiveExit(MQChannelExit channelExitParms,
                          MQChannelDefinition channelDefinition,
                          byte[] agentBuffer);
```

Calls the receive exit. This is normally made by the Java client code.

Parameters

- channelExitParms - the definition of the chain of exits.
- channelDefinition - the definition of the channel.
- agentBuffer - the message being passed into the chain of exits.

Returns

- agentBuffer the data to be processed. If the exit response code (in channelExitParms is MQXCC_OK the WebSphere MQ Client for Java can process the data. The simplest receiveExit method therefore, consists of a single line.

setExitChain

```
public void setExitChain(List collection);
```

Inserts a collection of receive exits into the chain.

Parameters

- collection - a List object which defines the receive exits which are to be chained.

MQSendExitChain

public class **MQSendExitChain**
extends MQExitChain
implements MQSendExit

```
java.lang.Object
   |
   +----com.ibm.mq.MQExitChain
           |
           +----com.ibm.mq.MQSendExitChain
```

Chains send exits together. The exits are of class MQSendExit; as well as exits written in Java, this includes non-Java send exits made available by means of the MQExternalSendExit class.

Constructors

MQSendExitChain

```
public MQSendExitChain();
```

The default constructor. Creates a Send Exit Chain. Use setExitChain() to add an exit chain.

MQSendExitChain

```
public MQSendExitChain(List collection);
```

Constructor which defines the send exits which are to be chained.

Parameters

- collection - defines the send exits which are to be chained.

Methods

sendExit

```
public byte[] sendExit(MQChannelExit channelExitParms,
                       MQChannelDefinition channelDefinition,
                       byte[] agentBuffer);
```

Calls the send exit. This is normally made by the Java client code.

Parameters

- channelExitParms - the definition of the chain of exits.
- channelDefinition - the definition of the channel.
- agentBuffer - the message being passed into the chain of exits.

Returns

- agentBuffer the data to be processed. If the exit response code (in channelExitParms) is MQXCC_OK the WebSphere MQ Client for Java can process the data. The simplest sendExit method therefore, consists of a single line.

setExitChain

```
public void setExitChain(List collection);
```

Inserts a collection of send exits into the chain.

Parameters

- collection - defines the send exits which are to be chained.

MQSimpleConnectionManager

public final class **MQSimpleConnectionManager**
extends Object
implements MQConnectionManager

```
java.lang.Object
   |
   +----com.ibm.mq.MQSimpleConnectionManager
```

An MQSimpleConnectionManager provides basic connection pooling function. You can use an MQSimpleConnectionManager either as the default connection manager, or as a parameter to an MQQueueManager constructor. When an MQQueueManager is constructed, the most recently used connection in the pool is used. Connections are destroyed by a separate thread when they are unused for a specified period, when there are more than a specified number of unused connections in the pool, or when the maximum number of connections has been reached and room must be made for new connections. You can specify the timeout period, the maximum number of managed connections, and the maximum number of unused connections.

Constructors

MQSimpleConnectionManager

```
public MQSimpleConnectionManager();
```

Constructs an MQSimpleConnectionManager.

Fields

MODE_ACTIVE

```
public final static int
```

The pool is always active. On MQQueueManager.disconnect(), the underlying connection is pooled and can be reused the next time an MQQueueManager object is constructed. Connections are destroyed by a separate thread if they have been unused for longer then the timeout period or if the size of the pool exceeds the value set by setMaxUnusedConnections().

MODE_AUTO

```
public final static int
```

An MQSimpleConnectionManager is active if it is the default connection manager and there is at least one connection in the pool.

This is the default.

MODE_INACTIVE

```
public final static int
```

The pool is always inactive. The pool of connections is cleared on entering this mode. The connection underlying any active MQQueueManager objects is destroyed when MQQueueManager.disconnect() is called.

Methods

allocateConnection

```
public Object allocateConnection(MQManagedConnectionFactory mcf,
                                 ConnectionRequestInfo cxRequestInfo)
             throws ResourceException;
```

Makes a connection to a queue manager, either by reusing an existing connection or by creating a new one. It is called by the connection factory instance of the resource adapter.

Normal applications should not call this method.

Parameters

- mcf - the connection factory.
- cxRequestInfo - represents information specific to the resource adapter for handling the connection request.

Returns

- a connection.

Exceptions

- ResourceException - if the call fails.

createConnection

```
public Object createConnection(MQManagedConnectionFactory mcf,
                               ConnectionRequestInfo cxRequestInfo)
             throws ResourceException;
```

Makes a connection to a queue manager. It is called by the connection factory instance of the resource adapter.

Normal applications should not call this method.

Parameters

- mcf - the connection factory.
- cxRequestInfo - represents information specific to the resource adapter for handling the connection request.

Returns

- a connection.

Exceptions

- ResourceException - if the call fails.

getActive

```
public int getActive();
```

Gets the active mode of the pool.

Returns

- one of:
  - MODE_AUTO
  - MODE_ACTIVE
  - MODE_INACTIVE

getHighThreshold

```
public int getHighThreshold();
```

getMaxConnections

```
public int getMaxConnections();
```

Gets the maximum number of connections.

Returns

- the maximum number of connections.

getMaxUnusedConnections

```
public int getMaxUnusedConnections();
```

Gets the maximum number of unused connections in the pool.

Returns

- the maximum number of unused connections.

getTimeout

```
public long getTimeout();
```

Gets the timeout value.

Returns

- the time out value in milliseconds. Connections which have been unused for this length of time are destroyed.

recycleConnection

```
public Object recycleConnection(MQManagedConnectionFactory mcf,
                                ConnectionRequestInfo cxRequestInfo);
```

Finds an existing connection to a queue manager. It is called by the connection factory instance of the resource adapter.

Normal applications should not call this method.

Parameters

- mcf - the connection factory.
- cxRequestInfo - represents information specific to the resource adapter for handling the connection request.

Returns

- a connection, or null if the call fails.

setActive

```
public void setActive(int mode);
```

Sets the active mode of the pool.

Parameters

- mode - one of:
  - MODE_AUTO
  - MODE_ACTIVE
  - MODE_INACTIVE

setHighThreshold

```
public void setHighThreshold(int limit);
```

setMaxConnections

```
public void setMaxConnections(int newLimit)
             throws IllegalArgumentException;
```

Sets the maximum number of connections.

Parameters

- newLimit - the new maximum number of connections.

Exceptions

- IllegalArgumentException -

setMaxUnusedConnections

```
public void setMaxUnusedConnections(int limit);
```

Sets the maximum number of unused connections in the pool.

Parameters

- limit - recently used connections are destroyed if the size of the pool exceeds this value.

setTimeout

```
public void setTimeout(long timeout);
```

Sets the timeout value.

Parameters

- timeout - the time out value in milliseconds. Connections which have been unused for this length of time are destroyed.

MQC

public interface **MQC**

`com.ibm.mq.MQC`

The MQC interface defines all the constants used by the WebSphere MQ Java programming interface (except for completion code constants and error code constants). To refer to one of these constants from within your programs, simply prefix constant name with ″MQC.″. For example, you can set the close options for a queue as follows:

```
MQQueue queue;
...
queue.closeOptions = MQC.MQCO_DELETE;  // delete the queue when it is closed
...
```

Fields

ASSOCIATE_ALL

`public final static int`

This value can be defined in the MQEnvironment to indicate that the MQQueueManager object being created can be shared within the context of the Java Virtual Machine. A subsequent call to MQEnvironment.getQueueManagerReference(int) or MQEnvironment.getQueueManagerReference(int, Object), where the Object is a Java String containing the name of the WebSphere MQ queue manager, will return a reference to this MQQueueManager object.

ASSOCIATE_NONE

`public final static int`

This value can be defined in the MQEnvironment to indicate that the MQQueueManager object being created will not be available for sharing within any context. This is the default value.

ASSOCIATE_THREAD

`public final static int`

This value can be defined in the MQEnvironment to indicate that the MQQueueManager object being created can be shared within the context of the currently executing thread. A subsequent call to MQEnvironment.getQueueManagerReference(int) or MQEnvironment.getQueueManagerReference(int, Object), where the Object is a Java String containing the name of the WebSphere MQ queue manager, will return a reference to this MQQueueManager object.

CCSID_PROPERTY

`public final static java.lang.String`

WebSphere MQ Java environment key for defining the CCSID, the coded-character-set-ID to be used on connections. The corresponding value must be an Integer. This property overrides MQEnvironment.CCSID.

CHANNEL_PROPERTY

`public final static java.lang.String`

WebSphere MQ Java environment key for defining the channel name. The corresponding value must be a String. This property overrides MQEnvironment.channel .

CONNECT_OPTIONS_PROPERTY

```
public final static java.lang.String
```

WebSphere MQ Java environment key for defining the connect options. Permitted values are

- MQCNO_STANDARD_BINDING
- MQCNO_FASTPATH_BINDING
- MQCNO_NONE
- MQCNO_SHARED_BINDING
- MQCNO_ISOLATED_BINDING
- MQCNO_SERIALIZE_CONN_TAG_Q_MGR
- MQCNO_SERIALIZE_CONN_TAG_QSG
- MQCNO_RESTRICT_CONN_TAG_Q_MGR
- MQCNO_RESTRICT_CONN_TAG_QSG

The default value is MQC.MQCNO_NONE

CONNTAG_PROPERTY

```
public final static java.lang.String
```

WebSphere MQ Java environment key for defining the connection tag. This is a tag that the queue manager associates with the resources that are affected by the application during this connection. It is only used by z/OS. The length of the connection tag must be 128 bytes.

HDRCOMPLIST_LENGTH

```
public final static int
```

The maximum length of the list of header compression techniques which can be set.

HEADER_COMPRESSION_PROPERTY

```
public final static java.lang.String
```

WebSphere MQ Java environment key for specifying compression techniques (in order of preference) to be applied to message header data. The corresponding value must be of type java.util.Collection. The following compression technique options are valid:

MQCOMPRESS_NONE MQCOMPRESS_SYSTEM.

This property overrides MQEnvironment.hdrCompList .

HOST_NAME_PROPERTY

```
public final static java.lang.String
```

The WebSphere MQ Java environment key for defining the host name property. The corresponding value must be a String. This property overrides MQEnvironment.hostname.

LOCAL_ADDRESS_PROPERTY

```
public final static java.lang.String
```

WebSphere MQ Java environment key for defining firewall local address property. The corresponding value must be a String, in the format ″IP(Low port, High port)″, e.g. ″9.20.0.1(2000,3000)″. This defines a range of local ports to be selected when making a connection to an MQ queue manager.

MESSAGE_COMPRESSION_PROPERTY

```
public final static java.lang.String
```

WebSphere MQ Java environment key for specifying compression techniques (in order of preference) to be applied to message data. The corresponding value must be of type java.util.Collection. The following compression technique options are valid:

MQCOMPRESS_NONE MQCOMPRESS_RLE MQCOMPRESS_ZLIBFAST MQCOMPRESS_ZLIBHIGH.

This property overrides `MQEnvironment.msgCompList`.

MQ_ACCOUNTING_TOKEN_LENGTH

```
public final static int
```

This defines the length of the accounting token field. The length is 32 bytes.

MQ_APPL_IDENTITY_DATA_LENGTH

```
public final static int
```

This defines the length of the application identity field. The length is 32 characters.

MQ_APPL_NAME_LENGTH

```
public final static int
```

This defines the length of the application name. The length is 28 characters.

MQ_APPL_ORIGIN_DATA_LENGTH

```
public final static int
```

This defines length of the application origin data field. ApplOriginData is information that is defined by the application suite that can be used to provide additional information about the origin of the message. The length is 4 characters.

MQ_CHANNEL_NAME_LENGTH

```
public final static int
```

This defines the length of the channel name field. The length is 20 characters.

MQ_CONN_NAME_LENGTH

```
public final static int
```

This defines the length of the connection name field. The length is 264 characters.

MQ_CONN_TAG_LENGTH

```
public final static int
```

This defines length of the connection tag. This is a tag that the queue manager associates with the resources that are affected by the application during this connection. The length is 128 bytes.

MQ_CORREL_ID_LENGTH
```
public final static int
```

This defines the length of the correlation ID field. The length is 24 bytes.

MQ_EXIT_DATA_LENGTH
```
public final static int
```

This defines the length of the exit data. The length is 32 bytes.

MQ_EXIT_NAME_LENGTH
```
public final static int
```

This defines the length of the exit name. The length is variable.

MQ_EXIT_USER_AREA_LENGTH
```
public final static int
```

This defines the length of the exit user area. The length is 16 bytes.

MQ_FORMAT_LENGTH
```
public final static int
```

This defines length of the message format field. The length is 8 bytes.

MQ_GROUP_ID_LENGTH
```
public final static int
```

This defines the length of the Group ID field. The length is 24 bytes

MQ_MSG_HEADER_LENGTH
```
public final static int
```

This defines the length of the message header. The length is 4000 bytes.

MQ_MSG_ID_LENGTH
```
public final static int
```

This defines the length of the message ID field. The length is 24 bytes.

MQ_MSG_TOKEN_LENGTH
```
public final static int
```

This defines the length of the message token field. The length is 16 bytes.

MQ_NAMELIST_DESC_LENGTH
```
public final static int
```

This defines the length of the namelist description field. The length is 64 characters.

MQ_NAMELIST_NAME_LENGTH
```
public final static int
```

This defines the length of the name of the namelist. The length is 48 characters.

MQ_PASSWORD_LENGTH
```
public final static int
```

This defines the length of the password field. The length is 12 characters.

MQ_PROCESS_APPL_ID_LENGTH
```
public final static int
```

This defines the length of the process application ID field. The length is 256 bytes.

MQ_PROCESS_DESC_LENGTH
```
public final static int
```

This defines the length of the process description field. The length is 64 bytes.

MQ_PROCESS_ENV_DATA_LENGTH
```
public final static int
```

This defines the length of the environment data field. The length is 128 bytes.

MQ_PROCESS_NAME_LENGTH
```
public final static int
```

This defines the length of the process name field. The length is 48 bytes.

MQ_PROCESS_USER_DATA_LENGTH
```
public final static int
```

This defines the length of the process user data field. The length is 128 bytes.

MQ_PUT_APPL_NAME_LENGTH
```
public final static int
```

This defines the length of the MQ_PUT_APPL_NAME field. This contains the name of the application that put a message on the dead-letter (undelivered-message) queue. The length is 28 characters.

MQ_Q_DESC_LENGTH
```
public final static int
```

This defines the length of the queue description field. The length is 64 characters.

MQ_Q_MGR_DESC_LENGTH
```
public final static int
```

This defines the length of the queue manager description field. The length is 64 characters.

MQ_Q_MGR_NAME_LENGTH
```
public final static int
```

This defines the length of the queue manager name field. The length is 48 characters.

MQ_Q_NAME_LENGTH
```
public final static int
```

This defines the length of the queue name field. The length is 48 characters.

MQ_QMGR_ASSOCIATION_PROPERTY
```
public final static java.lang.String
```

WebSphere MQ Java environment key for defining whether an MQQueueManager object can be shared within a specified context. If this value is not set, then the default behavior is to prevent the object being shared.

MQ_QSG_NAME_LENGTH
```
public final static int
```

This defines the length of the queue sharing group name field. The length is 4 characters.

MQ_SECURITY_ID_LENGTH
```
public final static int
```

This defines the length of the security ID field. The length is 40 bytes.

MQ_STORAGE_CLASS_LENGTH
```
public final static int
```

This defines the length of the storage class name field. The length is 8 characters.

MQ_TRIGGER_DATA_LENGTH
```
public final static int
```

This defines the length of the trigger data field. The length is 64 bytes.

MQ_USER_ID_LENGTH
```
public final static int
```

This defines the length of the user ID field. The length is 12 bytes.

MQACT_NONE
```
public final static byte[]
```

No accounting token is specified. The value is binary zero for the length of the field.

MQACTT_CICS_LUOW_ID
```
public final static byte
```

This defines a CICS LUOW accounting token.

MQACTT_DOS_DEFAULT
```
public final static byte
```

This defines the default MS-DOS accounting token.

MQACTT_NT_SECURITY_ID
```
public final static byte
```

This defines the Windows security ID accounting token.

MQACTT_OS2_DEFAULT
```
public final static byte
```

This defines the default OS/2® accounting token.

MQACTT_OS400_ACCOUNT_TOKEN
```
public final static byte
```

This defines the default i5/OS accounting token.

MQACTT_UNIX_NUMERIC_ID
```
public final static byte
```

This defines the default UNIX numeric accounting token.

MQACTT_UNKNOWN
```
public final static byte
```

This defines an unknown accounting token type.

MQACTT_USER
```
public final static byte
```

A user-defined accounting token.

MQACTT_WINDOWS_DEFAULT
```
public final static byte
```

This defines the default Windows accounting token.

MQAT_AIX
```
public final static int
```

This value indicates that an AIX application put the message. This is the same value as MQAT_UNIX.

MQAT_CICS
```
public final static int
```

This value indicates that a CICS transaction put the message.

MQAT_CICS_BRIDGE
```
public final static int
```

This value indicates that the CICS bridge put the message.

MQAT_CICS_VSE
```
public final static int
```

This value indicates that a CICS/VSE® transaction put the message.

MQAT_DEFAULT
```
public final static int
```

This value indicates the default application type. This is the default application type for the platform on which the application is running.

MQAT_DOS
```
public final static int
```

This value indicates that a WebSphere MQ client application on PC DOS put the message.

MQAT_GUARDIAN
```
public final static int
```

This value indicates that a Tandem Guardian application put the message. This is the same value as MQAT_NSK.

MQAT_IMS
```
public final static int
```

This value indicates that an IMS™ application put the message.

MQAT_IMS_BRIDGE
```
public final static int
```

This value indicates that the IMS bridge put the message.

MQAT_JAVA
```
public final static int
```

This value indicates that a Java application put the message.

MQAT_MVS
```
public final static int
```

This value indicates that an MVS™ or TSO application put the message. This is the same value as MQAT_ZOS.

MQAT_NO_CONTEXT
```
public final static int
```

This value is set by the queue manager when a message is put with no context (that is, the MQPMO_NO_CONTEXT context option is specified).

MQAT_NOTES_AGENT
```
public final static int
```

This value indicates that a Lotus Notes® Agent application put the message.

MQAT_NSK
```
public final static int
```

This value indicates that a Compaq NonStop Kernel application put the message.

MQAT_OS2
```
public final static int
```

This value indicates that an OS/2 application put the message.

MQAT_OS400
```
public final static int
```

This value indicates that a i5/OS application put the message.

MQAT_QMGR
```
public final static int
```

This value indicates that a queue manager put the message.

MQAT_UNIX
```
public final static int
```

This value indicates that a UNIX application put the message. This is the same value as MQAT_AIX.

MQAT_UNKNOWN
```
public final static int
```

This value indicates that the type of application that put the message is unknown, even though other context information is present.

MQAT_USER_FIRST
```
public final static int
```

This defines the lowest value for user-defined application types.

MQAT_USER_LAST
```
public final static int
```

This defines the highest value for user-defined application types.

MQAT_VMS
```
public final static int
```

This value indicates that a Digital OpenVMS application put the message.

MQAT_VOS
```
public final static int
```

This value indicates that a Stratus VOS application put the message.

MQAT_WINDOWS
```
public final static int
```

This value indicates that a 16-bit Windows application put the message.

MQAT_WINDOWS_NT
```
public final static int
```

This value indicates that a 32-bit Windows application put the message.

MQAT_XCF

```
public final static int
```

This value indicates that XCF put the message.

MQBND_BIND_NOT_FIXED

```
public final static int
```

This value indicates that binding is not fixed by the MQOPEN call. This is the binding that is used when MQOO_BIND_AS_Q_DEF is specified on the MQOPEN call and the queue is a cluster queue.

MQBND_BIND_ON_OPEN

```
public final static int
```

This value indicates that binding is fixed by the MQOPEN call. This is the binding that is used when MQOO_BIND_AS_Q_DEF is specified on the MQOPEN call and the queue is a cluster queue.

MQCA_ALTERATION_DATE

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the date of most-recent alteration. The length of the string is MQ_DATE_LENGTH.

MQCA_ALTERATION_TIME

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the time of most-recent alteration. The length of the string is MQ_TIME_LENGTH.

MQCA_APPL_ID

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the application ID. This is a character string that identifies the application to be started. This information is for use by a trigger-monitor application that processes messages on the initiation queue; the information is sent to the initiation queue as part of the trigger message. The length of the string is MQ_PROCESS_APPL_ID_LENGTH.

MQCA_AUTH_INFO_CONN_NAME

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the connection name of an AuthInfo object.

MQCA_AUTH_INFO_DESC

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the description of an AuthInfo object.

MQCA_AUTH_INFO_NAME

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the name of an AuthInfo object.

MQCA_BACKOUT_REQ_Q_NAME

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the excessive backout requeue queue name. The length of the string is MQ_Q_NAME_LENGTH.

MQCA_BASE_Q_NAME

```
public final static int
```

This character attribute selector is used to determine the name of queue that alias resolves to. The length of string is MQ_Q_NAME_LENGTH.

MQCA_CF_STRUC_DESC

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the description of the coupling-facility structure where the messages on the queue are stored. The length of this attribute is given by MQ_CF_STRUC_NAME_LENGTH.

This attribute applies only to shared queues. This attribute is supported only on z/OS.

MQCA_CF_STRUC_NAME

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the name of the coupling-facility structure where the messages on the queue are stored. The length of this attribute is given by MQ_CF_STRUC_NAME_LENGTH.

This attribute applies only to shared queues. This attribute is supported only on z/OS.

MQCA_CHANNEL_AUTO_DEF_EXIT

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the name of the user exit for automatic channel definition. The length of the string is MQ_EXIT_NAME_LENGTH.

MQCA_CLUSTER_DATE

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the date when the cluster definition became available to the local queue manager. The format of the date is YYYY-MM-DD, padded with two trailing blanks to make the length 12 bytes. The length of the string is MQ_CREATION_DATE_LENGTH.

MQCA_CLUSTER_NAME

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the name of the cluster to which the queue belongs. The length of the string is MQ_CLUSTER_NAME_LENGTH.

MQCA_CLUSTER_NAMELIST

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the name of a namelist object that contains the names of clusters to which this queue belongs. The length of the string is MQ_NAMELIST_NAME_LENGTH.

MQCA_CLUSTER_Q_MGR_NAME

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the name of a cluster queue manager. The length of the string is MQ_Q_MGR_NAME_LENGTH.

MQCA_CLUSTER_TIME

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the time when the cluster definition became available to the local queue manager. The format of the time is HH.MM.SS using the 24-hour clock, with a leading zero if the hour is less than 10. The length of the string is MQ_CREATION_TIME_LENGTH.

MQCA_CLUSTER_WORKLOAD_DATA

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the user-defined 32-byte character string that is passed to the cluster workload exit when it is called. If there is no data to pass to the exit, the string is blank. The length of the string is MQ_EXIT_DATA_LENGTH.

MQCA_CLUSTER_WORKLOAD_EXIT

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the name of the user exit for cluster workload management. If this name is non-blank, the exit is called each time that a message is put to a cluster queue or moved from one cluster-sender queue to another. The length of the string is MQ_EXIT_NAME_LENGTH.

MQCA_COMMAND_INPUT_Q_NAME

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the name of the command input queue defined on the local queue manager. This is a queue to which users can send commands, if authorized to do so. The name of the queue depends on the environment. The length of the string is MQ_Q_NAME_LENGTH.

MQCA_COMMAND_REPLY_Q_NAME

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the name of the command reply queue defined on the local queue manager. The name of the queue depends on the environment. The length of the string is MQ_Q_NAME_LENGTH.

MQCA_CREATION_DATE

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the date when the queue was created. The format of the date is YYYY-MM-DD, padded with two trailing blanks to make the length 12 bytes. The length of the string is MQ_CREATION_DATE_LENGTH.

MQCA_CREATION_TIME

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the time when the queue was created. The format of the time is HH.MM.SS using the 24-hour clock, with a leading zero if the hour is less than 10. The length of the string is MQ_CREATION_TIME_LENGTH.

MQCA_DEAD_LETTER_Q_NAME

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the name of the queue defined on the local queue manager as the dead-letter (undelivered-message) queue. Messages are sent to this queue if they cannot be routed to their correct destination. The length of the string is MQ_Q_NAME_LENGTH.

MQCA_DEF_XMIT_Q_NAME

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the name of the transmission queue that is used for the transmission of messages to remote queue managers, if there is no other indication of which transmission queue to use. If there is no default transmission queue, the name is entirely blank. The initial value of this attribute is blank. The length of the string is MQ_Q_NAME_LENGTH.

MQCA_ENV_DATA

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the name of the character string that contains environment-related information pertaining to the application to be started. This information is for use by a trigger-monitor application that processes messages on the initiation queue; the information is sent to the initiation queue as part of the trigger message. The length of the string is MQ_PROCESS_ENV_DATA_LENGTH.

MQCA_FIRST

```
public final static int
```

This defines the start of the range of valid character attribute selectors. The integer and character attribute selectors are allocated within two different ranges, with MQCA_* selectors within the range MQCA_FIRST through MQCA_LAST.

MQCA_IGQ_USER_ID

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the user identifier that is associated with the local intra-group queuing agent (IGQ agent). This attribute is applicable only if the local queue manager is a member of a queue-sharing group. The length of the string is MQ_USER_ID_LENGTH.

MQCA_INITIATION_Q_NAME

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the name of the initiation queue defined on the local queue manager. The queue must be of type MQQT_LOCAL. The queue manager sends a trigger message to the initiation queue when application start-up is required as a result of a message arriving on the queue to which this attribute belongs. The length of the string is MQ_Q_NAME_LENGTH.

MQCA_LAST

```
public final static int
```

This defines the end of the range of valid character attribute selectors. The integer and character attribute selectors are allocated within two different ranges, with MQCA_* selectors within the range MQCA_FIRST through MQCA_LAST.

MQCA_LAST_USED

```
public final static int
```

This defines the highest value in the range of valid character attribute selectors that the queue manager will accept.

MQCA_LDAP_PASSWORD

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the password needed to access the defined LDAP CRL server.

MQCA_LDAP_USER_NAME

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the LDAP user name. It consists of the Distinguished Name of the user who is attempting to access the LDAP CRL server.

MQCA_NAMELIST_DESC

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the namelist description. The content of the field is of no significance to the queue manager. The length of the string is MQ_NAMELIST_DESC_LENGTH.

MQCA_NAMELIST_NAME

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the namelist name. For more information about namelist names, see the *WebSphere MQ Application Programming Guide*. The length of the string is MQ_NAMELIST_NAME_LENGTH.

MQCA_NAMES

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine a list of NameCount names, where each name is the name of an object that is defined to the local queue manager. For more information about object names, see *WebSphere MQ Application Programming Reference*. The length of the each of the names in the list is MQ_OBJECT_NAME_LENGTH.

MQCA_PROCESS_DESC

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the process description. The content of the field is of no significance to the queue manager. The length of the string is MQ_PROCESS_DESC_LENGTH.

MQCA_PROCESS_NAME

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the name of a process definition that is defined on the local queue manager. Each process definition has a name that is different from the names of other process definitions belonging to the queue manager. The length of the string is MQ_PROCESS_NAME_LENGTH.

MQCA_Q_DESC

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine a queue description. The content of the field is of no significance to the queue manager. The length of the string is MQ_Q_DESC_LENGTH.

MQCA_Q_MGR_DESC

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the queue manager description. The content of the field is of no significance to the queue manager. The length of the string is MQ_Q_MGR_DESC_LENGTH.

MQCA_Q_MGR_IDENTIFIER

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the internally-generated unique name for the queue manager. The length of the string is MQ_Q_MGR_IDENTIFIER_LENGTH.

MQCA_Q_MGR_NAME

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the name of the local queue manager. That is, the name of the queue manager to which the application is connected. The length of the string is MQ_Q_MGR_NAME_LENGTH.

MQCA_Q_NAME

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the name of a queue defined on the local queue manager. For more information about queue names, see *WebSphere MQ Application Programming Guide*. All queues defined on a queue manager share the same queue name space. Therefore, an MQQT_LOCAL queue and an MQQT_ALIAS queue cannot have the same name. The length of the string is MQ_Q_NAME_LENGTH.

MQCA_QSG_NAME

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the name of a queue sharing group to which the local queue manager belongs. If the local queue manager does not belong to a queue-sharing group, the name is blank. The length of the string is MQ_QSG_NAME_LENGTH.

MQCA_REMOTE_Q_MGR_NAME

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the name of the remote queue manager on which the queue RemoteQName is defined. If the RemoteQName queue has a QSGDisp value of MQQSGD_COPY or MQQSGD_SHARED, RemoteQMgrName can be the name of the queue-sharing group that owns RemoteQName. The length of the string is MQ_Q_MGR_NAME_LENGTH.

MQCA_REMOTE_Q_NAME

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the name of the queue as it is known on the remote queue manager RemoteQMgrName. The length of the string is MQ_Q_NAME_LENGTH.

MQCA_REPOSITORY_NAME

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the name of a cluster for which this queue manager provides a repository-manager service. The length of the string is MQ_Q_MGR_NAME_LENGTH.

MQCA_REPOSITORY_NAMELIST

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the name of a namelist object that contains the names of clusters for which this queue manager provides a repository-manager service. If the queue manager provides this service for only one cluster, the namelist object contains only one name. The length of the string is MQ_NAMELIST_NAME_LENGTH.

MQCA_SSL_CRL_NAMELIST

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the name of the namelist object containing names of authentication information objects.

MQCA_SSL_CRYPTO_HARDWARE

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the cryptographic hardware configuration string. This field is relevant only for WebSphere MQ clients running on UNIX systems.

MQCA_STORAGE_CLASS

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the user-defined name that defines the physical storage used to hold the queue. This attribute is supported only on z/OS. The length of the string is MQ_STORAGE_CLASS_LENGTH.

MQCA_STORAGE_CLASS_DESC

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the user-defined description of the physical storage used to hold the queue. The content of the field is of no significance to the queue manager. This attribute is supported only on z/OS. The length of the string is MQ_STORAGE_CLASS_LENGTH.

MQCA_TRIGGER_DATA

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the free-format data that the queue manager inserts into the trigger message when a message arriving on this queue causes a trigger message to be written to the initiation queue. The length of the string is MQ_TRIGGER_DATA_LENGTH.

MQCA_USER_DATA

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the string that contains user information pertaining to the application to be started. This information is for use by a trigger-monitor application that processes messages on the initiation queue, or the application that is started by the trigger monitor. The information is sent to the initiation queue as part of the trigger message. The length of the string is MQ_PROCESS_USER_DATA_LENGTH.

MQCA_XCF_GROUP_NAME

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the XCF group name. The maximum length of the string is MQ_XCF_GROUP_NAME_LENGTH.

MQCA_XCF_MEMBER_NAME

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the XCF member name. The maximum length of the string is MQ_XCF_MEMBER_NAME_LENGTH.

MQCA_XMIT_Q_NAME

```
public final static int
```

This character attribute selector is used with an MQINQ call to determine the transmission queue name. If this attribute is not blank when an open occurs, either for a remote queue or for a queue-manager alias definition, it specifies the name of the local transmission queue to be used for forwarding the message. If XmitQName is blank, the local queue whose name is the same as RemoteQMgrName is used as the transmission queue. If there is no queue with the name RemoteQMgrName, the queue identified by the DefXmitQName queue-manager attribute is used. The length of the string is MQ_Q_NAME_LENGTH.

MQCCSI_DEFAULT

```
public final static int
```

The CodedCharSetId of the data in the String field is defined by the CodedCharSetId field in the header structure that precedes the MQCFH structure, or by the CodedCharSetId field in the MQMD if the MQCFH is at the start of the message.

MQCCSI_INHERIT

```
public final static int
```

Character data in the message is in the same character set as this structure. This is the queue manager's character set. (For MQMD only, MQCCSI_INHERIT has the same meaning as MQCCSI_Q_MGR).

The queue manager changes this value in the MQMD that is sent with the message to the actual character-set identifier of MQMD. Provided no error occurs, the value MQCCSI_INHERIT is not returned by the MQGET call.

MQCCSI_Q_MGR

```
public final static int
```

Character data in the message is in the queue manager's character set.

On the MQPUT and MQPUT1 calls, the queue manager changes this value in the MQMD that is sent with the message to the true character-set identifier of the queue manager. As a result, the value MQCCSI_Q_MGR is never returned by the MQGET call.

MQCF_DIST_LISTS

```
public final static int
```

This flag indicates that distribution lists are supported by the local queue manager.

MQCI_NEW_SESSION

```
public final static byte[]
```

This indicates that the Message is the start of a new session. This value is recognized by the CICS bridge as indicating the start of a new session, that is, the start of a new sequence of messages.

MQCI_NONE

```
public final static byte[]
```

No correlation ID is specified. The value is binary zero for the length of the field.

MQCMDL_LEVEL_1
```
public final static int
```

This indicates that level 1 of system control commands are supported by the queue manager.

MQCMDL_LEVEL_101
```
public final static int
```

This indicates that level 1.01 of system control commands are supported by the queue manager.

MQCMDL_LEVEL_110
```
public final static int
```

This indicates that level 1.10 of system control commands are supported by the queue manager.

MQCMDL_LEVEL_114
```
public final static int
```

This indicates that level 1.14 of system control commands are supported by the queue manager.

MQCMDL_LEVEL_120
```
public final static int
```

This indicates that level 1.20 of system control commands are supported by the queue manager.

MQCMDL_LEVEL_200
```
public final static int
```

This indicates that level 2.00 of system control commands are supported by the queue manager.

MQCMDL_LEVEL_201
```
public final static int
```

This indicates that level 2.01 of system control commands are supported by the queue manager.

MQCMDL_LEVEL_210
```
public final static int
```

This indicates that level 2.10 of system control commands are supported by the queue manager.

MQCMDL_LEVEL_221
```
public final static int
```

This indicates that level 2.21 of system control commands are supported by the queue manager.

MQCMDL_LEVEL_230
```
public final static int
```

This indicates that level 2.30 of system control commands are supported by the queue manager.

MQCMDL_LEVEL_320

```
public final static int
```

This indicates that level 3.20 of system control commands are supported by the queue manager.

MQCMDL_LEVEL_500

```
public final static int
```

This indicates that level 5.00 of system control commands are supported by the queue manager.

MQCMDL_LEVEL_510

```
public final static int
```

This indicates that level 5.10 of system control commands are supported by the queue manager.

MQCMDL_LEVEL_520

```
public final static int
```

This indicates that level 5.20 of system control commands are supported by the queue manager.

MQCMDL_LEVEL_530

```
public final static int
```

This indicates that level 5.30 of system control commands are supported by the queue manager.

MQCMDL_LEVEL_531

```
public final static int
```

This indicates that level 5.31 of system control commands are supported by the queue manager.

MQCMDL_LEVEL_600

```
public final static int
```

This indicates that level 6.00 of system control commands are supported by the queue manager.

MQCNO_FASTPATH_BINDING

```
public final static int
```

This option causes the application and the local-queue-manager agent to be part of the same unit of execution. This is in contrast to the normal method of binding, where the application and the local-queue-manager agent run in separate units of execution.

MQCNO_ISOLATED_BINDING

```
public final static int
```

This option causes the application and the local-queue-manager agent (the component that manages queuing operations) to run in separate units of execution (generally, in separate processes). This arrangement maintains the integrity of the queue manager, that is, it protects the queue manager from errant programs. The application process and the local-queue-manager agent are isolated from each other in that they do not share resources.

MQCNO_NONE

```
public final static int
```

This field can be specified to aid program documentation when no MQCNO_* options need be specified. It is not intended that this option be used with any other MQCNO_* option, but as its value is zero, such use cannot be detected.

MQCNO_RESTRICT_CONN_TAG_Q_MGR

```
public final static int
```

This option indicates that connection tag use is restricted within the queue manager. This connection option is supported on z/OS only. It requests shared use of the connection tag within the local queue manager. If the connection tag is already in use in the local queue manager, the MQCONNX call can succeed provided that the requesting application is running in the same processing scope as the existing user of the tag. If this condition is not satisfied, the MQCONNX call fails with reason code MQRC_CONN_TAG_IN_USE. The outcome of the call is not affected by use of the connection tag elsewhere in the queue-sharing group to which the local queue manager belongs.

On z/OS, applications must run within the same MVS address space in order to share the connection tag.

If the application using the connection tag is a client application, MQCNO_RESTRICT_CONN_TAG_Q_MGR is not allowed.

MQCNO_RESTRICT_CONN_TAG_QSG

```
public final static int
```

This option indicates that connection tag use is restricted within the queue-sharing group. This connection option is supported on z/OS only. It requests shared use of the connection tag within the queue-sharing group to which the local queue manager belongs. If the connection tag is already in use in the queue-sharing group, the MQCONNX call can succeed provided that: the requesting application is running in the same processing scope as the existing user of the tag; the requesting application is connected to the same queue manager as the existing user of the tag. If these conditions are not satisfied, the MQCONNX call fails with reason code MQRC_CONN_TAG_IN_USE.

On z/OS, applications must run within the same MVS address space in order to share the connection tag.

If the application using the connection tag is a client application, MQCNO_RESTRICT_CONN_TAG_Q_QSG is not allowed.

MQCNO_SERIALIZE_CONN_TAG_Q_MGR

```
public final static int
```

This option indicates that connection tag use is serialized within the queue manager. This connection option is supported on z/OS only. It requests exclusive use of the connection tag within the local queue manager. If the connection tag is already in use in the local queue manager, the MQCONNX call fails with reason code MQRC_CONN_TAG_IN_USE. The outcome of the call is not affected by use of the connection tag elsewhere in the queue-sharing group to which the local queue manager belongs.

MQCNO_SERIALIZE_CONN_TAG_QSG

```
public final static int
```

This option indicates that connection tag use is serialized within the queue-sharing group. This connection option is supported on z/OS only. It requests exclusive use of the connection tag within the queue-sharing group to which the local queue manager belongs. If the connection tag is already in use in the queue-sharing group, the MQCONNX call fails with reason code MQRC_CONN_TAG_IN_USE.

MQCNO_SHARED_BINDING

```
public final static int
```

This connection option causes the application and the local-queue-manager agent (the component that manages queuing operations) to run in separate units of execution (generally, in separate processes). This arrangement maintains the integrity of the queue manager. That is, it protects the queue manager from errant programs. However some resources are shared between the application and the local-queue-manager agent.

MQCNO_STANDARD_BINDING

```
public final static int
```

This connection option causes the application and the local-queue-manager agent (the component that manages queuing operations) to run in separate units of execution (generally, in separate processes). This arrangement maintains the integrity of the queue manager, that is, it protects the queue manager from errant programs.

MQCNO_VERSION_1

```
public final static int
```

This defines a version 1 connection options structure.

MQCNO_VERSION_2

```
public final static int
```

This defines a version 2 connection options structure.

MQCNO_VERSION_3

```
public final static int
```

This defines a version 3 connection options structure.

MQCNO_VERSION_4

```
public final static int
```

This defines a version 4 connection options structure.

MQCNO_VERSION_5

```
public final static int
```

This defines a version 5 connection options structure.

MQCO_DELETE

```
public final static int
```

The queue is deleted if either of the following is true:

- It is a permanent dynamic queue, and there are no messages on the queue and no uncommitted get or put requests outstanding for the queue (either for the current task or any other task).
- It is the temporary dynamic queue that was created by the MQOPEN call that returned Hobj. In this case, all the messages on the queue are purged.

In all other cases the call fails with reason code MQRC_OPTION_NOT_VALID_FOR_TYPE, and the object is not deleted.

MQCO_DELETE_PURGE

```
public final static int
```

The queue is deleted, and any messages on it purged, if either of the following is true:

- It is a permanent dynamic queue and there are no uncommitted get or put requests outstanding for the queue (either for the current task or any other task).
- It is the temporary dynamic queue that was created by the MQOPEN call that returned Hobj.

In all other cases the call fails with reason code MQRC_OPTION_NOT_VALID_FOR_TYPE, and the object is not deleted.

MQCO_NONE

```
public final static int
```

No optional close processing is required. This must be specified for:

- Objects other than queues
- Predefined queues
- Temporary dynamic queues (but only in those cases where Hobj is not the handle returned by the MQOPEN call that created the queue).
- Distribution lists

MQCOMPRESS_NONE

```
public final static int
```

Setting this value specifies that no message or header data compression is to take place. See MESSAGE_COMPRESSION_PROPERTY and HEADER_COMPRESSION_PROPERTY.

MQCOMPRESS_RLE

```
public final static int
```

Setting this value specifies that message data compression is to be performed using run-length encoding (RLE) compression. See MESSAGE_COMPRESSION_PROPERTY .

MQCOMPRESS_SYSTEM

```
public final static int
```

Setting this value specifies that header data compression is performed using run-length encoding (RLE) compression. See HEADER_COMPRESSION_PROPERTY .

MQCOMPRESS_ZLIBFAST

```
public final static int
```

Setting this value specifies that message data compression is performed using ZLIB encoding and with speed of compression prioritized over degree of compression. See MESSAGE_COMPRESSION_PROPERTY.

MQCOMPRESS_ZLIBHIGH

```
public final static int
```

Setting this value specifies that message data compression is performed using ZLIB encoding and with degree of compression prioritized over speed of compression. See MESSAGE_COMPRESSION_PROPERTY.

MQCSP_AUTH_NONE

```
public final static int
```

This value indicates that MQCSP user ID and password fields are not used by the Object Authority Manager (OAM) to perform authentication on a MQCONNX call. This is the default value.

MQCSP_AUTH_USER_ID_AND_PWD

```
public final static int
```

This value indicates that MQCSP user ID and password fields will be used by the Object Authority Manager (OAM) to perform authentication on a MQCONNX call. When this is specified, the MQCSP structure is passed to the OAM Authenticate User function, which can set appropriate identity context fields.

MQCSP_VERSION_1

```
public final static int
```

This defines a version 1 connection security parameters structure.

MQCT_NONE

```
public final static byte[]
```

MQCT_NONE can be used when no connection tag is required. The value is binary zero for the length of the field.

The connection tag field is only used when connecting to a z/OS queue manager. In other environments, specify the value MQCT_NONE.

MQDL_NOT_SUPPORTED

```
public final static int
```

Distribution-list messages cannot be stored on the queue, because the partnering queue manager does not support distribution lists. If an application puts a distribution-list message, and that message is to be placed on this queue, the queue

manager splits the distribution-list message and places the individual messages on the queue instead. This increases the amount of processing required to send the message to multiple destinations, but ensures that the messages are processed correctly by the partnering queue manager.

MQDL_SUPPORTED

```
public final static int
```

Distribution-list messages can be stored on the queue, and transmitted to the partnering queue manager in that form. This reduces the amount of processing required to send the message to multiple destinations.

MQEI_UNLIMITED

```
public final static int
```

This field marks a message as having an unlimited expiration time.

MQENC_DECIMAL_MASK

```
public final static int
```

Mask for packed-decimal-integer encoding. This subfield occupies bit positions 24 through 27 within the Encoding field.

MQENC_DECIMAL_NORMAL

```
public final static int
```

Packed-decimal integers are represented in the conventional way:

- Each decimal digit in the printable form of the number is represented in packed decimal by a single hexadecimal digit in the range X'0' through X'9'. Each hexadecimal digit occupies four bits, and so each byte in the packed decimal number represents two decimal digits in the printable form of the number.
- The least significant byte in the packed-decimal number is the byte that contains the least significant decimal digit. Within that byte, the most significant four bits contain the least significant decimal digit, and the least significant four bits contain the sign. The sign is either X'C' (positive), X'D' (negative), or X'F' (unsigned).
- The least significant byte in the number has the highest address of any of the bytes in the number; the most significant byte has the lowest address.
- The least significant bit in each byte is adjacent to the byte with the next higher address; the most significant bit in each byte is adjacent to the byte with the next lower address.

MQENC_DECIMAL_REVERSED

```
public final static int
```

Packed-decimal integers are represented in the same way as MQENC_DECIMAL_NORMAL, but with the bytes arranged in reverse order. The bits within each byte are arranged in the same way as MQENC_DECIMAL_NORMAL.

MQENC_DECIMAL_UNDEFINED

```
public final static int
```

Packed-decimal integers are represented using an encoding that is undefined.

MQENC_FLOAT_IEEE_NORMAL

```
public final static int
```

Floating-point numbers are represented using the standard IEEE3 floating-point format

MQENC_FLOAT_IEEE_REVERSED

```
public final static int
```

Floating-point numbers are represented in the same way as MQENC_FLOAT_IEEE_NORMAL, but with the bytes arranged in reverse order. The bits within each byte are arranged in the same way as MQENC_FLOAT_IEEE_NORMAL.

MQENC_FLOAT_MASK

```
public final static int
```

Mask for floating-point encoding. This subfield occupies bit positions 20 through 23 within the Encoding field.

MQENC_FLOAT_S390

```
public final static int
```

Floating-point numbers are represented using the standard zSeries (System/390®) floating-point format. This is also used by System/370™.

MQENC_FLOAT_TNS

```
public final static int
```

Floating-point numbers are represented using TNSFloat floating-point format. This is for use on Compaq NonStop Kernel applications.

MQENC_FLOAT_UNDEFINED

```
public final static int
```

Floating-point numbers are represented using an encoding that is undefined.

MQENC_INTEGER_MASK

```
public final static int
```

Mask for binary-integer encoding. This subfield occupies bit positions 28 through 31 within the Encoding field.

MQENC_INTEGER_NORMAL

```
public final static int
```

Binary integers are represented in the conventional way:

- The least significant byte in the number has the highest address of any of the bytes in the number; the most significant byte has the lowest address.
- The least significant bit in each byte is adjacent to the byte with the next higher address; the most significant bit in each byte is adjacent to the byte with the next lower address.

MQENC_INTEGER_REVERSED

```
public final static int
```

Binary integers are represented in the same way as MQENC_INTEGER_NORMAL, but with the bytes arranged in reverse order. The bits within each byte are arranged in the same way as MQENC_INTEGER_NORMAL.

MQENC_INTEGER_UNDEFINED
```
public final static int
```

Binary integers are represented using an encoding that is undefined.

MQENC_NATIVE
```
public final static int
```

Numbers are encoded using the system encoding. This is the default value

MQENC_RESERVED_MASK
```
public final static int
```

Mask for reserved bits. This subfield occupies bit positions 0 through 19 within the Encoding field.

MQFB_ACTIVITY
```
public final static int
```

Feedback indicating that an activity was performed on behalf of message.

MQFB_APPL_CANNOT_BE_STARTED
```
public final static int
```

Feedback indicating that an application processing a trigger message cannot start the application named in the ApplId field of the trigger message.

MQFB_APPL_FIRST
```
public final static int
```

This defines the lowest value for application-defined feedback.

MQFB_APPL_LAST
```
public final static int
```

This defines the highest value for application-defined feedback.

MQFB_APPL_TYPE_ERROR
```
public final static int
```

Feedback indicating that an application processing a trigger message cannot start the application because the ApplType field of the trigger message is not valid

MQFB_BUFFER_OVERFLOW
```
public final static int
```

The feedback codes can be generated by the IMS bridge to indicate that the value of one of the length fields would cause the data to overflow the message buffer.

MQFB_COA
```
public final static int
```

Feedback confirming arrival on the destination queue (see MQRO_COA).

MQFB_COD
```
public final static int
```

Feedback confirming delivery to the receiving application (see MQRO_COD).

MQFB_DATA_LENGTH_NEGATIVE
```
public final static int
```

The feedback codes can be generated by the IMS bridge to indicate that a segment length was negative in the application data of the message.

MQFB_DATA_LENGTH_TOO_BIG
```
public final static int
```

The feedback codes can be generated by the IMS bridge to indicate that a segment length too big in the application data of the message.

MQFB_DATA_LENGTH_ZERO
```
public final static int
```

The feedback codes can be generated by the IMS bridge to indicate that a segment length was zero in the application data of the message.

MQFB_EXPIRATION
```
public final static int
```

Feedback indicating that the message was discarded because it had not been removed from the destination queue before its expiry time had elapsed.

MQFB_IIH_ERROR
```
public final static int
```

The feedback codes can be generated by the IMS bridge to indicate that the Format field in MQMD specifies MQFMT_IMS, but the message does not begin with a valid MQIIH structure.

MQFB_LENGTH_OFF_BY_ONE
```
public final static int
```

The feedback codes can be generated by the IMS bridge to indicate that the value of one of the length fields was one byte too short.

MQFB_MAX_ACTIVITIES
```
public final static int
```

Feedback indicating that a trace-route message was discarded because it was involved in more than the specified maximum number of activities.

MQFB_NAN
```
public final static int
```

This is used with a message of type MQMT_REPORT to indicate the nature of the report, and is only meaningful with that type of message. This value indicates a negative action notification.

MQFB_NONE
```
public final static int
```

This is used with a message of type report, and indicates no feedback is provided.

MQFB_NOT_DELIVERED
```
public final static int
```

Feedback indicating that a trace-route message was discarded because it was about to be delivered to a local queue.

MQFB_NOT_FORWARDED
```
public final static int
```

Feedback indicating that a trace-route message was discarded because it was about to be forwarded to a queue manager that is unable to honor the value of the specified forwarding options.

MQFB_PAN
```
public final static int
```

This is used with a message of type MQMT_REPORT to indicate the nature of the report, and is only meaningful with that type of message. This value indicates a positive action notification.

MQFB_QUIT
```
public final static int
```

Feedback indicating an application ended. This can be used by a workload scheduling program to control the number of instances of an application program that are running. Sending an MQMT_REPORT message with this feedback code to an instance of the application program indicates to that instance that it should stop processing.

Adherence to this convention is a matter for the application; it is not enforced by the queue manager.

MQFB_STOPPED_BY_MSG_EXIT
```
public final static int
```

Feedback indicating that a message was stopped by a channel message exit.

MQFB_SYSTEM_FIRST
```
public final static int
```

This defines the lowest value for system-generated feedback.

MQFB_SYSTEM_LAST
```
public final static int
```

This defines the highest value for system-generated feedback.

MQFB_TM_ERROR
```
public final static int
```

Feedback indicating that the Format field in MQMD specifies MQFMT_TRIGGER, but the message does not begin with a valid MQTM structure.

MQFB_UNSUPPORTED_DELIVERY

```
public final static int
```

Feedback indicating that a trace-route message was discarded because at least one of the delivery options was not recognized and was in the MQROUTE_DELIVER_REJ_UNSUP_MASK bitmask.

MQFB_UNSUPPORTED_FORWARDING

```
public final static int
```

Feedback indicating that a trace-route message was discarded because at least one of the forwarding options was not recognized and was in the MQROUTE_FORWARD_REJ_UNSUP_MASK bitmask.

MQFB_XMIT_Q_MSG_ERROR

```
public final static int
```

Feedback indicating that a message channel agent has found that a message on the transmission queue is not in the correct format. The message channel agent puts the message on the dead-letter queue using this feedback code.

MQFMT_ADMIN

```
public final static java.lang.String
```

The message is a command-server request or reply message in programmable command format (PCF). Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

MQFMT_CICS

```
public final static java.lang.String
```

The message data begins with the CICS information header MQCIH, followed by the application data. The format name of the application data is given by the Format field in the MQCIH structure. On z/OS, specify the MQGMO_CONVERT option on the MQGET call to convert messages that have format MQFMT_CICS.

MQFMT_COMMAND_1

```
public final static java.lang.String
```

The message is an MQSC command-server reply message containing the object count, completion code, and reason code. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

MQFMT_COMMAND_2

```
public final static java.lang.String
```

The message is an MQSC command-server reply message containing information about the objects requested. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

MQFMT_DEAD_LETTER_HEADER

```
public final static java.lang.String
```

The message data begins with the dead-letter header MQDLH. The data from the original message immediately follows the MQDLH structure. The format name of the original message data is given by the Format field in the MQDLH structure. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

MQFMT_DIST_HEADER

```
public final static java.lang.String
```

The message data begins with the distribution-list header MQDH; this includes the arrays of MQOR and MQPMR records. The distribution-list header can be followed by additional data. The format of the additional data (if any) is given by the Format field in the MQDH structure.

MQFMT_EVENT

```
public final static java.lang.String
```

The message is an MQ event message that reports an event that occurred. Event messages have the same structure as programmable commands.

Version 1 event messages can be converted in all environments if the MQGMO_CONVERT option is specified on the MQGET call. Version 2 event messages can be converted only on z/OS.

MQFMT_IMS

```
public final static java.lang.String
```

The message data begins with the IMS information header MQIIH, which is followed by the application data. The format name of the application data is given by the Format field in the MQIIH structure. Specify the MQGMO_CONVERT option on the MQGET call to convert messages that have format MQFMT_IMS.

MQFMT_IMS_VAR_STRING

```
public final static java.lang.String
```

The message is an IMS variable string, which is a string of the form *llzzccc*.

MQFMT_MD_EXTENSION

```
public final static java.lang.String
```

The message data begins with the message-descriptor extension MQMDE, and is optionally followed by other data (usually the application message data). The format name, character set, and encoding of the data that follow the MQMDE are given by the Format, CodedCharSetId, and Encoding fields in the MQMDE.

MQFMT_NONE

```
public final static java.lang.String
```

The nature of the data is undefined, and the data cannot be converted when the message is retrieved from a queue using the MQGMO_CONVERT option.

MQFMT_PCF

```
public final static java.lang.String
```

The message is a user-defined message that conforms to the structure of a programmable command format (PCF) message. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

MQFMT_REF_MSG_HEADER

```
public final static java.lang.String
```

The message data begins with the reference message header MQRMH, and is optionally followed by other data. The format name, character set, and encoding of the data is given by the Format, CodedCharSetId, and Encoding fields in the MQRMH.

MQFMT_RF_HEADER_1

```
public final static java.lang.String
```

The message data begins with the rules and formatting header MQRFH, and is optionally followed by other data. The format name, character set, and encoding of the data (if any) are given by the Format, CodedCharSetId, and Encoding fields in the MQRFH. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

MQFMT_RF_HEADER_2

```
public final static java.lang.String
```

The message data begins with the version 2 rules and formatting header MQRFH2, and is optionally followed by other data. The format name, character set, and encoding of the optional data (if any) are given by the Format, CodedCharSetId, and Encoding fields in the MQRFH2. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

MQFMT_STRING

```
public final static java.lang.String
```

The application message data can be either an SBCS string (single-byte character set), or a DBCS string (double-byte character set). Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

MQFMT_TRIGGER

```
public final static java.lang.String
```

The message is a trigger message. Messages of this format can be converted if the MQGMO_CONVERT option is specified on the MQGET call.

MQFMT_XMIT_Q_HEADER

```
public final static java.lang.String
```

The message data begins with the transmission queue header MQXQH. The data from the original message immediately follows the MQXQH structure. The format name of the original message data is given by the Format field in the MQMD structure, which is part of the transmission queue header MQXQH.

MQGI_NONE

```
public final static byte[]
```

No group identifier is specified. The value is binary zero for the length of the field. This is the value that is used for messages that are not in groups, that are not segments of logical messages, and for which segmentation is not allowed.

MQGMO_ACCEPT_TRUNCATED_MSG

```
public final static int
```

If the message buffer is too small to hold the complete message, allow the MQGET call to fill the buffer with as much of the message as the buffer can hold.

MQGMO_ALL_MSGS_AVAILABLE

```
public final static int
```

Messages in a group become available for retrieval only when all messages in the group are available. If the queue contains message groups with some of the messages missing (perhaps because they have been delayed in the network and have not yet arrived), specifying MQGMO_ALL_MSGS_AVAILABLE prevents retrieval of messages belonging to incomplete groups. However, those messages still contribute to the value of the CurrentQDepth queue attribute; this means that there might be no retrievable message groups, even though CurrentQDepth is greater than zero.

MQGMO_ALL_SEGMENTS_AVAILABLE

```
public final static int
```

Segments in a logical message become available for retrieval only when all segments in the logical message are available. If the queue contains segmented messages with some of the segments missing (perhaps because they have been delayed in the network and have not yet arrived), specifying MQGMO_ALL_SEGMENTS_AVAILABLE prevents retrieval of segments belonging to incomplete logical messages. However, those segments still contribute to the value of the CurrentQDepth queue attribute; this means that there might be no retrievable logical messages, even though CurrentQDepth is greater than zero.

MQGMO_BROWSE_FIRST

```
public final static int
```

When a queue is opened with the MQOO_BROWSE option, a browse cursor is established, positioned logically before the first message on the queue. You can then use MQGET calls specifying the MQGMO_BROWSE_FIRST, MQGMO_BROWSE_NEXT, or MQGMO_BROWSE_MSG_UNDER_CURSOR option to retrieve messages from the queue nondestructively. The browse cursor marks the position, within the messages on the queue, from which the next MQGET call with MQGMO_BROWSE_NEXT searches for a suitable message.

MQGMO_BROWSE_MSG_UNDER_CURSOR

```
public final static int
```

Retrieve the message pointed to by the browse cursor nondestructively, regardless of the MQMO_* options specified in the MatchOptions field in MQGMO.

The message pointed to by the browse cursor is the one that was last retrieved using either the MQGMO_BROWSE_FIRST or the MQGMO_BROWSE_NEXT option. The call fails if neither of these calls have been issued for this queue since it was opened, or if the message that was under the browse cursor has since been retrieved destructively.

The position of the browse cursor is not changed by this call.

MQGMO_BROWSE_NEXT

```
public final static int
```

Advance the browse cursor to the next message on the queue that satisfies the selection criteria specified on the MQGET call. The message is returned to the application, but remains on the queue. After a queue has been opened for browse, the first browse call using the handle has the same effect whether it specifies the MQGMO_BROWSE_FIRST or MQGMO_BROWSE_NEXT option.

See *WebSphere MQ Application Programming Reference* for more information on this parameter.

MQGMO_COMPLETE_MSG

```
public final static int
```

Only a complete logical message can be returned by the MQGET call. If the logical message is segmented, the queue manager reassembles the segments and returns the complete logical message to the application; the fact that the logical message was segmented is not apparent to the application retrieving it.

MQGMO_CONVERT

```
public final static int
```

Requests the application data to be converted. The conversion conforms to the characterSet and `encoding` attributes of MQMessage, before the data is copied into the message buffer.

MQGMO_FAIL_IF_QUIESCING

```
public final static int
```

Force the MQGET call to fail if the queue manager is in the quiescing state. On z/OS, this option also forces the MQGET call to fail if the connection (for a CICS or IMS application) is in the quiescing state.

MQGMO_LOCK

```
public final static int
```

Lock the message that is browsed, so that the message becomes invisible to any other handle open for the queue. The option can be specified only if one of the following options is also specified:

- MQGMO_BROWSE_FIRST
- MQGMO_BROWSE_NEXT
- MQGMO_BROWSE_MSG_UNDER_CURSOR

MQGMO_LOGICAL_ORDER

```
public final static int
```

This option controls the order in which messages are returned by successive MQGET calls for the queue handle. The option must be specified on each of those calls in order to have an effect.

MQGMO_MARK_SKIP_BACKOUT

```
public final static int
```

Back out a unit of work without reinstating on the queue the message that was marked with this option.

When an application requests the backout of a unit of work containing a get request, a message that was retrieved using this option is not restored to its

previous state. (Other resource updates, however, are still backed out.) Instead, the message is treated as if it had been retrieved by a get request without this option, in a new unit of work started by the backout request.

MQGMO_MSG_UNDER_CURSOR

```
public final static int
```

Retrieve the message pointed to by the browse cursor, regardless of the MQMO_* options specified in the MatchOptions field in MQGMO. The message is removed from the queue. The message pointed to by the browse cursor is the one that was last retrieved using either the MQGMO_BROWSE_FIRST or the MQGMO_BROWSE_NEXT option.

MQGMO_NO_SYNCPOINT

```
public final static int
```

The request is to operate outside the normal unit-of-work protocols. The message is deleted from the queue immediately (unless this is a browse request). The message cannot be made available again by backing out the unit of work.

MQGMO_NO_WAIT

```
public final static int
```

The application does not wait if no suitable message is available. This is the opposite of the MQGMO_WAIT option, and is defined to aid program documentation. It is the default if neither is specified.

MQGMO_NONE

```
public final static int
```

This value indicates that no other options have been specified and all options assume their default values. MQGMO_NONE aids program documentation; it is not intended that this option be used with any other, but as its value is zero, such use cannot be detected.

MQGMO_SYNCPOINT

```
public final static int
```

The request is to operate within the normal unit-of-work protocols. The message is marked as being unavailable to other applications, but it is deleted from the queue only when the unit of work is committed. The message is made available again if the unit of work is backed out.

If neither this option nor MQGMO_NO_SYNCPOINT is specified, the inclusion of the get request in unit-of-work protocols is determined by the environment:

- On z/OS, the get request is within a unit of work.
- In all other environments, the get request is not within a unit of work.

MQGMO_SYNCPOINT_IF_PERSISTENT

```
public final static int
```

The request is to operate within the normal unit-of-work protocols, but only if the message retrieved is persistent. A persistent message has the value MQPER_PERSISTENT in the Persistence field in MQMD.

If the message is persistent, the queue manager processes the call as though the application had specified MQGMO_SYNCPOINT.

If the message is not persistent, the queue manager processes the call as though the application had specified MQGMO_NO_SYNCPOINT.

MQGMO_UNLOCK

```
public final static int
```

Unlock a message. The message to be unlocked must have been previously locked by an MQGET call with the MQGMO_LOCK option. If there is no message locked for this handle, the call completes with MQRC_NO_MSG_LOCKED.

This option is not valid with any options except MQGMO_NO_WAIT and MQGMO_NO_SYNCPOINT. Both of these options are assumed whether specified or not.

MQGMO_VERSION_1

```
public final static int
```

This is the version number of the get-message options structure. This value indicates version 1 of the structure.

MQGMO_VERSION_2

```
public final static int
```

This is the version number of the get-message options structure. This value indicates version 2 of the structure.

MQGMO_VERSION_3

```
public final static int
```

This is the version number of the get-message options structure. This value indicates version 3 of the structure.

MQGMO_WAIT

```
public final static int
```

The application waits until a suitable message arrives. The maximum time that the application waits is specified in WaitInterval.

If MQGET requests are inhibited, or MQGET requests become inhibited while waiting, the wait is canceled and the call completes with MQCC_FAILED and reason code MQRC_GET_INHIBITED, regardless of whether there are suitable messages on the queue.

MQGS_LAST_MSG_IN_GROUP

```
public final static char
```

This flag indicates that the message retrieved is the last in a group. This is also the value returned if the group consists of only one message.

MQGS_MSG_IN_GROUP

```
public final static char
```

This flag indicates that the message retrieved is in a group.

MQGS_NOT_IN_GROUP

```
public final static char
```

This flag indicates that the message retrieved is not in a group.

MQIA_ACCOUNTING_CONN_OVERRIDE

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine is applications can override the setting of the ACCTMQI and ACCTQDATA values in the Qmgr attribute. The value is one of MQMON_DISABLED or MQMON_ENABLED. The default is MQMON_DISABLED.

MQIA_ACCOUNTING_INTERVAL

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine how long before intermediate accounting records are written (in seconds). The value is an integer in the range 0 to 604800, with a default value of 1800 (30 minutes). Specify 0 to turn off intermediate records.

MQIA_ACCOUNTING_MQI

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the collection of accounting information for MQI data. The value is one of MQMON_ON or MQMON_OFF. The default is MQMON_OFF.

MQIA_ACCOUNTING_Q

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the collection of accounting information for queues. The value is one of MQMON_NONE, MQMON_OFF or MQMON_ON. The default is MQMON_NONE.

MQIA_APPL_TYPE

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the nature of the program to be started in response to the receipt of a trigger message. This information is for use by a trigger-monitor application that processes messages on the initiation queue. This value will be one of the MQAT_* values.

MQIA_ARCHIVE

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the way that archiving mode works. This value will be one of MQAR_NONE or MQAR_ALL.

MQIA_AUTH_INFO_TYPE

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the type of authentication information contained in an AuthInfoRecord. The value will always be MQAIT_CRL_LDAP, indicating that certificate revocation lists are stored on a LDAP server.

MQIA_AUTHORITY_EVENT

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine whether authorization (Not Authorized) events are generated. The value is one of MQEVR_DISABLED or MQEVR_ENABLED. For more information about events, see Monitoring WebSphere MQ.

MQIA_BACKOUT_THRESHOLD

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the backout threshold. Apart from allowing its value to be queried, the queue manager takes no action based on the value of this attribute.

MQIA_CHANNEL_AUTO_DEF

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the automatic definition of channels of type MQCHT_RECEIVER and MQCHT_SVRCONN. Automatic definition of MQCHT_CLUSSDR channels is always enabled. The value is one of MQCHAD_DISABLED or MQCHAD_ENABLED.

MQIA_CHANNEL_AUTO_DEF_EVENT

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine whether channel automatic-definition events are generated. It applies to channels of type MQCHT_RECEIVER, MQCHT_SVRCONN, and MQCHT_CLUSSDR. The value is one of MQEVR_DISABLED or MQEVR_ENABLED.

MQIA_CHANNEL_EVENT

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine whether channel events are generated. The value is one of MQEVR_EXCEPTION, MQEVR_ENABLED or MQEVR_DISABLED. The default is MQEVR_DISABLED.

MQIA_CLUSTER_Q_TYPE

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the cluster queue type.

MQIA_CLUSTER_WORKLOAD_LENGTH

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the maximum length of message data that is passed to the cluster workload exit.

MQIA_CLWL_MRU_CHANNELS

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the maximum number of most recently used channels in cluster workload balancing.

MQIA_CLWL_Q_PRIORITY

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the priority of a queue for cluster workload management purposes.

MQIA_CLWL_Q_RANK

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the rank of a queue for cluster workload management purposes.

MQIA_CLWL_USE_REMOTE_Q

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the behavior of a put when the target queue has both a local instance and at least one remote cluster instance. This value will be one of MQQF_CLWL_USEQ_ANY or MQQF_CLWL_USEQ_LOCAL.

MQIA_CODED_CHAR_SET_ID

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the character set used by the queue manager for all character string fields. The character set must be one that has single-byte characters for the characters that are valid in object names. It does not apply to application data carried in the message. The value depends on the environment.

MQIA_COMMAND_EVENT

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine whether command events are generated. The value is one of MQEVR_DISABLED, MQEVR_ENABLED or MQEVR_NO_DISPLAY. The default is MQEVR_DISABLED.

MQIA_COMMAND_LEVEL

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the level of system control commands supported by the queue manager. The value is one of the MQCMDL_LEVEL_* values.

MQIA_CONFIGURATION_EVENT

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine whether configuration events are generated.

MQIA_CURRENT_Q_DEPTH

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the number of messages currently on the queue. It is incremented during an MQPUT call, and during backout of an MQGET call. It is decremented during a non-browsing MQGET call, and during backout of an MQPUT call.

MQIA_DEF_BIND

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the default binding that is used when MQOO_BIND_AS_Q_DEF is specified on the MQOPEN call and the queue is a cluster queue. The value is one MQBND_BIND_ON_OPEN or MQBND_BIND_NOT_FIXED.

MQIA_DEF_INPUT_OPEN_OPTION

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the default way in which to open the queue for input. It applies if the MQOO_INPUT_AS_Q_DEF option is specified on the MQOPEN call when the queue is opened. The value is one of MQOO_INPUT_EXCLUSIVE or MQOO_INPUT_SHARED.

MQIA_DEF_PERSISTENCE

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the default persistence of messages on the queue. It applies if MQPER_PERSISTENCE_AS_Q_DEF is specified in the message descriptor when the message is put. The value is one of MQPER_PERSISTENT or MQPER_NON_PERSISTENT.

MQIA_DEF_PRIORITY

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the default priority for messages on the queue. This applies if MQPRI_PRIORITY_AS_Q_DEF is specified in the message descriptor when the message is put on the queue.

MQIA_DEFINITION_TYPE

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine how the queue was defined. The value is one of MQQDT_PREDEFINED, MQQDT_PERMANENT_DYNAMIC, MQQDT_TEMPORARY_DYNAMIC or MQQDT_SHARED_DYNAMIC.

MQIA_DIST_LISTS

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine whether distribution-list messages can be placed on the queue. The value is one of MQDL_SUPPORTED or MQDL_NOT_SUPPORTED.

MQIA_EXPIRY_INTERVAL

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the interval between scans for expired messages. It is either a time interval (in seconds) in the range 1 to 99,999,999, or the special value MQEXPI_OFF to indicate that the queue manager does not scan the queues looking for expired messages.

MQIA_FIRST

```
public final static int
```

This defines the start of the range of valid integer attribute selectors. The integer and character attribute selectors are allocated within two different ranges, with MQIA_* selectors within the range MQIA_FIRST through MQIA_LAST.

MQIA_HARDEN_GET_BACKOUT

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine if hardening is used to ensure that the backout count for messages on this queue is accurate. The values is one of MQQA_BACKOUT_HARDENED or MQQA_BACKOUT_NOT_HARDENED. The default is MQQA_BACKOUT_NOT_HARDENED.

MQIA_HIGH_Q_DEPTH

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the maximum number of messages on the queue since the queue statistics were last reset.

MQIA_IGQ_PUT_AUTHORITY

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the type of authority checking that is performed when the local intra-group queuing agent (IGQ agent) removes a message from the shared transmission queue and places the message on a local queue. It applies only if the local queue manager is a member of a queue-sharing group. The value is one of MQIGQPA_DEFAULT, MQIGQPA_CONTEXT, MQIGQPA_ONLY_IGQ or MQIGQPA_ALTERNATE_OR_IGQ.

MQIA_INDEX_TYPE

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the type of index that the queue manager maintains for messages on the queue. The type of index required depends on how the application retrieves messages, and whether the queue is a shared queue or a nonshared queue.

MQIA_INHIBIT_EVENT

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine whether inhibit (Inhibit Get and Inhibit Put) events are generated. The value is one of

MQEVR_DISABLED or MQEVR_ENABLED. On z/OS, you cannot use the MQINQ call to determine the value of this attribute.

MQIA_INHIBIT_GET

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine whether get operations for this queue are allowed. If the queue is an alias queue, get operations must be allowed for both the alias and the base queue at the time of the get operation, for the MQGET call to succeed. The value is one of MQQA_GET_INHIBITED or MQQA_GET_ALLOWED.

MQIA_INHIBIT_PUT

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine whether put operations for this queue are allowed. If there is more than one definition in the queue-name resolution path, put operations must be allowed for every definition in the path (including any queue-manager alias definitions) at the time of the put operation. The value is one of MQQA_PUT_INHIBITED or MQQA_PUT_ALLOWED.

MQIA_INTRA_GROUP_QUEUING

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine whether intra-group queuing is enabled for the queue-sharing group. This attribute applies only if the local queue manager is a member of a queue-sharing group, and is supported only on z/OS. The value is one of MQIGQ_DISABLED or MQIGQ_ENABLED.

MQIA_LAST

```
public final static int
```

This defines the end of the range of valid integer attribute selectors. The integer and character attribute selectors are allocated within two different ranges, with MQIA_* selectors within the range MQIA_FIRST through MQIA_LAST.

MQIA_LAST_USED

```
public final static int
```

This defines the highest value in the range of valid integer attribute selectors that the queue manager will accept.

MQIA_LOCAL_EVENT

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine whether local error events are generated. The value is one of MQEVR_DISABLED or MQEVR_ENABLED. The default is MQEVR_DISABLED.

MQIA_MAX_HANDLES

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the maximum number of open handles that any one task can use concurrently. Each

successful MQOPEN call for a single queue (or for an object that is not a queue) uses one handle. That handle becomes available for reuse when the object is closed.

The value is in the range 1 to 999,999,999. The default value is determined by the environment. On z/OS, the default value is 100. In all other environments, the default value is 256.

MQIA_MAX_MSG_LENGTH

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the length of the longest physical message that the queue manager can handle. However, because the MaxMsgLength queue-manager attribute can be set independently of the MaxMsgLength queue attribute, the longest physical message that can be placed on a queue is the lesser of those two values.

The lower limit for the MaxMsgLength attribute is 32 KB (32 768 bytes). The upper limit is 100 MB (104 857 600 bytes).

MQIA_MAX_PRIORITY

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the maximum message priority supported by the queue manager. Priorities range from zero (lowest) to MaxPriority (highest).

MQIA_MAX_Q_DEPTH

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the defined upper limit for the number of physical messages that can exist on the queue at any one time. An attempt to put a message on a queue that already contains MaxQDepth messages fails with reason code MQRC_Q_FULL.

The value of this attribute is zero or greater. The upper limit is determined by the environment. On AIX, HP-UX, z/OS, Solaris, Linux, and Windows, the value cannot exceed 999,999,999. On i5/OS, the value cannot exceed 640 000.

MQIA_MAX_UNCOMMITTED_MSGS

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the maximum number of uncommitted messages that can exist within a unit of work.

MQIA_MSG_DELIVERY_SEQUENCE

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the order in which the MQGET call returns messages to the application. It can be either MQMDS_FIFO (messages are returned in first in, first out order) or MQMDS_PRIORITY (higher priority messages are returned first).

MQIA_MSG_DEQ_COUNT

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the number of messages that were removed from the queue since the queue statistics were last reset.

MQIA_MSG_ENQ_COUNT

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the number of messages that were put on the queue since the queue statistics were last reset.

MQIA_NAME_COUNT

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the number of names in the namelist. It is greater than or equal to zero. MQNC_MAX_NAMELIST_NAME_COUNT defines the maximum number of names in a namelist.

MQIA_NAMELIST_TYPE

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the nature of the names in the namelist, and indicates how the namelist is used. The value is one of MQNT_NONE, MQNT_Q, MQNT_CLUSTER or MQNT_AUTH_INFO. This attribute is supported only on z/OS.

MQIA_NPM_CLASS

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the reliability goal for nonpersistent messages. This specifies the circumstances under which nonpersistent messages put on this queue are discarded. The value is one of MQNPM_CLASS_NORMAL or MQNPM_CLASS_HIGH.

MQIA_OPEN_INPUT_COUNT

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the number of handles that are currently valid for removing messages from the queue by means of the MQGET call. It is the total number of such handles known to the local queue manager. If the queue is a shared queue, the count does not include opens for input that were performed for the queue at other queue managers in the queue-sharing group to which the local queue manager belongs.

MQIA_OPEN_OUTPUT_COUNT

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the number of handles that are currently valid for adding messages to the queue by means of the MQPUT call. It is the total number of such handles known to the local queue manager; it does not include opens for output that were performed for this queue at remote queue managers. If the queue is a shared queue, the count does not include opens for output that were performed for the queue at other queue managers in the queue-sharing group to which the local queue manager belongs.

MQIA_PERFORMANCE_EVENT

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine whether performance-related events are generated. The value is one of MQEVR_DISABLED or MQEVR_ENABLED. On z/OS, you cannot use the MQINQ call to determine the value of this attribute.

MQIA_PLATFORM

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the operating system on which the queue manager is running. The value will be one of the MQPL_* values.

MQIA_Q_DEPTH_HIGH_EVENT

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine whether Queue Depth High events are generated. A Queue Depth High event indicates that an application has put a message on a queue, and this has caused the number of messages on the queue to become greater than or equal to the queue depth high threshold. The value is one of MQEVR_DISABLED or MQEVR_ENABLED.

MQIA_Q_DEPTH_HIGH_LIMIT

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the threshold against which the queue depth is compared to generate a Queue Depth High event. This event indicates that an application has put a message on a queue, and that this has caused the number of messages on the queue to become greater than or equal to the queue depth high threshold.

MQIA_Q_DEPTH_LOW_EVENT

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine whether Queue Depth Low events are generated. A Queue Depth Low event indicates that an application has removed a message from a queue, and this has caused the number of messages on the queue to become less than or equal to the queue depth low threshold. The value is one of MQEVR_DISABLED or MQEVR_ENABLED.

MQIA_Q_DEPTH_LOW_LIMIT

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the threshold against which the queue depth is compared to generate a Queue Depth Low event. This event indicates that an application has removed a message from a queue, and that this has caused the number of messages on the queue to become less than or equal to the queue depth low threshold.

MQIA_Q_DEPTH_MAX_EVENT

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine whether Queue Full events are generated. A Queue Full event indicates that a put to a

queue has been rejected because the queue is full, that is, the queue depth has already reached its maximum value. The value is one of MQEVR_DISABLED or MQEVR_ENABLED.

MQIA_Q_SERVICE_INTERVAL

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the service interval used for comparison to generate Service Interval High and Service Interval OK events. The interval is set in milliseconds, and its value is not less than zero and not greater than 999,999,999.

MQIA_Q_SERVICE_INTERVAL_EVENT

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine whether Service Interval High or Service Interval OK events are generated. This value is one of MQQSIE_HIGH, MQQSIE_OK or MQQSIE_NONE.

MQIA_Q_TYPE

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the type of queue. It has one of the following values: MQQT_ALIAS, MQQT_CLUSTER, MQQT_LOCAL or MQQT_REMOTE.

MQIA_QSG_DISP

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the disposition of the queue. The value is one of MQQSGD_Q_MGR, MQQSGD_COPY or MQQSGD_SHARED. This attribute is only supported on z/OS.

MQIA_REMOTE_EVENT

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine whether remote error events are generated. The value is one of MQEVR_DISABLED or MQEVR_ENABLED.

MQIA_RETENTION_INTERVAL

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the period of time for which to retain the queue. After this time has elapsed, the queue is eligible for deletion. The time is measured in hours, counting from the date and time when the queue was created. This information is provided to enable a housekeeping application or the operator to identify and delete queues that are no longer required.

MQIA_SCOPE

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine whether an entry for this queue also exists in a cell directory. A cell directory is provided by an installable Name service. The value is one of MQSCO_Q_MGR or MQSCO_CELL.

MQIA_SHAREABILITY

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine whether the queue can be opened for input multiple times concurrently. The value is one of MQQA_SHAREABLE or MQQA_NOT_SHAREABLE.

MQIA_SSL_EVENT

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine whether SSL events are generated. The value is one of MQEVR_DISABLED or MQEVR_ENABLED.

MQIA_SSL_FIPS_REQUIRED

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine if only FIPS-certified algorithms are to be used if the cryptography is executed in WebSphere MQ-provided software. The value is one of MQSSL_FIPS_NO or MQSSL_FIPS_YES. The default is MQSSL_FIPS_NO.

MQIA_SSL_RESET_COUNT

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine when SSL channel message channel agents (MCAs) that initiate communication reset the secret key used for encryption on the channel. The value represents the total number of unencrypted bytes that are sent and received on the channel before the secret key is renegotiated. The number of bytes includes control information sent by the MCA.

The value is a number between 0 and 999,999,999, with a default value of 0.

MQIA_SSL_TASKS

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the number of server subtasks for processing SSL calls.

MQIA_START_STOP_EVENT

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine whether start and stop events are generated. The value is one of MQEVR_DISABLED or MQEVR_ENABLED.

MQIA_STATISTICS_AUTO_CLUSSDR

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine whether to collect online monitoring data for auto-defined cluster sender channels. The value is one of MQMON_Q_MGR, MQMON_OFF, MQMON_LOW, MQMON_MEDIUM or MQMON_HIGH. The default is MQMON_Q_MGR.

MQIA_STATISTICS_CHANNEL

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the collection of statistics data for channels. The value is one of MQMON_NONE, MQMON_OFF, MQMON_LOW, MQMON_MEDIUM or MQMON_HIGH. The default is MQMON_NONE.

MQIA_STATISTICS_INTERVAL

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine how often (in seconds) to write statistics monitoring data to the monitoring queue. The value is an integer in the range 0 to 604800, with a default value of 1800 (30 minutes).

MQIA_STATISTICS_MQI

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the collection of statistics monitoring information for the queue manager. The value is one of MQMON_ON or MQMON_OFF. The default is MQMON_OFF.

MQIA_STATISTICS_Q

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the collection of statistics data for queues. The value is one of MQMON_NONE, MQMON_OFF or MQMON_ON. The default is MQMON_NONE.

MQIA_SYNCPOINT

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine whether the local queue manager supports units of work and syncpointing with the MQGET, MQPUT, and MQPUT1 calls. The value is one of MQSP_AVAILABLE or MQSP_NOT_AVAILABLE.

MQIA_TIME_SINCE_RESET

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the time, in seconds, since the statistics were last reset. The value recorded by this timer is also used as the interval time in queue service interval events.

MQIA_TRACE_ROUTE_RECORDING

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the recording of trace route information. The value is one of MQROUTE_DISABLED, MQROUTE_RECORDING_Q or MQROUTE_RECORDING_MSG.

MQIA_TRIGGER_CONTROL

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine whether trigger messages are written to an initiation queue to start an application to service the queue. This is one of MQTC_OFF or MQTC_ON.

MQIA_TRIGGER_DEPTH

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the number of messages of priority TriggerMsgPriority or greater that must be on the queue before a trigger message is written. This applies when TriggerType is set to MQTT_DEPTH. The value of TriggerDepth is 1 or greater.

MQIA_TRIGGER_INTERVAL

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine a time interval (in milliseconds) used to restrict the number of trigger messages. This is relevant only when the TriggerType is MQTT_FIRST. In this case trigger messages are usually generated only when a suitable message arrives on the queue, and the queue was previously empty. Under certain circumstances, however, an additional trigger message can be generated with MQTT_FIRST triggering even if the queue was not empty. These additional trigger messages are not generated more often than every TriggerInterval milliseconds.

The value is not less than 0 and not greater than 999,999,999. The default value is 999,999,999.

MQIA_TRIGGER_MSG_PRIORITY

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the message priority below which messages do not contribute to the generation of trigger messages (that is, the queue manager ignores these messages when deciding whether to generate a trigger message). TriggerMsgPriority can be in the range zero (lowest) to MaxPriority. A value of zero causes all messages to contribute to the generation of trigger messages.

MQIA_TRIGGER_TYPE

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine the conditions under which trigger messages are written as a result of messages arriving on this queue. The value is one of MQTT_NONE, MQTT_FIRST, MQTT_EVERY or MQTT_DEPTH.

MQIA_USAGE

```
public final static int
```

This integer attribute selector is used with an MQINQ call to determine what the queue is used for. The value is one of MQUS_NORMAL or MQUS_TRANSMISSION.

MQIAV_NOT_APPLICABLE

```
public final static int
```

This indicates that an integer attribute (IntAttrs) value is not applicable.

MQIAV_UNDEFINED

```
public final static int
```

This indicates that an integer attribute (IntAttrs) value is undefined.

MQMD_VERSION_1

```
public final static int
```

This is the message descriptor structure version number. This value indicates version 1 of the structure.

MQMD_VERSION_2

```
public final static int
```

This is the message descriptor structure version number. This value indicates version 2 of the structure.

MQMDS_FIFO

```
public final static int
```

This determines the order in which the MQGET call returns messages to the application. With this option, messages are returned in FIFO order (first in, first out). An MQGET call returns the first message that satisfies the selection criteria specified on the call, regardless of the priority of the message.

MQMDS_PRIORITY

```
public final static int
```

This determines the order in which the MQGET call returns messages to the application. Messages are returned in priority order. An MQGET call returns the highest-priority message that satisfies the selection criteria specified on the call. Within each priority level, messages are returned in FIFO order (first in, first out).

MQMF_ACCEPT_UNSUP_IF_XMIT_MASK

```
public final static int
```

This mask identifies the bit positions within the MsgFlags field where message flags that are not supported by the local queue manager are nevertheless accepted on the MQPUT or MQPUT1 calls provided that both of the following conditions are satisfied:

- The message is destined for a remote queue manager.
- The application is not putting the message directly on a local transmission queue

This subfield occupies bit positions 12 through 19.

MQMF_ACCEPT_UNSUP_MASK

```
public final static int
```

This mask identifies the bit positions within the MsgFlags field where message flags that are not supported by the local queue manager are nevertheless accepted on the MQPUT or MQPUT1 calls.

This subfield occupies bit positions 0 through 11.

MQMF_LAST_MSG_IN_GROUP

```
public final static int
```

Message is the last logical message in a group. If this flag is set, the queue manager turns on MQMF_MSG_IN_GROUP in the copy of MQMD that is sent with the message, but does not alter the settings of these flags in the MQMD provided by the application on the MQPUT or MQPUT1 call.

MQMF_LAST_SEGMENT

```
public final static int
```

Message is the last segment of a logical message. If this flag is set, the queue manager turns on MQMF_SEGMENT in the copy of MQMD that is sent with the message, but does not alter the settings of these flags in the MQMD provided by the application on the MQPUT or MQPUT1 call.

MQMF_MSG_IN_GROUP

```
public final static int
```

Indicates that the message is a member of a group.

MQMF_NONE

```
public final static int
```

No message flags (default message attributes). This inhibits segmentation, and indicates that the message is not in a group and is not a segment of a logical message. MQMF_NONE is defined to aid program documentation. It is not intended that this flag be used with any other, but as its value is zero, such use cannot be detected.

MQMF_REJECT_UNSUP_MASK

```
public final static int
```

This mask identifies the bit positions within the MsgFlags field where message flags that are not supported by the local queue manager cause the MQPUT or MQPUT1 call to fail with completion code MQCC_FAILED and reason code MQRC_MSG_FLAGS_ERROR.

This subfield occupies bit positions 20 through 31.

MQMF_SEGMENT

```
public final static int
```

Message is a segment of a logical message. When MQMF_SEGMENT is specified without MQMF_LAST_SEGMENT, the length of the application message data in the segment (excluding the lengths of any MQ header structures that might be present) must be at least one. If the length is zero, the MQPUT or MQPUT1 call fails with reason code MQRC_SEGMENT_LENGTH_ZERO.

MQMF_SEGMENTATION_ALLOWED

```
public final static int
```

This option allows the message to be broken into segments by the queue manager. If specified for a message that is already a segment, this option allows the segment to be broken into smaller segments. MQMF_SEGMENTATION_ALLOWED can be set without either MQMF_SEGMENT or MQMF_LAST_SEGMENT being set.

MQMF_SEGMENTATION_INHIBITED

```
public final static int
```

This option prevents the message being broken into segments by the queue manager. If specified for a message that is already a segment, this option prevents the segment being broken into smaller segments.

The value of this flag is binary zero. This is the default.

MQMI_NONE

```
public final static byte[]
```

No message identifier is specified. The value is binary zero for the length of the field.

MQMO_MATCH_CORREL_ID

```
public final static int
```

The message to be retrieved must have a correlation identifier that matches the value of the CorrelId field in the MsgDesc parameter of the MQGET call. This match is in addition to any other matches that might apply (for example, the message identifier).

If you omit this option, the CorrelId field in the MsgDesc parameter is ignored, and any correlation identifier will match.

MQMO_MATCH_GROUP_ID

```
public final static int
```

The message to be retrieved must have a group identifier that matches the value of the GroupId field in the MsgDesc parameter of the MQGET call. This match is in addition to any other matches that might apply (for example, the correlation identifier).

If you omit this option, the GroupId field in the MsgDesc parameter is ignored, and any group identifier will match.

MQMO_MATCH_MSG_ID

```
public final static int
```

The message to be retrieved must have a message identifier that matches the value of the MsgId field in the MsgDesc parameter of the MQGET call. This match is in addition to any other matches that might apply (for example, the correlation identifier).

If you omit this option, the MsgId field in the MsgDesc parameter is ignored, and any message identifier will match.

MQMO_MATCH_MSG_SEQ_NUMBER

```
public final static int
```

The message to be retrieved must have a message sequence number that matches the value of the MsgSeqNumber field in the MsgDesc parameter of MQGMO the MQGET call. This match is in addition to any other matches that might apply (for example, the group identifier).

If you omit this option, the MsgSeqNumber field in the MsgDesc parameter is ignored, and any message sequence number will match.

MQMO_MATCH_MSG_TOKEN

```
public final static int
```

The message to be retrieved must have a message token that matches the value of the MsgToken field in the MQGMO structure specified on the MQGET call.

If you omit this option, the MsgToken field in MQGMO is ignored, and any message token will match.

MQMO_MATCH_OFFSET

```
public final static int
```

The message to be retrieved must have an offset that matches the value of the Offset field in the MsgDesc parameter of the MQGET call. This match is in addition to any other matches that might apply (for example, the message sequence number).

If you omit this option or it is not specified, the Offset field in the MsgDesc parameter is ignored, and any offset will match.

MQMO_NONE

```
public final static int
```

Do not use matches in selecting the message to be returned. All messages on the queue are eligible for retrieval. MQMO_NONE aids program documentation. It is not intended that this option be used with any other MQMO_* option, but as its value is zero, such use cannot be detected.

MQMT_APPL_FIRST

```
public final static int
```

This defines the lowest value for application-defined message types.

MQMT_APPL_LAST

```
public final static int
```

This defines the highest value for application-defined message types.

MQMT_DATAGRAM

```
public final static int
```

The message is one that does not require a reply.

MQMT_REPLY

```
public final static int
```

The message is the reply to an earlier request message (MQMT_REQUEST). The message must be sent to the queue indicated by the ReplyToQ field of the request message. Use the Report field of the request to control how to set the MsgId and CorrelId of the reply.

Note: The queue manager does not enforce the request-reply relationship; this is an application responsibility.

MQMT_REPORT

```
public final static int
```

The message is reporting on an expected or unexpected occurrence, usually related to another message. For example, a request message was received that contained data that was not valid. Send the message to the queue indicated by the ReplyToQ field of the message descriptor of the original message. Set the Feedback fields to indicate the nature of the report. Use the Report field of the original message to control how to set the MsgId and CorrelId of the report message.

Report messages generated by the queue manager or message channel agent are always sent to the ReplyToQ queue, with the Feedback and CorrelId fields set.

MQMT_REQUEST

```
public final static int
```

The message is one that requires a reply. Specify the name of the queue to which to send the reply in the ReplyToQ field. The Report field indicates how to set the MsgId and CorrelId of the reply.

MQMT_SYSTEM_FIRST

```
public final static int
```

This defines the lowest value for system-defined message types.

MQMT_SYSTEM_LAST

```
public final static int
```

This defines the highest value for system-defined message types.

MQMTOK_NONE

```
public final static byte[]
```

No message token is specified. The value is binary zero for the length of the field.

MQOL_UNDEFINED

```
public final static int
```

Original length of message is not defined. This field is relevant only for report messages that are segments. It specifies the length of the message segment to which the report message relates; it does not specify the length of the logical message of which the segment forms part, or the length of the data in the report message.

MQOO_ALTERNATE_USER_AUTHORITY

```
public final static int
```

The AlternateUserId field in the ObjDesc parameter contains a user identifier to use to validate this MQOPEN call. The call can succeed only if this AlternateUserId is authorized to open the object with the specified access options, regardless of whether the user identifier under which the application is running is authorized to do so.

This option is valid for all types of object.

MQOO_BIND_AS_Q_DEF

```
public final static int
```

The local queue manager binds the queue handle in the way defined by the DefBind queue attribute. The value of this attribute is either MQBND_BIND_ON_OPEN or MQBND_BIND_NOT_FIXED.

MQOO_BIND_AS_Q_DEF is the default if neither MQOO_BIND_ON_OPEN nor MQOO_BIND_NOT_FIXED is specified.

MQOO_BIND_NOT_FIXED

```
public final static int
```

This stops the local queue manager binding the queue handle to a particular instance of the destination queue. As a result, successive MQPUT calls using this handle send the messages to different instances of the destination queue, or to the same instance but by different routes. It also allows the instance selected to be changed subsequently by the local queue manager, by a remote queue manager, or by a message channel agent (MCA), according to network conditions.

MQOO_BIND_ON_OPEN

```
public final static int
```

The local queue manager binds the queue handle to a particular instance of the destination queue when the queue is opened. As a result, all messages put using this handle are sent to the same instance of the destination queue, and by the same route.

This option is valid only for queues, and affects only cluster queues. If specified for a queue that is not a cluster queue, the option is ignored.

MQOO_BROWSE

```
public final static int
```

Open the queue to browse messages. The queue is opened for use with subsequent MQGET calls with one of the following options: MQGMO_BROWSE_FIRST, MQGMO_BROWSE_NEXT and MQGMO_BROWSE_MSG_UNDER_CURSOR. This is allowed even if the queue is currently open for MQOO_INPUT_EXCLUSIVE. An MQOPEN call with the MQOO_BROWSE option establishes a browse cursor, and positions it logically before the first message on the queue.

MQOO_FAIL_IF_QUIESCING

```
public final static int
```

The MQOPEN call fails if the queue manager is in quiescing state. This option is valid for all types of object.

MQOO_INPUT_AS_Q_DEF

```
public final static int
```

Open the queue to get messages using the queue-defined default. The queue is opened for use with subsequent MQGET calls. The type of access is either shared or exclusive, depending on the value of the DefInputOpenOption queue attribute.

MQOO_INPUT_EXCLUSIVE

```
public final static int
```

Open the queue to get messages with exclusive access. The queue is opened for use with subsequent MQGET calls. The call fails with reason code MQRC_OBJECT_IN_USE if the queue is currently open by this or another application for input of any type.

MQOO_INPUT_SHARED

```
public final static int
```

Open the queue to get messages with shared access. The queue is opened for use with subsequent MQGET calls. The call can succeed if the queue is currently open by this or another application with MQOO_INPUT_SHARED, but fails with reason code MQRC_OBJECT_IN_USE if the queue is currently open with MQOO_INPUT_EXCLUSIVE.

MQOO_INQUIRE

```
public final static int
```

Open the object to query attributes. The queue, namelist, process definition, or queue manager is opened for use with subsequent MQINQ calls. This option is valid for all types of object other than distribution lists. It is not valid if ObjectQMgrName is the name of a queue manager alias; this is true even if the value of the RemoteQMgrName attribute in the local definition of a remote queue used for queue-manager aliasing is the name of the local queue manager.

MQOO_OUTPUT

```
public final static int
```

Open the queue to put messages. The queue is opened for use with subsequent MQPUT calls.

An MQOPEN call with this option can succeed even if the InhibitPut queue attribute is set to MQQA_PUT_INHIBITED (although subsequent MQPUT calls will fail while the attribute is set to this value). This option is valid for all types of queue, including distribution lists.

MQOO_PASS_ALL_CONTEXT

```
public final static int
```

This allows the MQPMO_PASS_ALL_CONTEXT option to be specified in the PutMsgOpts parameter when a message is put on a queue. This gives the message the identity and origin context information from an input queue that was opened with the MQOO_SAVE_ALL_CONTEXT option. This option implies MQOO_PASS_IDENTITY_CONTEXT, which need not therefore be specified. The MQOO_OUTPUT option must be specified. For more information on message context, see *WebSphere MQ Application Programming Guide*.

This option is valid for all types of queue, including distribution lists.

MQOO_PASS_IDENTITY_CONTEXT

```
public final static int
```

This allows the MQPMO_PASS_IDENTITY_CONTEXT option to be specified in the PutMsgOpts parameter when a message is put on a queue. This gives the message the identity context information from an input queue that was opened with the

MQOO_SAVE_ALL_CONTEXT option. The MQOO_OUTPUT option must be specified. For more information on message context, see *WebSphere MQ Application Programming Guide.*

This option is valid for all types of queue, including distribution lists.

MQOO_RESOLVE_LOCAL_Q

```
public final static int
```

Fill the ResolvedQName in the MQOD structure with the name of the local queue that was opened. Similarly, the ResolvedQMgrName is filled with the name of the local queue manager hosting the local queue.

MQOO_RESOLVE_LOCAL_QUEUE

```
public final static int
```

Deprecated

use MQC.MQOO_RESOLVE_LOCAL_Q instead.

MQOO_SAVE_ALL_CONTEXT

```
public final static int
```

Context information is associated with this queue handle. This information is set from the context of any message retrieved using this handle. For more information on message context, see *WebSphere MQ Application Programming Guide*.

This option is valid only for local, alias, and model queues; it is not valid for remote queues, distribution lists, and objects that are not queues.

MQOO_SET

```
public final static int
```

Open the queue to set attributes. The queue is opened for use with subsequent MQSET calls. This option is valid for all types of object other than distribution lists. It is not valid if ObjectQMgrName is the name of a queue manager alias; this is true even if the value of the RemoteQMgrName attribute in the local definition of a remote queue used for queue-manager aliasing is the name of the local queue manager.

MQOO_SET_ALL_CONTEXT

```
public final static int
```

This allows the MQPMO_SET_ALL_CONTEXT option to be specified in the PutMsgOpts parameter when a message is put on a queue. This gives the message the identity and origin context information contained in the MsgDesc parameter specified on the MQPUT or MQPUT1() call. The MQOO_OUTPUT option must be specified. For more information on message context, see *WebSphere MQ Application Programming Guide.*

This option is valid for all types of queue, including distribution lists.

MQOO_SET_IDENTITY_CONTEXT

```
public final static int
```

This allows the MQPMO_SET_IDENTITY_CONTEXT option to be specified in the PutMsgOpts parameter when a message is put on a queue. This gives the message the identity context information contained in the MsgDesc parameter specified on the MQPUT() or MQPUT1 call. This option implies MQOO_PASS_IDENTITY_CONTEXT, which need not therefore be specified. The MQOO_OUTPUT option must be specified. For more information on message context, see *WebSphere MQ Application Programming Guide*.

This option is valid for all types of queue, including distribution lists.

MQPER_NOT_PERSISTENT

```
public final static int
```

The message does not usually survive system failures or queue manager restarts. This applies even if an intact copy of the message is found on auxiliary storage when the queue manager restarts.

In the case of shared queues, nonpersistent messages survive queue manager restarts in the queue-sharing group, but do not survive failures of the coupling facility used to store messages on the shared queues.

MQPER_PERSISTENCE_AS_Q_DEF

```
public final static int
```

If the queue is not a cluster queue, the persistence of the message is taken from the DefPersistence attribute defined at the local queue manager, even if the destination queue manager is remote.

If the queue is a cluster queue, the persistence of the message is taken from the DefPersistence attribute defined at the destination queue manager that owns the particular instance of the queue on which the message is placed.

MQPER_PERSISTENT

```
public final static int
```

The message survives system failures and restarts of the queue manager. Once the message has been put, and the unit of work in which it was put has been committed (if the message is put as part of a unit of work), the message is preserved on auxiliary storage. It remains there until the message is removed from the queue, and the unit of work of which it was part has been committed (if the message is retrieved as part of a unit of work).

MQPL_AIX

```
public final static int
```

This indicates that the operating system on which the queue manager is running is AIX (same value as MQPL_UNIX).

MQPL_MVS

```
public final static int
```

This indicates that the operating system on which the queue manager is running is MVS/ESA™ (same value as MQPL_ZOS).

MQPL_NSK

```
public final static int
```

This indicates that the operating system on which the queue manager is running is Compaq NonStop Kernel.

MQPL_OS2
```
public final static int
```

This indicates that the operating system on which the queue manager is running is OS/2.

MQPL_OS400
```
public final static int
```

This indicates that the operating system on which the queue manager is running is OS/400 or i5/OS.

MQPL_UNIX
```
public final static int
```

This indicates that the operating system on which the queue manager is running is a UNIX system (same value as MQPL_AIX).

MQPL_WINDOWS
```
public final static int
```

This indicates that the operating system on which the queue manager is running is Windows 3.1.

MQPL_WINDOWS_NT
```
public final static int
```

This indicates that the operating system on which the queue manager is running is Windows NT®, Windows 2000 or Windows XP.

MQPMO_ALTERNATE_USER_AUTHORITY
```
public final static int
```

This indicates that the AlternateUserId field in the ObjDesc parameter of the MQPUT1 call contains a user identifier that is to be used to validate authority to put messages on the queue. The call can succeed only if this AlternateUserId is authorized to open the queue with the specified options, regardless of whether the user identifier under which the application is running is authorized to do so.

MQPMO_DEFAULT_CONTEXT
```
public final static int
```

The message is to have default context information associated with it, for both identity and origin.

MQPMO_FAIL_IF_QUIESCING
```
public final static int
```

This option forces the MQPUT or MQPUT1 call to fail if the queue manager is in the quiescing state.

MQPMO_LOGICAL_ORDER
```
public final static int
```

This option tells the queue manager how the application puts messages in groups and segments of logical messages. It can be specified only on the MQPUT call; it is not valid on the MQPUT1 call. See *WebSphere MQ Application Programming Reference* for more information on this option.

MQPMO_NEW_CORREL_ID

```
public final static int
```

The queue manager replaces the contents of the CorrelId field in MQMD with a new correlation identifier. This correlation identifier is sent with the message, and returned to the application on output from the MQPUT or MQPUT1 call.

MQPMO_NEW_MSG_ID

```
public final static int
```

The queue manager replaces the contents of the MsgId field in MQMD with a new message identifier. This message identifier is sent with the message, and returned to the application on output from the MQPUT or MQPUT1 call.

MQPMO_NO_CONTEXT

```
public final static int
```

Both identity and origin context are set to indicate no context. This means that the context fields in MQMD are set to:

- Blank, for character fields
- Null, for byte fields
- Zero, for numeric fields

MQPMO_NO_SYNCPOINT

```
public final static int
```

The request is to operate outside the normal unit-of-work protocols. The message is available immediately, and it cannot be deleted by backing out a unit of work. If neither this option nor MQPMO_SYNCPOINT is specified, the inclusion of the put request in unit-of-work protocols is determined by the environment, see MQPMO_SYNCPOINT.

Do not specify MQPMO_NO_SYNCPOINT with MQPMO_SYNCPOINT.

MQPMO_NONE

```
public final static int
```

Use this value to indicate that no other options have been specified. All options assume their default values. MQPMO_NONE is defined to aid program documentation; it is not intended that this option be used with any other, but as its value is zero, such use cannot be detected.

MQPMO_PASS_ALL_CONTEXT

```
public final static int
```

The message is to have context information associated with it. Both identity and origin context are taken from the queue handle specified in the Context field.

MQPMO_PASS_IDENTITY_CONTEXT

```
public final static int
```

The message is to have context information associated with it. Identity context is taken from the queue handle specified in the Context field. Origin context information is generated by the queue manager in the same way that it is for MQPMO_DEFAULT_CONTEXT.

MQPMO_RESOLVE_LOCAL_Q

```
public final static int
```

Use this option to fill ResolvedQName in the MQPMO structure with the name of the local queue to which the message is put, and ResolvedQMgrName with the name of the local queue manager that hosts the local queue.

MQPMO_SET_ALL_CONTEXT

```
public final static int
```

The message is to have context information associated with it. The application specifies the identity and origin context in the MQMD structure. For more information on message context.

MQPMO_SET_IDENTITY_CONTEXT

```
public final static int
```

The message is to have context information associated with it. The application specifies the identity context in the MQMD structure. Origin context information is generated by the queue manager in the same way that it is for MQPMO_DEFAULT_CONTEXT.

MQPMO_SYNCPOINT

```
public final static int
```

The request is to operate within the normal unit-of-work protocols. The message is not visible outside the unit of work until the unit of work is committed. If the unit of work is backed out, the message is deleted.

If neither this option nor MQPMO_NO_SYNCPOINT is specified, the inclusion of the put request in unit-of-work protocols is determined by the environment:

- On z/OS, the put request is within a unit of work.
- In all other environments, the put request is not within a unit of work.

Do not specify MQPMO_NO_SYNCPOINT with MQPMO_SYNCPOINT.

MQPMO_VERSION_1

```
public final static int
```

This is the version number of the put message options structure. This value indicates version 1 of the structure.

MQPMO_VERSION_2

```
public final static int
```

This is the version number of the put message options structure. This value indicates version 2 of the structure.

MQPMRF_ACCOUNTING_TOKEN

```
public final static int
```

This flag indicates that an accounting token field is present in the put message records provided by the application. This is only used when sending messages to a distribution list. For fields that are present, the queue manager uses for each destination the values from the fields in the corresponding put message record. For fields that are absent, the queue manager uses the values from the MQMD structure.

If you specify this flag, also specify either MQPMO_SET_IDENTITY_CONTEXT or MQPMO_SET_ALL_CONTEXT in the Options field; if this condition is not satisfied, the call fails with reason code MQRC_PMO_RECORD_FLAGS_ERROR.

MQPMRF_CORREL_ID

```
public final static int
```

This flag indicates that a correlation ID field is present in the put message records provided by the application. This is only used when sending messages to a distribution list. For fields that are present, the queue manager uses for each destination the values from the fields in the corresponding put message record. For fields that are absent, the queue manager uses the values from the MQMD structure.

MQPMRF_FEEDBACK

```
public final static int
```

This flag indicates that a feedback field is present in the put message records provided by the application. This is only used when sending messages to a distribution list. For fields that are present, the queue manager uses for each destination the values from the fields in the corresponding put message record. For fields that are absent, the queue manager uses the values from the MQMD structure.

MQPMRF_GROUP_ID

```
public final static int
```

This flag indicates that a group ID field is present in the put message records provided by the application. This is only used when sending messages to a distribution list. For fields that are present, the queue manager uses for each destination the values from the fields in the corresponding put message record. For fields that are absent, the queue manager uses the values from the MQMD structure.

MQPMRF_MSG_ID

```
public final static int
```

This flag indicates that a message-identifier field is present in the put message records provided by the application. This is only used when sending messages to a distribution list. For fields that are present, the queue manager uses for each destination the values from the fields in the corresponding put message record. For fields that are absent, the queue manager uses the values from the MQMD structure.

MQPMRF_NONE

```
public final static int
```

This flag indicates that no put message record fields are present. MQPMRF_NONE is defined to aid program documentation. It is not intended that this constant be used with any other, but as its value is zero, such use cannot be detected.

MQPRI_PRIORITY_AS_Q_DEF

```
public final static int
```

Priority is taken from the default priority attribute of the destination

MQQA_BACKOUT_HARDENED

```
public final static int
```

This option indicates that information is written to disk to ensure that the backout count for messages on this queue is accurate. This option imposes a performance overhead, so only use it if it is essential that the count is accurate.

MQQA_BACKOUT_NOT_HARDENED

```
public final static int
```

This option indicates that the backout count is not saved to disk. The count will survive queue manager restarts, but in the event of a queue manager failure the backout count might be lower than it should be.

MQQA_GET_ALLOWED

```
public final static int
```

This controls whether get operations for this queue are allowed. With this option, get operations are allowed.

MQQA_GET_INHIBITED

```
public final static int
```

This controls whether get operations for this queue are allowed. With this option, get operations are inhibited. MQGET calls fail with reason code MQRC_GET_INHIBITED. This includes MQGET calls that specify MQGMO_BROWSE_FIRST or MQGMO_BROWSE_NEXT.

MQQA_NOT_SHAREABLE

```
public final static int
```

This indicates whether the queue can be opened for input multiple times concurrently. With this option, the queue is not shareable.

MQQA_PUT_ALLOWED

```
public final static int
```

This controls whether put operations for this queue are allowed. With this option, put operations are allowed.

MQQA_PUT_INHIBITED

```
public final static int
```

This controls whether put operations for this queue are allowed. With this option, put operations are inhibited. MQPUT and MQPUT1 calls fail with reason code MQRC_PUT_INHIBITED.

MQQA_SHAREABLE

```
public final static int
```

This indicates whether the queue can be opened for input multiple times concurrently. With this option, the queue is shareable. Multiple opens with the MQOO_INPUT_SHARED option are allowed.

MQQDT_PERMANENT_DYNAMIC

```
public final static int
```

The queue is a permanent queue that was created by an application issuing an MQOPEN call with the name of a model queue specified in the object descriptor MQOD. The model queue definition had the value MQQDT_PERMANENT_DYNAMIC for the DefinitionType attribute.

MQQDT_PREDEFINED

```
public final static int
```

The queue is a permanent queue created by the system administrator. Only the system administrator can delete it. Predefined queues are created using the DEFINE MQSC command, and can be deleted only by using the DELETE MQSC command. Predefined queues cannot be created from model queues.

MQQDT_TEMPORARY_DYNAMIC

```
public final static int
```

The queue is a temporary queue that was created by an application issuing an MQOPEN call with the name of a model queue specified in the object descriptor MQOD. The model queue definition had the value MQQDT_TEMPORARY_DYNAMIC for the DefinitionType attribute.

This type of queue is deleted automatically by the MQCLOSE call when it is closed by the application that created it.

MQQT_ALIAS

```
public final static int
```

This defines an alias queue. This is not a physical queue; it is an alternative name for a local queue, a shared queue, a cluster queue, or a remote queue. The name of the queue to which the alias resolves is part of the definition of the alias queue.

MQQT_CLUSTER

```
public final static int
```

This defines a cluster queue. This is a physical queue that stores messages. The queue exists either on the local queue manager, or on one or more of the queue managers that belong to the same cluster as the local queue manager.

MQQT_LOCAL

```
public final static int
```

This defines a local queue. It is a physical queue that stores messages. The queue exists on the local queue manager. Applications connected to the local queue manager can place messages on and remove messages from queues of this type.

MQQT_MODEL
```
public final static int
```

This defines a model queue. It is not a physical queue; it is a set of queue attributes from which a local queue can be created. Messages cannot be stored on queues of this type.

MQQT_REMOTE
```
public final static int
```

This defines a remote queue. This is not a physical queue; it is the local definition of a queue that exists on a remote queue manager. The local definition of the remote queue contains information that tells the local queue manager how to route messages to the remote queue manager.

Applications connected to the local queue manager can place messages on queues of this type; the messages are placed on the local transmission queue used to route messages to the remote queue manager. Applications cannot remove messages from remote queues.

MQRFH_NO_FLAGS
```
public final static int
```

This defines an RFH flags field containing no flags.

MQRFH_STRUC_ID
```
public final static java.lang.String
```

This field is the identifier for the rules and formatting header structure.

MQRFH_STRUC_LENGTH_FIXED_1
```
public final static int
```

This defines the length of the fixed length part of a version 1 rules and formatting header structure. The length is 32 bytes.

MQRFH_STRUC_LENGTH_FIXED_2
```
public final static int
```

This defines the length of the fixed length part of a version 2 rules and formatting header structure. The length is 36 bytes.

MQRFH_VERSION_1
```
public final static int
```

This defines a version 1 rules and formatting header structure.

MQRFH_VERSION_2
```
public final static int
```

This defines a version 2 rules and formatting header structure.

MQRL_UNDEFINED
```
public final static int
```

The ReturnedLength field in the MQGMO is set by the queue manager to the length in bytes of the message data returned by the MQGET call in the Buffer parameter. If the queue manager does not support this capability, ReturnedLength is set to the value MQRL_UNDEFINED.

MQRO_ACCEPT_UNSUP_IF_XMIT_MASK

```
public final static int
```

This mask identifies the bit positions within the Report field where report options that are not supported by the local queue manager are nevertheless accepted on the MQPUT or MQPUT1 calls provided that both of the following conditions are satisfied:

- The message is destined for a remote queue manager.
- The application is not putting the message directly on a local transmission queue.

MQRO_ACCEPT_UNSUP_MASK

```
public final static int
```

This mask identifies the bit positions within the Report field where report options that are not supported by the local queue manager are nevertheless accepted on the MQPUT or MQPUT1 calls.

MQRO_ACTIVITY

```
public final static int
```

This type of report is generated by applications that are enabled for activity recording.

MQRO_COA

```
public final static int
```

This type of report is generated by the queue manager that owns the destination queue when the message is placed on the destination queue. Message data from the original message is not included with the report message.

If the message is put as part of a unit of work, and the destination queue is a local queue, the COA report message generated by the queue manager can be retrieved only if the unit of work is committed.

Do not specify more than one of MQRO_COA, MQRO_COA_WITH_DATA, and MQRO_COA_WITH_FULL_DATA.

MQRO_COA_WITH_DATA

```
public final static int
```

This is the same as MQRO_COA, except that the first 100 bytes of the application message data from the original message are included in the report message. If the original message contains one or more MQ header structures, they are included in the report message, in addition to the 100 bytes of application data.

Do not specify more than one of MQRO_COA, MQRO_COA_WITH_DATA, and MQRO_COA_WITH_FULL_DATA.

MQRO_COA_WITH_FULL_DATA

```
public final static int
```

This is the same as MQRO_COA, except that all the application message data from the original message is included in the report message.

Do not specify more than one of MQRO_COA, MQRO_COA_WITH_DATA, and MQRO_COA_WITH_FULL_DATA.

MQRO_COD

```
public final static int
```

This type of report is generated by the queue manager when an application retrieves the message from the destination queue in a way that deletes the message from the queue. Message data from the original message is not included with the report message.

If the message is retrieved as part of a unit of work, the report message is generated within the same unit of work, so that the report is not available until the unit of work is committed. If the unit of work is backed out, the report is not sent.

Do not specify more than one of MQRO_COD, MQRO_COD_WITH_DATA, and MQRO_COD_WITH_FULL_DATA.

MQRO_COD_WITH_DATA

```
public final static int
```

This is the same as MQRO_COD, except that the first 100 bytes of the application message data from the original message are included in the report message. If the original message contains one or more MQ header structures, they are included in the report message, in addition to the 100 bytes of application data.

Do not specify more than one of MQRO_COD, MQRO_COD_WITH_DATA, and MQRO_COD_WITH_FULL_DATA.

MQRO_COD_WITH_FULL_DATA

```
public final static int
```

This is the same as MQRO_COD, except that all the application message data from the original message is included in the report message.

Do not specify more than one of MQRO_COD, MQRO_COD_WITH_DATA, and MQRO_COD_WITH_FULL_DATA.

MQRO_COPY_MSG_ID_TO_CORREL_ID

```
public final static int
```

This is the default action, and indicates that if a report or reply is generated as a result of this message, the MsgId of this message is copied to the CorrelId of the report or reply message.

MQRO_DEAD_LETTER_Q

```
public final static int
```

This is the default action, and places the message on the dead-letter queue if the message cannot be delivered to the destination queue. An exception report message is generated, if one was requested by the sender.

MQRO_DISCARD_MSG

```
public final static int
```

This discards the message if it cannot be delivered to the destination queue. An exception report message is generated, if one was requested by the sender.

If you want to return the original message to the sender, without the original message being placed on the dead-letter queue, the sender must specify MQRO_DISCARD_MSG with MQRO_EXCEPTION_WITH_FULL_DATA.

MQRO_EXCEPTION

```
public final static int
```

A message channel agent generates this type of report when a message is sent to another queue manager and the message cannot be delivered to the specified destination queue. For example, the destination queue or an intermediate transmission queue might be full, or the message might be too big for the queue.

Do not specify more than one of MQRO_EXCEPTION, MQRO_EXCEPTION_WITH_DATA, and MQRO_EXCEPTION_WITH_FULL_DATA.

MQRO_EXCEPTION_WITH_DATA

```
public final static int
```

This is the same as MQRO_EXCEPTION, except that the first 100 bytes of the application message data from the original message are included in the report message. If the original message contains one or more MQ header structures, they are included in the report message, in addition to the 100 bytes of application data.

Do not specify more than one of MQRO_EXCEPTION, MQRO_EXCEPTION_WITH_DATA, and MQRO_EXCEPTION_WITH_FULL_DATA.

MQRO_EXCEPTION_WITH_FULL_DATA

```
public final static int
```

Exception reports with full data required. This is the same as MQRO_EXCEPTION, except that all the application message data from the original message is included in the report message.

Do not specify more than one of MQRO_EXCEPTION, MQRO_EXCEPTION_WITH_DATA, and MQRO_EXCEPTION_WITH_FULL_DATA.

MQRO_EXPIRATION

```
public final static int
```

This type of report is generated by the queue manager if the message is discarded before delivery to an application because its expiry time has passed. If this option is not set, no report message is generated if a message is discarded for this reason.

Do not specify more than one of MQRO_EXPIRATION, MQRO_EXPIRATION_WITH_DATA, and MQRO_EXPIRATION_WITH_FULL_DATA.

MQRO_EXPIRATION_WITH_DATA

```
public final static int
```

This is the same as MQRO_EXPIRATION, except that the first 100 bytes of the application message data from the original message are included in the report message. If the original message contains one or more MQ header structures, they are included in the report message, in addition to the 100 bytes of application data.

Do not specify more than one of MQRO_EXPIRATION, MQRO_EXPIRATION_WITH_DATA, and MQRO_EXPIRATION_WITH_FULL_DATA.

MQRO_EXPIRATION_WITH_FULL_DATA

```
public final static int
```

This is the same as MQRO_EXPIRATION, except that all the application message data from the original message is included in the report message.

Do not specify more than one of MQRO_EXPIRATION, MQRO_EXPIRATION_WITH_DATA, and MQRO_EXPIRATION_WITH_FULL_DATA.

MQRO_NAN

```
public final static int
```

This type of report is generated by the application that retrieves the message and acts upon it. It indicates that the action requested in the message has not been performed successfully. The application generating the report determines whether any data is to be included with the report.

MQRO_NEW_MSG_ID

```
public final static int
```

This is the default action, and indicates that if a report or reply is generated as a result of this message, a new MsgId is generated for the report or reply message.

MQRO_NONE

```
public final static int
```

Use this value to indicate that no other options have been specified. MQRO_NONE is defined to aid program documentation. It is not intended that this option be used with any other, but as its value is zero, such use cannot be detected.

MQRO_PAN

```
public final static int
```

This type of report is generated by the application that retrieves the message and acts upon it. It indicates that the action requested in the message has been performed successfully. The application generating the report determines whether any data is to be included with the report.

MQRO_PASS_CORREL_ID

```
public final static int
```

If a report or reply is generated as a result of this message, the CorrelId of this message is copied to the CorrelId of the report or reply message.

If this option is not specified, MQRO_COPY_MSG_ID_TO_CORREL_ID is assumed.

MQRO_PASS_DISCARD_AND_EXPIRY

```
public final static int
```

If this option is set on a message, and a report or reply is generated because of it, the message descriptor of the report inherits:

- MQRO_DISCARD_MSG if it was set.
- The remaining expiry time of the message (if this is not an expiry report). If this is an expiry report the expiry time is set to 60 seconds.

MQRO_PASS_MSG_ID

```
public final static int
```

If a report or reply is generated as a result of this message, the MsgId of this message is copied to the MsgId of the report or reply message.

If this option is not specified, MQRO_NEW_MSG_ID is assumed.

MQRO_REJECT_UNSUP_MASK

```
public final static int
```

This mask identifies the bit positions within the Report field where report options that are not supported by the local queue manager cause the MQPUT or MQPUT1 call to fail with completion code MQCC_FAILED and reason code MQRC_REPORT_OPTIONS_ERROR.

MQSCO_VERSION_1

```
public final static int
```

This defines a version 1 SSL configuration options structure.

MQSCO_VERSION_2

```
public final static int
```

This defines a version 2 SSL configuration options structure.

MQSEG_ALLOWED

```
public final static char
```

This is a flag that indicates that further segmentation is allowed for the message retrieved.

MQSEG_INHIBITED

```
public final static char
```

This is a flag that indicates that further segmentation is inhibited for the message retrieved.

MQSIDT_NONE

```
public final static byte
```

This indicates that no security identifier is present

MQSIDT_NT_SECURITY_ID

```
public final static byte
```

This indicates that a Windows security identifier is present.

MQSP_AVAILABLE

```
public final static int
```

This indicates that the local queue manager supports units of work and syncpointing with the MQGET, MQPUT, and MQPUT1 calls.

MQSP_NOT_AVAILABLE

```
public final static int
```

This indicates that the local queue manager does not support units of work and syncpointing with the MQGET, MQPUT, and MQPUT1 calls.

MQSS_LAST_SEGMENT

```
public final static char
```

This is a flag that indicates whether the message retrieved is the last segment of a logical message. This is also the value returned if the logical message consists of only one segment.

MQSS_NOT_A_SEGMENT

```
public final static char
```

This is a flag that indicates whether the message retrieved is not a segment of a logical message.

MQSS_SEGMENT

```
public final static char
```

This is a flag that indicates whether the message retrieved is a segment of a logical message.

MQTC_OFF

```
public final static int
```

This controls whether trigger messages are written to an initiation queue to start an application to service the queue. With this option, no trigger messages are to be written for this queue.

MQTC_ON

```
public final static int
```

This controls whether trigger messages are written to an initiation queue to start an application to service the queue. With this option, trigger messages are to be written for this queue when the appropriate trigger events occur.

MQTT_DEPTH

```
public final static int
```

This controls the conditions under which trigger messages are written as a result of messages arriving on this queue. With this option, a trigger message is written whenever the number of messages of priority TriggerMsgPriority or greater on the queue equals or exceeds TriggerDepth.

After the trigger message has been written, TriggerControl is set to MQTC_OFF to prevent further triggering until it is explicitly turned on again.

MQTT_EVERY

```
public final static int
```

This controls the conditions under which trigger messages are written as a result of messages arriving on this queue. With this option, a trigger message is written whenever a message of priority TriggerMsgPriority or greater arrives on the queue.

MQTT_FIRST

```
public final static int
```

This controls the conditions under which trigger messages are written as a result of messages arriving on this queue. With this option, a trigger message is written whenever the number of messages of priority TriggerMsgPriority or greater on the queue changes from 0 to 1.

MQTT_NONE

```
public final static int
```

This controls the conditions under which trigger messages are written as a result of messages arriving on this queue. With this option, no trigger messages are written as a result of messages on this queue. This has the same effect as setting TriggerControl to MQTC_OFF.

MQUS_NORMAL

```
public final static int
```

This indicates what the queue is used for. This value indicates that this is a queue that applications use when putting and getting messages; the queue is not a transmission queue.

MQUS_TRANSMISSION

```
public final static int
```

This indicates what the queue is used for. This value indicates that this is a queue used to hold messages destined for remote queue managers. When an application sends a message to a remote queue, the local queue manager stores the message temporarily on the appropriate transmission queue.

MQWI_UNLIMITED

```
public final static int
```

This option indicates that the MQGET call can wait an unlimited time for a suitable message to arrive.

MQXCC_CLOSE_CHANNEL

```
public final static int
```

This value can be set by any type of channel exit, and indicates that the connection to the queue manager can be closed.

MQXCC_OK

```
public final static int
```

This is set by the exit to indicate that the exit completed successfully. For a channel security exit, this indicates that message transfer can now proceed normally. In the case of a send exit, it indicates that the returned data is to be transmitted to the queue manager, while in the case of a receive exit, it indicates that the returned data is available for processing by the WebSphere MQ Client for Java.

MQXCC_SEND_AND_REQUEST_SEC_MSG

```
public final static int
```

This is set by the security exit to indicate that the returned data is to be transmitted to the queue manager, and that a response is expected. If no response is received, the channel must be terminated, because the exit has not yet decided whether communications can proceed. It is not valid for send or receive exits.

MQXCC_SEND_SEC_MSG

```
public final static int
```

This is set by the security exit to indicate that the returned data is to be transmitted to the queue manager. No response is expected. It is not valid for send or receive exits.

MQXCC_SUPPRESS_EXIT

```
public final static int
```

This value can be set by a send exit or receive exit, to indicate that it can no longer be called. It suppresses any further invocation of that exit, until termination of the channel, when the exit is again invoked with an exit reason of MQXR_TERM.

MQXCC_SUPPRESS_FUNCTION

```
public final static int
```

This is set by the security exit to indicate that communications with the queue manager must be shut down. It is not valid for send or receive exits.

MQXR_INIT

```
public final static int
```

This indicates that the exit is being invoked for the first time. It allows the exit to acquire and initialize any resources that it might need

This is set after the channel connection conditions have been negotiated, but before any security flows have been sent.

MQXR_INIT_SEC

```
public final static int
```

This indicates that the exit is to initiate the security dialog with the queue manager. This occurs for channel security exits only.

The receiver's security exit is always invoked with this reason immediately after being invoked with MQC.MQXR_INIT, to give it the opportunity to initiate a security exchange. If it declines the opportunity by returning MQC.MQXCC_OK

instead of MQC.MQXCC_SEND_SEC_MSG or MQC.MQXCC_SEND_AND_REQUEST_SEC_MSG , the sender's security exit is invoked with MQXR_INIT_SEC.

See *WebSphere MQ Intercommunication* for more details of the possible security exchanges that can take place when an exit is invoked with this reason.

MQXR_SEC_MSG

```
public final static int
```

This indicates that a security message has been received from the queue manager. This occurs for channel security exits only.

MQXR_SEC_PARMS

```
public final static int
```

This indicates that the exit might create a MQConnectionSecurityParameters object. If it does so, and MQChannelExit.getMQCSP() is not null after the exit completes, then the data returned from the exit is then sent to the server-connection end of the channel.

This occurs for channel security exits only, and takes place when the normal security message exchange has ended and the channel is ready to run.

See *WebSphere MQ Intercommunication* for more details of the possible security exchanges that can take place when an exit is invoked with this reason.

MQXR_TERM

```
public final static int
```

This indicates that the exit is about to be terminated. The exit should free any resources that it has acquired since it was initialized.

This is called after the disconnect flows have been sent but before the socket connection is destroyed.

MQXR_XMIT

```
public final static int
```

This indicates that the exit is about to process a transmission. This occurs for channel send and receive exits only.

MQXT_CHANNEL_RCV_EXIT

```
public final static int
```

This indicates that a Channel receive exit is being called. It is set on entry to the exit routine.

MQXT_CHANNEL_SEC_EXIT

```
public final static int
```

This indicates that a Channel security exit is being called. It is set on entry to the exit routine.

MQXT_CHANNEL_SEND_EXIT

```
public final static int
```

This indicates that a Channel send exit is being called. It is set on entry to the exit routine.

MSGCOMPLIST_LENGTH

```
public final static int
```

The maximum length of the list of message compression techniques which can be set.

PASSWORD_PROPERTY

```
public final static java.lang.String
```

WebSphere MQ Java environment key for defining the password. The corresponding value must be a String. This property overrides MQEnvironment.password .

PORT_PROPERTY

```
public final static java.lang.String
```

WebSphere MQ Java environment key for defining the port number. The corresponding value must be an Integer. This property overrides MQEnvironment.port .

RECEIVE_EXIT_PROPERTY

```
public final static java.lang.String
```

WebSphere MQ Java environment key for defining a channel receive exit. The corresponding value must be an Object that implements com.ibm.mq.MQReceiveExit. This property overrides MQEnvironment.receiveExit.

SECURITY_EXIT_PROPERTY

```
public final static java.lang.String
```

WebSphere MQ Java environment key for defining a channel security exit. The corresponding value must be an Object that implements com.ibm.mq.MQSecurityExit. This property overrides MQEnvironment.securirtyExit.

SEND_EXIT_PROPERTY

```
public final static java.lang.String
```

WebSphere MQ Java environment key for defining a channel send exit. The corresponding value must be an Object that implements com.ibm.mq.MQSendExit. This property overrides MQEnvironment.sendExit .

SSL_CERT_STORE_PROPERTY

```
public final static java.lang.String
```

WebSphere MQ Java environment key for defining the SSL certificate store. The corresponding value must be a java.util.Collection or a java.security.cert.CertStore. This property overrides MQEnvironment.sslCertStores.

SSL_CIPHER_SUITE_PROPERTY

```
public final static java.lang.String
```

WebSphere MQ Java environment key for defining the name of the SSL cipher suite. The corresponding value must be a String. This property overrides MQEnvironment.sslCipherSuite.

SSL_FIPS_REQUIRED_PROPERTY

```
public final static java.lang.String
```

WebSphere MQ Java environment key for defining the SSL FIPS required flag. The corresponding value must be an Boolean. If this is set to true, then only FIPS-certified cipher suites will be used. This property overrides MQEnvironment.sslFipsRequired.

SSL_PEER_NAME_PROPERTY

```
public final static java.lang.String
```

WebSphere MQ Java environment key for defining the SSL peer name. The corresponding value must be a String. This property overrides MQEnvironment.sslPeerName .

SSL_RESET_COUNT_PROPERTY

```
public final static java.lang.String
```

WebSphere MQ Java environment key for defining the SSL key reset count. The corresponding value must be an Integer, with a value between 0 (disabled) and 999,999,999. This property overrides MQEnvironment.sslResetCount .

SSL_SOCKET_FACTORY_PROPERTY

```
public final static java.lang.String
```

WebSphere MQ Java environment key for defining the SSL socket factory. The corresponding value must be a javax.net.ssl.SSLSocketFactory. This property overrides MQEnvironment.sslSocketFactory.

THREAD_AFFINITY

```
public final static java.lang.String
```

Deprecated

see THREAD_AFFINITY_PROPERTY.

THREAD_AFFINITY_PROPERTY

```
public final static java.lang.String
```

WebSphere MQ Java environment key for defining thread affinity. The corresponding value must be a Boolean. Thread affinity is disabled by default and connections can be shared. If it is enabled, then each queue manager connection will be bound to a worker thread.

Two-phase commit processing is not supported with shared connections, and so XA coordination is only possible when the MQQueueManager is created with THREAD_AFFINITY set to true. If it is not, MQQueueManager.begin() will fail with MQRC 2121, MQRC_NO_EXTERNAL_PARTICIPANTS.

TRANSPORT_MQSERIES

```
public final static java.lang.String
```

This value indicates that the mode of transport will be determined by the value of the hostname property. If this is not set, then the Java client will connect in Bindings mode, otherwise it will connect in Client mode.

TRANSPORT_MQSERIES_BINDINGS

```
public final static java.lang.String
```

This value indicates that the Java client will connect in Bindings mode.

TRANSPORT_MQSERIES_CLIENT

```
public final static java.lang.String
```

This value indicates that the Java client will connect in Client mode.

TRANSPORT_PROPERTY

```
public final static java.lang.String
```

WebSphere MQ Java environment key for defining the transport. The corresponding value must be an Integer, and must be one of MQC.TRANSPORT_MQSERIES_BINDINGS or MQC.TRANSPORT_MQSERIES_CLIENT. The default is MQC.TRANSPORT_MQSERIES, which selects a transport based on the value of the hostname property.

USER_ID_PROPERTY

```
public final static java.lang.String
```

WebSphere MQ Java environment key for defining the user ID. The corresponding value must be an String. This property overrides MQEnvironment.userID .

MQReceiveExit

public interface **MQReceiveExit**

`com.ibm.mq.MQReceiveExit`

The receive exit interface allows you to examine, and possibly alter, the data received from the queue manager by the WebSphere MQ Client for Java.

Note: This interface does not apply when connecting directly to WebSphere MQ in bindings mode.

To provide your own receive exit, define a class that implements this interface. Create a new instance of your class and assign it to the MQEnvironment.receiveExit field before constructing your MQQueueManager object.

For example,

```
// in MyReceiveExit.java
class MyReceiveExit implements MQReceiveExit
{
    // you must provide an implementation of the receiveExit method
    public byte[] receiveExit(MQChannelExit       channelExitParms,
                              MQChannelDefinition channelDefinition,
                              byte[]              agentBuffer)
    {
        // your exit code goes here...
    }
}

// in your main program...
MQEnvironment.receiveExit = new MyReceiveExit();
...    // other initialisation
MQQueueManager qMgr        = new MQQueueManager("");
```

Methods

receiveExit

```
public byte[] receiveExit(MQChannelExit channelExitParms,
                          MQChannelDefinition channelDefinition,
                          byte[] agentBuffer);
```

The receive exit method that your class must provide. It is invoked whenever the WebSphere MQ Client for Java receives a message from the queue manager.

Parameters

- channelExitParms - contains information about the context in which the exit is being invoked. channelExitParms.exitResponse is a parameter that you use to tell the WebSphere MQ Client for Java what action to take next.
- channelDefinition - contains details of the channel through which all communications with the queue manager take place.
- agentBuffer - contains the data received from the queue manager if channelExitParms.exitReason is MQChannelExit.MQXR_XMIT. Otherwise agentBuffer is null.

Returns

- the data to be processed. If the exit response code (in channelExitParms) is MQXCC_OK, the MQ Client for Java can now process the data. The simplest receive exit therefore, consists of the single line:

  ```
  return agentBuffer;
  ```

MQSecurityExit

public interface **MQSecurityExit**

`com.ibm.mq.MQSecurityExit`

The security exit interface allows you to customize the security flows that occur when an attempt is made to connect to a queue manager.

Note: This interface does not apply when connecting directly to WebSphere MQ in bindings mode.

To provide your own security exit, define a class that implements this interface. Create a new instance of your class and assign it to the MQEnvironment.securityExit field before constructing your MQQueueManager object.

For example,

```
// in MySecurityExit.java
class MySecurityExit implements MQSecurityExit
{
    // you must provide an implementation of the securityExit method
    public byte[] securityExit(MQChannelExit       channelExitParms,
                               MQChannelDefinition channelDefinition,
                               byte[]              agentBuffer)
    {
        // your exit code goes here...
    }
}

// in your main program...
MQEnvironment.securityExit = new MySecurityExit();
...    // other initialisation
MQQueueManager qMgr        = new MQQueueManager("");
```

Methods

securityExit

```
public byte[] securityExit(MQChannelExit channelExitParms,
                           MQChannelDefinition channelDefinition,
                           byte[] agentBuffer);
```

The security exit method that your class must provide.

Parameters

- channelExitParms - contains information about the context in which the exit is being invoked. channelExitParms.exitResponse is a parameter that you use to tell the WebSphere MQ Client for Java what action to take next.
- channelDefinition - Contains details of the channel through which all communications with the queue manager take place.
- agentBuffer - if channelExitParms.exitReason is MQChannelExit.MQXR_SEC_MSG, then agentBuffer contains the security message received from the queue manager, otherwise agentBuffer is null.

Returns

- the exit response code (in channelExitParms). If this is set so that a message is to be transmitted to the queue manager, then your security exit method must return the data to be transmitted.

MQSendExit

public interface **MQSendExit**

`com.ibm.mq.MQSendExit`

The send exit interface allows you to examine, and possibly alter, the data sent to the queue manager by the WebSphere MQ Client for Java.

Note: This interface does not apply when connecting directly to WebSphere MQ in bindings mode.

To provide your own send exit, define a class that implements this interface. Create a new instance of your class and assign it to the MQEnvironment.sendExit field before constructing your MQQueueManager object.

For example,

```
// in MySendExit.java
class MySendExit implements MQSendExit
{
    // you must provide an implementation of the sendExit method
    public byte[] sendExit(MQChannelExit       channelExitParms,
                           MQChannelDefinition channelDefinition,
                           byte[]              agentBuffer)
    {
        // your exit code goes here...
    }
}

// in your main program...
MQEnvironment.sendExit = new MySendExit();
...    // other initialisation
MQQueueManager qMgr        = new MQQueueManager("");
```

Methods

sendExit

```
public byte[] sendExit(MQChannelExit channelExitParms,
                       MQChannelDefinition channelDefinition,
                       byte[] agentBuffer);
```

The send exit method that your class must provide. It is invoked whenever the WebSphere MQ Client for Java sends a message to the queue manager.

Parameters

- channelExitParms - contains information about the context in which the exit is being invoked. channelExitParms.exitResponse is a parameter that you use to tell the WebSphere MQ Client for Java what action to take next.
- channelDefinition - contains details of the channel through which all communications with the queue manager take place.
- agentBuffer - contains the data received from the queue manager if channelExitParms.exitReason is MQChannelExit.MQXR_XMIT. Otherwise agentBuffer is null.

Returns

- the data to be processed. If the exit response code (in channelExitParms) is MQXCC_OK, the MQ Client for Java can now process the data. The simplest send exit therefore, consists of the single line:

```
return agentBuffer;
```

MQException

public class **MQException**
extends Exception

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----com.ibm.mq.MQException
```

An MQException is thrown whenever a WebSphere MQ error occurs. You can change the java.io.OutputStreamWriter to vary where exceptions are logged by setting the value of MQException.log. The default value is `System.err`.

Constructors

MQException

```
public MQException(int completionCode, int reasonCode, Object source);
```

Constructs a new MQException object.

Parameters

- completionCode - the WebSphere MQ completion code
- reasonCode - the WebSphere MQ reason code
- source - the object in which the error occurred

Fields

completionCode

```
public int
```

WebSphere MQ completion code giving rise to the error. The possible values are:

- MQException.MQCC_WARNING
- MQException.MQCC_FAILED

exceptionSource

```
public java.lang.Object
```

The object instance that threw the exception.

log

```
public static java.io.OutputStreamWriter
```

Stream to which exceptions will be logged (the default is System.err). If you set this to null then no logging will occur.

MQCC_FAILED

```
public final static int
```

Completion code - call failed. The processing of the call did not complete, and the state of the queue manager is normally unchanged; exceptions are specifically noted. The completion code and reason code output parameters have been set; other parameters are unchanged, except where noted. The failure might be the

result of a fault in the application program, or of a situation external to the program, for example the user's authority might have been revoked. The reason code parameter gives additional information about the error.

MQCC_OK
```
public final static int
```

Completion code - successful completion. The call completed fully; all output parameters have been set. The Reason parameter always has the value MQRC_NONE in this case.

MQCC_UNKNOWN
```
public final static int
```

Completion code - unknown.

MQCC_WARNING
```
public final static int
```

Completion code - warning (partial completion). The call completed partially. Some output parameters might have been set in addition to the completion code and reason code output parameters. The Reason parameter gives additional information about the partial completion.

MQRC_ADAPTER_CONN_LOAD_ERROR
```
public final static int
```

Reason code - unable to load adapter connection module.

MQRC_ADAPTER_CONV_LOAD_ERROR
```
public final static int
```

Reason code - unable to load data conversion services modules.

MQRC_ADAPTER_DEFS_ERROR
```
public final static int
```

Reason code - adapter subsystem definition module not valid.

MQRC_ADAPTER_DEFS_LOAD_ERROR
```
public final static int
```

Reason code - unable to load adapter subsystem definition module.

MQRC_ADAPTER_DISC_LOAD_ERROR
```
public final static int
```

Reason code - unable to load adapter disconnection module.

MQRC_ADAPTER_NOT_AVAILABLE
```
public final static int
```

Reason code - adapter not available.

MQRC_ADAPTER_SERV_LOAD_ERROR
```
public final static int
```

Reason code - unable to load adapter service module.

MQRC_ADAPTER_STORAGE_SHORTAGE
```
public final static int
```

Reason code - insufficient storage for adapter.

MQRC_ALIAS_BASE_Q_TYPE_ERROR
```
public final static int
```

Reason code - alias base queue not a valid type.

MQRC_ALREADY_CONNECTED
```
public final static int
```

Reason code - application is already connected.

MQRC_ANOTHER_Q_MGR_CONNECTED
```
public final static int
```

Reason code - another queue manager is already connected.

MQRC_API_EXIT_LOAD_ERROR
```
public final static int
```

Reason code - unable to load the API exit.

MQRC_ASID_MISMATCH
```
public final static int
```

Reason code - primary and home ASIDs differ.

MQRC_BACKED_OUT
```
public final static int
```

Reason code - unit of work is backed out.

MQRC_BACKOUT_THRESHOLD_REACHED
```
public final static int
```

Reason code - backout threshold has been reached.

MQRC_BUFFER_ERROR
```
public final static int
```

Reason code - buffer parameter is not valid.

MQRC_BUFFER_LENGTH_ERROR
```
public final static int
```

Reason code - buffer length parameter is not valid.

MQRC_CALL_IN_PROGRESS

```
public final static int
```

Reason code - MQI was entered before previous call was complete.

MQRC_CD_ERROR

```
public final static int
```

Reason code - invalid MQCD channel definition. An MQCONNX call was issued to connect to a queue manager, but the MQCD channel definition structure addressed by the ClientConnOffset or ClientConnPtr field in MQCNO contains data that is not valid.

MQRC_CF_NOT_AVAILABLE

```
public final static int
```

Reason code - the coupling-facility structure is unavailable. An MQOPEN or MQPUT1 call was issued to access a shared queue, but the allocation of the coupling-facility structure specified in the queue definition failed because there is no suitable coupling facility to hold the structure, based on the preference list in the active CFRM policy. Only applies to z/OS.

MQRC_CF_STRUC_AUTH_FAILED

```
public final static int
```

Reason code - the user is not authorized to access the coupling-facility structure. An MQOPEN or MQPUT1 call was issued to access a shared queue, but the call failed because the user is not authorized to access the coupling-facility structure specified in the queue definition. Only applies to z/OS.

MQRC_CF_STRUC_ERROR

```
public final static int
```

Reason code - coupling-facility structure is not valid. Z/OS only.

MQRC_CF_STRUC_IN_USE

```
public final static int
```

Reason code - the coupling-facility structure is in use. An MQI call was issued to operate on a shared queue, but the call failed because the coupling-facility structure specified in the queue definition is temporarily unavailable. Only applies to z/OS.

MQRC_CF_STRUC_LIST_HDR_IN_USE

```
public final static int
```

Reason code - the list header associated with the coupling-facility structure is in use. An MQGET, MQOPEN, MQPUT1, or MQSET call was issued to access a shared queue, but the call failed because the list header associated with the coupling-facility structure specified in the queue definition is temporarily unavailable. Only applies to z/OS.

MQRC_CFH_ERROR

```
public final static int
```

Reason code - PCF header structure is not valid.

MQRC_CFIL_ERROR
```
public final static int
```

Reason code - PCF integer list parameter structure is not valid.

MQRC_CFIN_ERROR
```
public final static int
```

Reason code - PCF integer parameter structure is not valid.

MQRC_CFSL_ERROR
```
public final static int
```

Reason code - PCF string list parameter structure is not valid.

MQRC_CFST_ERROR
```
public final static int
```

Reason code - PCF string parameter structure is not valid.

MQRC_CHANNEL_ACTIVATED
```
public final static int
```

Reason code - a channel is now able to become active because an active slot has been released by another channel.

MQRC_CHANNEL_AUTO_DEF_ERROR
```
public final static int
```

Reason code - automatically defined error

MQRC_CHANNEL_AUTO_DEF_OK
```
public final static int
```

Reason code - Automatic channel definition succeeded.

This condition is detected when the automatic definition of a channel is successful. The channel is defined by the MCA.

MQRC_CHANNEL_CONV_ERROR
```
public final static int
```

Reason code - a channel was unable to convert data and the MQGET call to get a message from the transmission queue resulted in a data conversion error.

MQRC_CHANNEL_NOT_ACTIVATED
```
public final static int
```

Reason code - a channel is required to become active, but is unable to do so because the limit on the number of active channels has been reached.

MQRC_CHANNEL_STARTED
```
public final static int
```

Reason code - either an operator has issued a Start Channel command or an instance of a channel has been successfully established.

MQRC_CHANNEL_STOPPED
```
public final static int
```

Reason code - channel has been stopped.

MQRC_CHANNEL_STOPPED_BY_USER
```
public final static int
```

Reason code - channel has been stopped by an operator.

MQRC_CHAR_ATTR_LENGTH_ERROR
```
public final static int
```

Reason code - length of character attributes is not valid.

MQRC_CHAR_ATTRS_ERROR
```
public final static int
```

Reason code - character attributes string is not valid.

MQRC_CHAR_ATTRS_TOO_SHORT
```
public final static int
```

Reason code - not enough space allowed for character attributes.

MQRC_CHAR_CONVERSION_ERROR
```
public final static int
```

Reason code - returned by the Java MQQueueManager constructor when a required character-set conversion is not available. The conversion required is between two non-Unicode character sets.

MQRC_CICS_WAIT_FAILED
```
public final static int
```

Reason code - wait request has been rejected by CICS.

MQRC_CLIENT_CHANNEL_CONFLICT
```
public final static int
```

Reason code - client channel definition table conflict error. An attempt is being made to specify a client channel definition table when the name of the channel has already been defined. Change the channel name to blank and try again. This reason code only occurs with Java applications.

MQRC_CLIENT_CONN_ERROR
```
public final static int
```

Reason code - an MQCONNX call was issued to connect to a queue manager, but the MQCD channel definition structure was not specified correctly.

MQRC_CLIENT_EXIT_ERROR
```
public final static int
```

Reason code - Error in client exit

MQRC_CLIENT_EXIT_LOAD_ERROR
```
public final static int
```

Reason code - client exit could not be loaded.

MQRC_CLUSTER_EXIT_ERROR
```
public final static int
```

Reason code - cluster workload exit has failed.

MQRC_CLUSTER_EXIT_LOAD_ERROR
```
public final static int
```

Reason code - unable to load cluster workload exit.

MQRC_CLUSTER_PUT_INHIBITED
```
public final static int
```

Reason code - put calls have been inhibited for all queues in cluster.

MQRC_CLUSTER_RESOLUTION_ERROR
```
public final static int
```

Reason code - cluster name resolution failed.

MQRC_CLUSTER_RESOURCE_ERROR
```
public final static int
```

Reason code - cluster resource error.

MQRC_CMD_SERVER_NOT_AVAILABLE
```
public final static int
```

Reason code - the command server that processes administration commands is not available.

MQRC_CNO_ERROR
```
public final static int
```

Reason code - connection options are not valid.

MQRC_COD_NOT_VALID_FOR_XCF_Q
```
public final static int
```

Reason code - COD report option not valid for XCF queue.

MQRC_CODED_CHAR_SET_ID_ERROR
```
public final static int
```

Reason code - the coded character set ID is not valid.

MQRC_CONN_ID_IN_USE
```
public final static int
```

Reason code - connection identifier is already in use.

MQRC_CONN_TAG_NOT_RELEASED
```
public final static int
```

Reason code - an MQDISC call was issued when there was a unit of work outstanding for the connection handle. Only applies to z/OS.

MQRC_CONN_TAG_NOT_USABLE
```
public final static int
```

Reason code - the connection tag is not usable. An MQCONNX call was issued specifying one of the MQCNO_* or _CONN_TAG_* options, but the call failed because the connection tag specified by ConnTag in MQCNO is being used by the queue manager for recovery processing, and this processing is delayed pending recovery of the coupling facility. Only applies to z/OS.

MQRC_CONNECTION_BROKEN
```
public final static int
```

Reason code - connection to queue manager has been lost.

MQRC_CONNECTION_NOT_AUTHORIZED
```
public final static int
```

Reason code - not authorized for connection.

MQRC_CONNECTION_QUIESCING
```
public final static int
```

Reason code - connection is quiescing.

MQRC_CONNECTION_STOPPING
```
public final static int
```

Reason code - connection is stopping.

MQRC_CONTEXT_HANDLE_ERROR
```
public final static int
```

Reason code - queue handle referred to does not save context.

MQRC_CONTEXT_NOT_AVAILABLE
```
public final static int
```

Reason code - context not available for queue handle referred to.

MQRC_CONVERTED_MSG_TOO_BIG
```
public final static int
```

Reason code - converted data is too big for buffer.

MQRC_CONVERTED_STRING_TOO_BIG
```
public final static int
```

Reason code - converted string is too big for field.

MQRC_CORREL_ID_ERROR
```
public final static int
```

Reason code - correlation ID error.

MQRC_CURRENT_RECORD_ERROR
```
public final static int
```

Reason code - error in current record.

MQRC_DATA_LENGTH_ERROR
```
public final static int
```

Reason code - data length parameter is not valid.

MQRC_DB2_NOT_AVAILABLE
```
public final static int
```

Reason code - an MQOPEN, MQPUT1, or MQSET call was issued to access a shared queue, but the call failed because the queue manager is not connected to a DB2 subsystem. As a result, the queue manager is unable to access the object definition relating to the shared queue. Only applies to z/OS.

MQRC_DBCS_ERROR
```
public final static int
```

Reason code - DBCS string is not valid.

MQRC_DEF_XMIT_Q_TYPE_ERROR
```
public final static int
```

Reason code - default transmission queue is not local.

MQRC_DEF_XMIT_Q_USAGE_ERROR
```
public final static int
```

Reason code - default transmission queue usage error.

MQRC_DLH_ERROR
```
public final static int
```

Reason code - dead letter header structure is not valid.

MQRC_DUPLICATE_RECOV_COORD
```
public final static int
```

Reason code - recovery coordinator already exists.

MQRC_DYNAMIC_Q_NAME_ERROR
```
public final static int
```

Reason code - name of dynamic queue is not valid.

MQRC_ENCODING_NOT_SUPPORTED
```
public final static int
```

Reason code - the Encoding field in the message descriptor MQMD contains a value that is not supported.

MQRC_ENVIRONMENT_ERROR

```
public final static int
```

Reason code - call not valid in this environment.

MQRC_EXPIRY_ERROR

```
public final static int
```

Reason code - expiry time is not valid.

MQRC_FEEDBACK_ERROR

```
public final static int
```

Reason code - feedback code is not valid.

MQRC_FILE_NOT_AUDITED

```
public final static int
```

Reason code - file has not been audited.

MQRC_FILE_SYSTEM_ERROR

```
public final static int
```

Reason code - error in file system.

MQRC_FORMAT_ERROR

```
public final static int
```

Reason code - message format is not valid.

MQRC_FORMAT_NOT_SUPPORTED

```
public final static int
```

Reason code - The format field in the message descriptor MQMD contains a value that is not supported.

MQRC_FUNCTION_NOT_SUPPORTED

```
public final static int
```

Reason code - An attempt was made to access functionality that is not supported from the environment from which it was called. For example, running the Client Configuration on z/OS

MQRC_GET_INHIBITED

```
public final static int
```

Reason code - gets are inhibited for the queue.

MQRC_GLOBAL_UOW_CONFLICT

```
public final static int
```

Reason code - global units of work conflict with each other.

MQRC_GMO_ERROR
```
public final static int
```

Reason code - get-message options object is not valid.

MQRC_GROUP_ID_ERROR
```
public final static int
```

Reason code - group identifier is not valid.

MQRC_HANDLE_IN_USE_FOR_UOW
```
public final static int
```

Reason code - handle is in use for a global unit of work.

MQRC_HANDLE_NOT_AVAILABLE
```
public final static int
```

Reason code - no more handles are available.

MQRC_HCONN_ERROR
```
public final static int
```

Reason code - connection handle is not valid.

MQRC_HEADER_ERROR
```
public final static int
```

Reason code - WebSphere MQ header structure is not valid.

MQRC_HOBJ_ERROR
```
public final static int
```

Reason code - object handle is not valid.

MQRC_INCOMPLETE_GROUP
```
public final static int
```

Reason code - message group is not complete.

MQRC_INCOMPLETE_MSG
```
public final static int
```

Reason code - message is not complete.

MQRC_INCONSISTENT_BROWSE
```
public final static int
```

Reason code - browse specification is inconsistent.

MQRC_INCONSISTENT_CCSIDS
```
public final static int
```

Reason code - message segments have differing CCSIDs.

MQRC_INCONSISTENT_ENCODINGS
```
public final static int
```

Reason code - message segments have differing encodings.

MQRC_INCONSISTENT_PERSISTENCE
```
public final static int
```

Reason code - inconsistent persistence specification.

MQRC_INCONSISTENT_UOW
```
public final static int
```

Reason code - unit-of-work specification is inconsistent.

MQRC_INDEX_ERROR
```
public final static int
```

Reason code - an index parameter to a call or method has a value that is not valid. The value must be zero or greater.

MQRC_INDEX_NOT_PRESENT
```
public final static int
```

Reason code - the specified index is not present.

MQRC_INHIBIT_VALUE_ERROR
```
public final static int
```

Reason code - value for inhibit-get or inhibit-put queue attribute is not valid.

MQRC_INT_ATTR_COUNT_ERROR
```
public final static int
```

Reason code - count of integer attributes is not valid.

MQRC_INT_ATTR_COUNT_TOO_SMALL
```
public final static int
```

Reason code - not enough space has been allowed for integer attributes.

MQRC_INT_ATTRS_ARRAY_ERROR
```
public final static int
```

Reason code - integer attributes array is not valid.

MQRC_INVALID_MSG_UNDER_CURSOR
```
public final static int
```

Reason code - message under cursor is not valid for retrieval.

MQRC_JMS_FORMAT_ERROR
```
public final static int
```

Reason code - this reason code is generated when JMS encounters a message that it is unable to parse. If such a message is encountered by a JMS

ConnectionConsumer, the message is processed as specified by the disposition options in the Report field in the MQMD of the message.

If the Report field specifies one of the MQRO_EXCEPTION_* options, this reason code appears in the Feedback field of the report message. If the Report field specifies MQRO_DEAD_LETTER_Q, or the disposition report options are left as default, this reason code appears in the Reason field of the MQDLH.

MQRC_JSSE_ERROR

```
public final static int
```

Reason code - JSSE reported an error (for example, while connecting to a queue manager using SSL encryption). The MQException object containing this reason code references the Exception thrown by JSSE; this can be obtained by using the MQException.getCause() method. From JMS, the MQException is linked to the thrown JMSException.

MQRC_LOCAL_UOW_CONFLICT

```
public final static int
```

Reason code - global unit of work conflicts with a local unit of work.

MQRC_MATCH_OPTIONS_ERROR

```
public final static int
```

Reason code - match options are not valid.

MQRC_MAX_CONNS_LIMIT_REACHED

```
public final static int
```

Reason code - maximum number of connections reached.

MQRC_MD_ERROR

```
public final static int
```

Reason code - message descriptor is not valid.

MQRC_MDE_ERROR

```
public final static int
```

Reason code - message descriptor extension is not valid.

MQRC_MISSING_REPLY_TO_Q

```
public final static int
```

Reason code - missing reply-to queue.

MQRC_MISSING_WIH

```
public final static int
```

Reason code - mismatch of queue IndexType and Format field in MQMD. An MQPUT or MQPUT1 call was issued to put a message on a queue whose IndexType attribute had the value MQIT_MSG_TOKEN, but the Format field in the MQMD was not MQFMT_WORK_INFO_HEADER. This error occurs only when the message arrives at the destination queue manager. Only allies to z/OS.

MQRC_MSG_FLAGS_ERROR
```
public final static int
```

Reason code - message flags are not valid.

MQRC_MSG_ID_ERROR
```
public final static int
```

Reason code - message ID error.

MQRC_MSG_MARKED_BROWSE_CO_OP
```
public final static int
```

Reason code - message is marked

MQRC_MSG_NOT_MATCHED
```
public final static int
```

Reason code - message is not matched.

MQRC_MSG_SEQ_NUMBER_ERROR
```
public final static int
```

Reason code - message sequence number is not valid.

MQRC_MSG_TOKEN_ERROR
```
public final static int
```

Reason code - An MQGET call was issued to retrieve a message using the message token as a selection criterion, but the options specified are not valid. only applies to z/OS.

MQRC_MSG_TOO_BIG_FOR_CHANNEL
```
public final static int
```

Reason code - message is too big for channel.

MQRC_MSG_TOO_BIG_FOR_Q
```
public final static int
```

Reason code - message length is greater than maximum for queue.

MQRC_MSG_TOO_BIG_FOR_Q_MGR
```
public final static int
```

Reason code - message length is greater than maximum for queue manager.

MQRC_MSG_TYPE_ERROR
```
public final static int
```

Reason code - message type in message descriptor is not valid.

MQRC_MULTIPLE_REASONS
```
public final static int
```

Reason code - multiple reason codes have been returned.

MQRC_NAME_IN_USE
```
public final static int
```

Reason code - an MQOPEN call was issued to create a dynamic queue, but a queue with the same name as the dynamic queue already exists. The existing queue is one that is logically deleted, but for which there are still one or more open handles. Only applies to z/OS.

MQRC_NAME_NOT_VALID_FOR_TYPE
```
public final static int
```

Reason code - object name is not valid for object type.

MQRC_NEXT_OFFSET_ERROR
```
public final static int
```

Reason code - error in offset to next record.

MQRC_NEXT_RECORD_ERROR
```
public final static int
```

Reason code - error in next record.

MQRC_NO_DESTINATIONS_AVAILABLE
```
public final static int
```

Reason code - no destination queues are available.

MQRC_NO_EXTERNAL_PARTICIPANTS
```
public final static int
```

Reason code - An MQBEGIN call was issued to start a unit of work coordinated by the queue manager, but no participating resource managers have been registered with the queue manager. As a result, only changes to WebSphere MQ resources can be coordinated by the queue manager in the unit of work.

MQRC_NO_MSG_AVAILABLE
```
public final static int
```

Reason code - no message is available.

MQRC_NO_MSG_LOCKED
```
public final static int
```

Reason code - no message is locked.

MQRC_NO_MSG_UNDER_CURSOR
```
public final static int
```

Reason code - browse cursor is not positioned on message.

MQRC_NO_RECORD_AVAILABLE
```
public final static int
```

Reason code - no record is available.

MQRC_NONE
```
public final static int
```

Reason code - none.

MQRC_NOT_AUTHORIZED
```
public final static int
```

Reason code - queue is not authorized for access.

MQRC_NOT_CONVERTED
```
public final static int
```

Reason code - message data has not been converted.

MQRC_NOT_OPEN_FOR_BROWSE
```
public final static int
```

Reason code - queue is not open for browse.

MQRC_NOT_OPEN_FOR_INPUT
```
public final static int
```

Reason code - queue is not open for input.

MQRC_NOT_OPEN_FOR_INQUIRE
```
public final static int
```

Reason code - queue is not open for inquire.

MQRC_NOT_OPEN_FOR_OUTPUT
```
public final static int
```

Reason code - queue is not open for output.

MQRC_NOT_OPEN_FOR_PASS_ALL
```
public final static int
```

Reason code - queue not open for pass all context.

MQRC_NOT_OPEN_FOR_PASS_IDENT
```
public final static int
```

Reason code - queue not open for pass identity context.

MQRC_NOT_OPEN_FOR_SET
```
public final static int
```

Reason code - queue is not open for set all context.

MQRC_NOT_OPEN_FOR_SET_ALL
```
public final static int
```

Reason code - queue not open for set all context.

MQRC_NOT_OPEN_FOR_SET_IDENT
```
public final static int
```

Reason code - queue not open for set identity context.

MQRC_OBJECT_ALREADY_EXISTS
```
public final static int
```

Reason code - object already exists.

MQRC_OBJECT_CHANGED
```
public final static int
```

Reason code - object definition has changed since it opened.

MQRC_OBJECT_DAMAGED
```
public final static int
```

Reason code - object has been damaged.

MQRC_OBJECT_IN_USE
```
public final static int
```

Reason code - object is already open with conflicting options.

MQRC_OBJECT_LEVEL_INCOMPATIBLE
```
public final static int
```

Reason code - object level is incompatible.

MQRC_OBJECT_NAME_ERROR
```
public final static int
```

Reason code - object name is not valid.

MQRC_OBJECT_NOT_UNIQUE
```
public final static int
```

Reason code - An MQOPEN or MQPUT1 call was issued to access a queue, but the call failed because the queue specified in the MQOD structure cannot be resolved unambiguously. There exists a shared queue with the specified name, and a nonshared queue with the same name. Only applies to z/OS.

MQRC_OBJECT_Q_MGR_NAME_ERROR
```
public final static int
```

Reason code - object queue-manager name is not valid.

MQRC_OBJECT_RECORDS_ERROR
```
public final static int
```

Reason code - object records are not valid.

MQRC_OBJECT_TYPE_ERROR
```
public final static int
```

Reason code - object type is not valid.

MQRC_OD_ERROR
```
public final static int
```

Reason code - object descriptor structure is not valid.

MQRC_OFFSET_ERROR
```
public final static int
```

Reason code - message segment offset is not valid.

MQRC_OPEN_FAILED
```
public final static int
```

Reason code - object did not open successfully.

MQRC_OPTION_ENVIRONMENT_ERROR
```
public final static int
```

Reason code - an MQGET call with the MQGMO_MARK_SKIP_BACKOUT option specified was issued from a DB2 Stored Procedure. The call failed because the MQGMO_MARK_SKIP_BACKOUT option cannot be used from a DB2 Stored Procedure.

MQRC_OPTION_NOT_VALID_FOR_TYPE
```
public final static int
```

Reason code - option is not valid for object type.

MQRC_OPTIONS_ERROR
```
public final static int
```

Reason code - options are not valid or not consistent.

MQRC_ORIGINAL_LENGTH_ERROR
```
public final static int
```

Reason code - original length is not valid.

MQRC_OUTCOME_MIXED
```
public final static int
```

Reason code - result of commit or backout operation is mixed.

MQRC_OUTCOME_PENDING
```
public final static int
```

Reason code - result of commit operation is pending.

MQRC_PAGESET_ERROR
```
public final static int
```

Reason code - error accessing page-set data set.

MQRC_PAGESET_FULL
```
public final static int
```

Reason code - external storage medium is full.

MQRC_PARTICIPANT_NOT_AVAILABLE
```
public final static int
```

Reason code - no participating resource managers have been registered.

MQRC_PARTICIPANT_NOT_DEFINED
```
public final static int
```

Reason code - participant has not been defined.

MQRC_PERSISTENCE_ERROR
```
public final static int
```

Reason code - persistence is not valid.

MQRC_PERSISTENT_NOT_ALLOWED
```
public final static int
```

Reason code - queue does not support persistent messages.

MQRC_PMO_ERROR
```
public final static int
```

Reason code - put-message options object is not valid.

MQRC_PMO_RECORD_FLAGS_ERROR
```
public final static int
```

Reason code - put message record flags not valid.

MQRC_PRIORITY_ERROR
```
public final static int
```

Reason code - message priority is not valid.

MQRC_PRIORITY_EXCEEDS_MAXIMUM
```
public final static int
```

Reason code - message priority exceeds maximum value supported.

MQRC_PUT_INHIBITED
```
public final static int
```

Reason code - put calls are inhibited for the queue.

MQRC_PUT_MSG_RECORDS_ERROR
```
public final static int
```

Reason code - put message records are not valid.

MQRC_Q_DELETED
```
public final static int
```

Reason code - queue has been deleted.

MQRC_Q_DEPTH_HIGH
```
public final static int
```

Reason code - Queue depth high limit reached or exceeded.

A message put has caused the queue depth to be incremented to or above the limit specified in the QDepthHighLimit attribute.

MQRC_Q_DEPTH_LOW
```
public final static int
```

Reason code - Queue depth low limit reached or exceeded.

A message get has caused the queue depth to be decremented to or below the limit specified in the QDepthLowLimit attribute.

MQRC_Q_FULL
```
public final static int
```

Reason code - queue is full.

MQRC_Q_MGR_ACTIVE
```
public final static int
```

Reason code - the queue manager is active.

MQRC_Q_MGR_NAME_ERROR
```
public final static int
```

Reason code - queue manager name is not valid or not known.

MQRC_Q_MGR_NOT_ACTIVE
```
public final static int
```

Reason code - the queue manager is not active.

MQRC_Q_MGR_NOT_AVAILABLE
```
public final static int
```

Reason code - queue manager is not available for connection.

MQRC_Q_MGR_QUIESCING
```
public final static int
```

Reason code - queue manager is quiescing.

MQRC_Q_MGR_STOPPING
```
public final static int
```

Reason code - queue manager is shutting down.

MQRC_Q_NOT_EMPTY
```
public final static int
```

Reason code - queue contains one or more messages or uncommitted put or get requests.

MQRC_Q_SERVICE_INTERVAL_HIGH
```
public final static int
```

Reason code - Queue service interval high.

No successful gets or puts have been detected within an interval greater than the limit specified in the QServiceInterval attribute.

MQRC_Q_SERVICE_INTERVAL_OK
```
public final static int
```

Reason code - Queue service interval OK.

A successful get has been detected within an interval less than or equal to the limit specified in the QServiceInterval attribute.

MQRC_Q_SPACE_NOT_AVAILABLE
```
public final static int
```

Reason code - no space available on disk for queue.

MQRC_Q_TYPE_ERROR
```
public final static int
```

Reason code - queue type not valid.

MQRC_RECS_PRESENT_ERROR
```
public final static int
```

Reason code - number of records present is not valid.

MQRC_REMOTE_Q_NAME_ERROR
```
public final static int
```

Reason code - remote queue name is not valid.

MQRC_REPORT_OPTIONS_ERROR
```
public final static int
```

Reason code - report options in message descriptor are not valid.

MQRC_RESOURCE_PROBLEM
```
public final static int
```

Reason code - insufficient system resources are available.

MQRC_RESPONSE_RECORDS_ERROR
```
public final static int
```

Reason code - object records not valid.

MQRC_RFH_COMMAND_ERROR
```
public final static int
```

Reason code - the message contains an MQRFH structure, but the command name contained in the NameValueString field is not valid.

MQRC_RFH_DUPLICATE_PARM
```
public final static int
```

Reason code - the message contains an MQRFH structure, but a parameter occurs more than once in the NameValueString field when only one occurrence is valid for the specified command.

MQRC_RFH_ERROR
```
public final static int
```

Reason code - an MQPUT or MQPUT1 call was issued, but the message data contains an MQRFH or MQRFH2 structure that is not valid.

MQRC_RFH_PARM_ERROR
```
public final static int
```

Reason code - the message contains an MQRFH structure, but a parameter name contained in the NameValueString field is not valid for the command specified.

MQRC_RFH_PARM_MISSING
```
public final static int
```

Reason code - the message contains an MQRFH structure, but the command specified in the NameValueString field requires a parameter that is not present.

MQRC_RFH_STRING_ERROR
```
public final static int
```

Reason code - the contents of the NameValueString field in the MQRFH structure are not valid.

MQRC_RMH_ERROR
```
public final static int
```

Reason code - Reference message header structure is not valid.

MQRC_SECOND_MARK_NOT_ALLOWED
```
public final static int
```

Reason code - a message is already marked.

MQRC_SECURITY_ERROR
```
public final static int
```

Reason code - a security error has occurred.

MQRC_SEGMENT_LENGTH_ZERO
```
public final static int
```

Reason code - length of data in message segment is zero.

MQRC_SIGNAL_OUTSTANDING
```
public final static int
```

Reason code - a signal is outstanding for this handle.

MQRC_SIGNAL_REQUEST_ACCEPTED
```
public final static int
```

Reason code - no message returned (but signal request accepted).

MQRC_SIGNAL1_ERROR
```
public final static int
```

Reason code - signal field is not valid.

MQRC_SOURCE_BUFFER_ERROR
```
public final static int
```

Reason code - source buffer parameter is not valid.

MQRC_SOURCE_CCSID_ERROR
```
public final static int
```

Reason code - source coded character set identifier is not valid.

MQRC_SOURCE_DECIMAL_ENC_ERROR
```
public final static int
```

Reason code - packed-decimal encoding specified by receiver is not recognized.

MQRC_SOURCE_FLOAT_ENC_ERROR
```
public final static int
```

Reason code - source floating point encoding specified is not recognized.

MQRC_SOURCE_INTEGER_ENC_ERROR
```
public final static int
```

Reason code - source integer encoding not recognized.

MQRC_SOURCE_LENGTH_ERROR
```
public final static int
```

Reason code - source length parameter is not valid.

MQRC_SSL_CERT_STORE_ERROR
```
public final static int
```

Reason code - SSL CertStore error.

MQRC_SSL_CERTIFICATE_REVOKED
```
public final static int
```

Reason code - SSL certificate has been revoked.

MQRC_SSL_INITIALIZATION_ERROR
```
public final static int
```

Reason code - SSL initialization error.

MQRC_SSL_KEY_RESET_ERROR
```
public final static int
```

Reason code - SSL key reset error.

MQRC_SSL_NOT_ALLOWED
```
public final static int
```

Reason code - SSL is not allowed.

MQRC_SSL_PEER_NAME_ERROR
```
public final static int
```

Reason code - SSL error in peer name.

MQRC_SSL_PEER_NAME_MISMATCH
```
public final static int
```

Reason code - SSL peer name mismatch.

MQRC_STOPPED_BY_CLUSTER_EXIT
```
public final static int
```

Reason code - call has been rejected by cluster workload exit.

MQRC_STORAGE_CLASS_ERROR
```
public final static int
```

Reason code - storage class error.

MQRC_STORAGE_NOT_AVAILABLE
```
public final static int
```

Reason code - insufficient storage is available.

MQRC_STRING_ERROR
```
public final static int
```

Reason code - the string parameter is not valid.

MQRC_STRING_LENGTH_ERROR
```
public final static int
```

The StringLength parameter is not valid.

MQRC_STRING_TRUNCATED
```
public final static int
```

Reason code - the string returned by the call is too long to fit in the buffer provided. The string has been truncated to fit in the buffer.

MQRC_SUPPRESSED_BY_EXIT
```
public final static int
```

Reason code - call suppressed by exit program.

MQRC_SYNCPOINT_LIMIT_REACHED
```
public final static int
```

Reason code - no more messages can be handled within current unit of work.

MQRC_SYNCPOINT_NOT_AVAILABLE
```
public final static int
```

Reason code - syncpoint support is not available.

MQRC_TARGET_BUFFER_ERROR
```
public final static int
```

Reason code - target buffer parameter is not valid.

MQRC_TARGET_CCSID_ERROR
```
public final static int
```

Reason code - the coded character-set identifier to which the character data is to be converted is not valid or not supported.

MQRC_TARGET_DECIMAL_ENC_ERROR
```
public final static int
```

Reason code - packed-decimal encoding specified by the receiver is not recognized.

MQRC_TARGET_FLOAT_ENC_ERROR
```
public final static int
```

Reason code - floating-point encoding specified by the receiver is not recognized.

MQRC_TARGET_INTEGER_ENC_ERROR
```
public final static int
```

Reason code - target coded character set identifier is not valid.

MQRC_TARGET_LENGTH_ERROR
```
public final static int
```

Reason code - target length parameter is not valid.

MQRC_TM_ERROR
```
public final static int
```

Reason code - trigger message structure is not valid.

MQRC_TMC_ERROR
```
public final static int
```

Reason code - character trigger message structure is not valid.

MQRC_TRIGGER_CONTROL_ERROR
```
public final static int
```

Reason code - value for trigger-point attribute is not valid.

MQRC_TRIGGER_DEPTH_ERROR
```
public final static int
```

Reason code - value for trigger-depth attribute is not valid.

MQRC_TRIGGER_MSG_PRIORITY_ERR
```
public final static int
```

Reason code - value for trigger-message priority attribute is not valid.

MQRC_TRIGGER_TYPE_ERROR
```
public final static int
```

Reason code - value for trigger-type attribute is not valid.

MQRC_TRUNCATED_MSG_ACCEPTED
```
public final static int
```

Reason code - truncated message returned (processing completed).

MQRC_TRUNCATED_MSG_FAILED
```
public final static int
```

Reason code - truncated message returned (processing not completed).

MQRC_UCS2_CONVERSION_ERROR
```
public final static int
```

Reason code - returned by the Java MQQueueManager constructor when a required character-set conversion is not available. The conversion required is between the UCS-2 Unicode character set and the queue manager's character set. IBM-500 is used for the queue manager's character set if no specific value is available.

MQRC_UNEXPECTED_ERROR
```
public final static int
```

Reason code - an unexpected error has occurred.

MQRC_UNIT_OF_WORK_NOT_STARTED
```
public final static int
```

Reason code - the unit of work has not started.

MQRC_UNKNOWN_ALIAS_BASE_Q
```
public final static int
```

Reason code - unknown alias base queue.

MQRC_UNKNOWN_DEF_XMIT_Q
```
public final static int
```

Reason code - unknown default transmission queue.

MQRC_UNKNOWN_OBJECT_NAME
```
public final static int
```

Reason code - unknown object name.

MQRC_UNKNOWN_OBJECT_Q_MGR
```
public final static int
```

Reason code - unknown object queue manager.

MQRC_UNKNOWN_REMOTE_Q_MGR
```
public final static int
```

Reason code - unknown remote queue manager.

MQRC_UNKNOWN_REPORT_OPTION
```
public final static int
```

Reason code - one or more report options in the in message descriptor are not recognized.

MQRC_UNKNOWN_XMIT_Q
```
public final static int
```

Reason code - unknown transmission queue.

MQRC_UNSUPPORTED_CIPHER_SUITE
```
public final static int
```

Reason code - cipher suite is unsupported.

MQRC_UOW_CANCELED
```
public final static int
```

Reason code - an MQI call was issued, but the unit of work (TM/MP transaction) being used for the WebSphere MQ operation had been canceled.

MQRC_UOW_ENLISTMENT_ERROR
```
public final static int
```

Reason code - enlistment in a global unit of work failed.

MQRC_UOW_MIX_NOT_SUPPORTED
```
public final static int
```

Reason code - mixture of unit-of-work calls is not supported.

MQRC_UOW_NOT_AVAILABLE
```
public final static int
```

Reason code - unit of work is not available for the queue manager to use.

MQRC_WAIT_INTERVAL_ERROR
```
public final static int
```

Reason code - wait interval in MQGMO not valid.

MQRC_WIH_HEADER
```
public final static int
```

Reason code - an MQPUT or MQPUT1 call was issued, but the message data contains an MQWIH structure that is not valid.

MQRC_WRONG_GMO_VERSION
```
public final static int
```

Reason code - wrong version of MQGMO has been supplied.

MQRC_WRONG_MD_VERSION
```
public final static int
```

Reason code - wrong version of MQMD has been supplied.

MQRC_WXP_ERROR
```
public final static int
```

Reason code - WXP error.

MQRC_XMIT_Q_TYPE_ERROR
```
public final static int
```

Reason code - transmission queue not local.

MQRC_XMIT_Q_USAGE_ERROR
```
public final static int
```

Reason code - transmission queue not local.

MQRC_XQH_ERROR
```
public final static int
```

Reason code - transmission queue header structure is not valid.

MQRC_XWAIT_CANCELED
```
public final static int
```

Reason code - XA wait has been canceled.

MQRC_XWAIT_ERROR
```
public final static int
```

Reason code - XA wait error.

reasonCode
```
public int
```

WebSphere MQ reason code describing the error.

Methods

getMessage

```
public String getMessage();
```

Gets the message detail.

Returns

- the detail

logExclude

```
public static void logExclude(Integer avoidCode);
```

Adds an exception type to be kept out of the log.

Parameters

- avoidCode - the exception which is not to be logged.

logInclude

```
public static void logInclude(Integer includeCode);
```

Allows an exception type to be put in the log.

Parameters

- includeCode - the exception which is to be logged.

Part 4. Programming with WebSphere MQ JMS

Chapter 10. Writing WebSphere MQ JMS applications

This chapter provides information to help with writing WebSphere MQ JMS applications. It gives a brief introduction to the JMS model, and detailed information on programming some common tasks that application programs are likely to need to perform.

The JMS model

JMS defines a generic view of a message passing service. The generic JMS model is based around the following interfaces that are defined in Sun's `javax.jms` package:

Connection
Provides access to the underlying transport, and is used to create *Sessions*.

Session
Provides a context for producing and consuming messages, including the methods used to create *MessageProducers* and *MessageConsumers*.

MessageProducer
Used to send messages.

MessageConsumer
Used to receive messages.

A Connection is thread safe, but Sessions, MessageProducers, and MessageConsumers are not. The recommended strategy is to use one Session per application thread.

In WebSphere MQ terms:

Connection
Provides a scope for temporary queues. Also, it provides a place to hold the parameters that control how to connect to WebSphere MQ. Examples of these parameters are the name of the queue manager, and the name of the remote host if you use the WebSphere MQ Java client connectivity.

Session
Contains an `HCONN` and therefore defines a transactional scope.

MessageProducer and MessageConsumer
Contain an `HOBJ` that defines a particular queue for writing to or reading from.

Note that normal WebSphere MQ rules apply:

- Only a single operation can be in progress per HCONN at any given time. Therefore, the MessageProducers or MessageConsumers associated with a Session cannot be called concurrently. This is consistent with the JMS restriction of a single thread per Session.
- PUTs can use remote queues, but GETs can only be applied to queues on the local queue manager.

The generic JMS interfaces are subclassed into more specific versions for point-to-point and publish/subscribe behavior.

The point-to-point versions are:

- QueueConnection
- QueueSession
- QueueSender
- QueueReceiver

When using JMS, always write application programs that use only references to the interfaces in `javax.jms`. All vendor-specific information is encapsulated in implementations of:

- QueueConnectionFactory
- TopicConnectionFactory
- Queue
- Topic

These are known as *administered objects*, that is, objects that can be built using a vendor-supplied administration tool and stored in a JNDI namespace. A JMS application can retrieve these objects from the namespace and use them without needing to know which vendor provided the implementation.

Building a connection

Connections are not created directly, but are built using a connection factory. Factory objects can be stored in a JNDI namespace, insulating the JMS application from provider-specific information. Details of how to create and store factory objects are in Chapter 5, "Using the WebSphere MQ JMS administration tool," on page 35.

If you do not have a JNDI namespace available, see "Creating factories at runtime" on page 315.

Retrieving the factory from JNDI

To retrieve an object from a JNDI namespace, set up an initial context, as shown in this fragment taken from the `IVTRun` sample file:

```
import javax.jms.*;
import javax.naming.*;
import javax.naming.directory.*;
 .
 .
 .
  java.util.Hashtable environment = new java.util.Hashtable();
  environment.put(Context.INITIAL_CONTEXT_FACTORY, icf);
  environment.put(Context.PROVIDER_URL, url);
  Context ctx = new InitialDirContext( environment );
```

where:

icf defines a factory class for the initial context

url defines a context specific URL

For more details about JNDI usage, see Sun's JNDI documentation.

Note: Some combinations of the JNDI packages and LDAP service providers can result in an LDAP error 84. To resolve the problem, insert the following line before the call to `InitialDirContext`.

```
environment.put(Context.REFERRAL, "throw");
```

Once an initial context is obtained, objects are retrieved from the namespace by using the `lookup()` method. The following code retrieves a QueueConnectionFactory named `ivtQCF` from an LDAP-based namespace:

```
QueueConnectionFactory factory;
factory = (QueueConnectionFactory)ctx.lookup("cn=ivtQCF");
```

Using the factory to create a connection

The `createQueueConnection()` method on the factory object is used to create a Connection, as shown in the following code:

```
QueueConnection connection;
connection = factory.createQueueConnection();
```

Creating factories at runtime

If a JNDI namespace is not available, it is possible to create factory objects at runtime. However, using this method reduces the portability of the JMS application because it requires references to WebSphere MQ specific classes.

The following code creates a QueueConnectionFactory with all default settings:

```
factory = new com.ibm.mq.jms.MQQueueConnectionFactory();
```

(You can omit the `com.ibm.mq.jms.` prefix if you import the `com.ibm.mq.jms` package instead.)

A connection created from the above factory uses the Java bindings to connect to the default queue manager on the local machine. The set methods described in "MQConnectionFactory" on page 478 can be used to customize the factory with WebSphere MQ specific information.

The only way to create a TopicConnectionFactory object at runtime is to construct it using the MQTopicConnectionFactory constructor. For example:

```
MQTopicConnectionFactory fact = new MQTopicConnectionFactory();
```

This creates a default TopicConnectionFactory object with the bindings transportType and all other default settings.

It is possible to change the transportType for the TopicConnectionFactory using its setTransportType() method. For example:

```
fact.setTransportType(JMSC.MQJMS_TP_BINDINGS_MQ);     // Bindings mode
fact.setTransportType(JMSC.MQJMS_TP_CLIENT_MQ_TCPIP); // Client mode
fact.setTransportType(JMSC.MQJMS_TP_DIRECT_TCPIP);    // Direct TCP/IP mode
```

The full JMS TopicConnectionFactory interface has been implemented. Refer to "MQTopicConnectionFactory" on page 547 for more details. Note that certain combinations of property settings are not valid for TopicConnectionFactory objects. See "Properties" on page 42 for more details.

Starting the connection

The JMS specification defines that connections should be created in the *stopped* state. Until the connection starts, MessageConsumers that are associated with the connection cannot receive any messages. To start the connection, issue the following command:

```
connection.start();
```

Choosing client or bindings transport

WebSphere MQ JMS can communicate with WebSphere MQ using either client or bindings transport. (However, client transport is not supported on z/OS.) If you use the Java bindings, the JMS application and the WebSphere MQ queue manager must be located on the same machine. If you use the client, the queue manager can be on a different machine from the application.

The contents of the connection factory object determine which transport to use. Chapter 5, “Using the WebSphere MQ JMS administration tool,” on page 35 describes how to define a factory object for use with client or bindings transport.

The following code fragment illustrates how you can define the transport within an application:

```
String HOSTNAME = "machine1";
String QMGRNAME = "machine1.QM1";
String CHANNEL = "SYSTEM.DEF.SVRCONN";

factory = new MQQueueConnectionFactory();
factory.setTransportType(JMSC.MQJMS_TP_CLIENT_MQ_TCPIP);
factory.setQueueManager(QMGRNAME);
factory.setHostName(HOSTNAME);
factory.setChannel(CHANNEL);
```

When used in client mode, WebSphere MQ JMS does not access information stored in a qm.ini file, or the equivalent information stored in the Windows Registry. Entries in a qm.ini file, such as the KeepAlive entry, are therefore ignored.

Specifying a range of ports for client connections

When a WebSphere MQ JMS application attempts to connect to a WebSphere MQ queue manager in client mode, a firewall might allow only those connections that originate from specified ports or a range of ports. In this situation, you can use the LOCALADDRESS property of a QueueConnectionFactory or TopicConnectionFactory object to specify a port, or a range of ports, that the application can bind to.

You can set the LOCALADDRESS property by using the WebSphere MQ JMS administration tool, or by calling the setLocalAddress() method in an application. Here is an example of setting the property from within an application:

```
mqQueueConnectionFactory.setLocalAddress("9.20.0.1(2000,3000)");
```

When the application connects to a queue manager subsequently, the application binds to a local IP address and port number in the range 9.20.0.1(2000) to 9.20.0.1(3000).

In a system with more than one network interface, you can also use the LOCALADDRESS property to specify which network interface must be used for a connection.

For a direct connection to a broker, the LOCALADDRESS property is relevant only when multicast is used. In this case, you can use the property to specify which local network interface must be used for a connection, but the value of the property must not contain a port number, or a range of port numbers.

Connection errors might occur if you restrict the range of ports. If an error occurs, a JMSException is thrown with an embedded MQException that contains the WebSphere MQ reason code MQRC_Q_MGR_NOT_AVAILABLE and the following message:

```
Socket connection attempt refused due to LOCAL_ADDRESS_PROPERTY restrictions
```

An error might occur if all the ports in the specified range are in use, or if the specified IP address, host name, or port number is not valid (a negative port number, for example).

Because the WebSphere MQ JMS client might create connections other than those required by an application, always consider specifying a range of ports. In general, every session created by an application requires one port and the WebSphere MQ JMS client might require three or four additional ports. If a connection error does occur, increase the range of ports.

Connection pooling, which is used by default in WebSphere MQ JMS, might have an effect on the speed at which ports can be reused. As a result, a connection error might occur while ports are being freed.

Obtaining a session

Once a connection is made, use the `createQueueSession` method on the QueueConnection to obtain a session.

The method takes two parameters:

1. A boolean that determines whether the session is *transacted* or *non-transacted*.
2. A parameter that determines the *acknowledge* mode.

The simplest case is that of the non-transacted session with `AUTO_ACKNOWLEDGE`, as shown in the following code fragment:

```
QueueSession session;

boolean transacted = false;
session = connection.createQueueSession(transacted,
                                        Session.AUTO_ACKNOWLEDGE);
```

Note: A connection is thread safe, but sessions (and objects that are created from them) are not. The recommended practice for multithreaded applications is to use a separate session for each thread.

Sending a message

Messages are sent using a MessageProducer. For point-to-point this is a `QueueSender` that is created using the `createSender` method on QueueSession. A `QueueSender` is normally created for a specific queue, so that all messages sent using that sender are sent to the same destination. The destination is specified using a Queue object. Queue objects can be either created at runtime, or built and stored in a JNDI namespace.

Queue objects are retrieved from JNDI in the following way:

```
Queue ioQueue;
ioQueue = (Queue)ctx.lookup( qLookup );
```

WebSphere MQ JMS provides an implementation of Queue in `com.ibm.mq.jms.MQQueue`. It contains properties that control the details of

WebSphere MQ specific behavior, but in many cases it is possible to use the default values. JMS defines a standard way to specify the destination that minimizes the WebSphere MQ specific code in the application. This mechanism uses the `QueueSession.createQueue` method, which takes a `string` parameter describing the destination. The string itself is still in a vendor-specific format, but this is a more flexible approach than directly referring to the vendor classes.

WebSphere MQ JMS accepts two forms for the `string` parameter of `createQueue()`.

- The first is the name of the WebSphere MQ queue, as illustrated in the following fragment taken from the `IVTRun` program in the `samples` directory:

  ```
  public static final String QUEUE = "SYSTEM.DEFAULT.LOCAL.QUEUE" ;
   .
   .
   .
    ioQueue = session.createQueue( QUEUE );
  ```

- The second, and more powerful, form is based on *uniform resource identifiers* (URIs). This form allows you to specify remote queues (queues on a queue manager other than the one to which you are connected). It also allows you to set the other properties contained in a `com.ibm.mq.jms.MQQueue` object.

  The URI for a queue begins with the sequence `queue://`, followed by the name of the queue manager on which the queue resides. This is followed by a further `/`, the name of the queue, and optionally, a list of name-value pairs that set the remaining Queue properties. For example, the URI equivalent of the previous example is:

  ```
  ioQueue = session.createQueue("queue:///SYSTEM.DEFAULT.LOCAL.QUEUE");
  ```

  The name of the queue manager is omitted. This is interpreted as the queue manager to which the owning QueueConnection is connected at the time when the Queue object is used.

Note: When sending a message to a cluster, leave the Queue Manager field in the JMS Queue object blank. This enables an MQOPEN to be performed in BIND_NOT_FIXED mode, which allows the queue manager to be determined. Otherwise an exception is returned reporting that the queue object cannot be found. This applies when using JNDI or defining queues at runtime.

The following example connects to queue Q1 on queue manager HOST1.QM1, and causes all messages to be sent as non-persistent and priority 5:

```
ioQueue = session.createQueue("queue://HOST1.QM1/Q1?persistence=1&priority=5");
```

The following is an example of creating a topic URI:

```
session.createTopic("topic://Sport/Football/Results?multicast=7");
```

Table 14 lists the names that can be used in the name-value part of the URI. A disadvantage of this format is that it does not support symbolic names for the values, so where appropriate, the table also indicates *special* values, which might change. (See “Setting properties with the set method” on page 319 for an alternative way of setting properties.)

Table 14. Property names for queue and topic URIs

Property	Description	Values
CCSID	Character set of the destination	integers - valid values listed in base WebSphere MQ documentation

Table 14. Property names for queue and topic URIs (continued)

Property	Description	Values
encoding	How to represent numeric fields	An integer value as described in the base WebSphere MQ documentation
expiry	Lifetime of the message in milliseconds	0 for unlimited, positive integers for timeout (ms)
multicast	Sets multicast mode for direct connections	-1=ASCF, 0=DISABLED, 3=NOTR, 5=RELIABLE, 7=ENABLED
persistence	Whether the message should be *hardened* to disk	1=non-persistent, 2=persistent, -1=QDEF, -2=APP
priority	Priority of the message	0 through 9, -1=QDEF, -2=APP
targetClient	Whether the receiving application is JMS compliant	0=JMS, 1=MQ
The special values are: **QDEF** Determine the property from the configuration of the WebSphere MQ queue. **APP** The JMS application can control this property.		

Once the Queue object is obtained (either using `createQueue` as above or from JNDI), it must be passed into the `createSender` method to create a QueueSender:

```
QueueSender queueSender = session.createSender(ioQueue);
```

The resulting `queueSender` object is used to send messages by using the `send` method:

```
queueSender.send(outMessage);
```

Note: If an application sends a message within a transaction, the message is not delivered to its destination until the transaction is committed. This means that an application cannot send a message and receive a reply to the message within the same transaction.

Setting properties with the set method

You can set Queue properties by first creating an instance of `com.ibm.mq.jms.MQQueue` using the default constructor. Then you can fill in the required values by using public `set` methods. This method means that you can use symbolic names for the property values. However, because these values are vendor-specific, and are embedded in the code, the applications become less portable.

The following code fragment shows the setting of a queue property with a `set` method.

```
com.ibm.mq.jms.MQQueue q1 = new com.ibm.mq.jms.MQQueue();
      q1.setBaseQueueManagerName("HOST1.QM1");
      q1.setBaseQueueName("Q1");
      q1.setPersistence(DeliveryMode.NON_PERSISTENT);
      q1.setPriority(5);
```

Table 15 on page 320 shows the symbolic property values that are supplied with WebSphere MQ JMS for use with the `set` methods.

Table 15. Symbolic values for queue properties

Property	Admin tool keyword	Values
expiry	UNLIM APP	JMSC.MQJMS_EXP_UNLIMITED JMSC.MQJMS_EXP_APP
priority	APP QDEF	JMSC.MQJMS_PRI_APP JMSC.MQJMS_PRI_QDEF
persistence	APP QDEF PERS NON	JMSC.MQJMS_PER_APP JMSC.MQJMS_PER_QDEF JMSC.MQJMS_PER_PER JMSC.MQJMS_PER_NON
targetClient	JMS MQ	JMSC.MQJMS_CLIENT_JMS_COMPLIANT JMSC.MQJMS_CLIENT_NONJMS_MQ
encoding	Integer(N) Integer(R) Decimal(N) Decimal(R) Float(N) Float(R) Native	JMSC.MQJMS_ENCODING_INTEGER_NORMAL JMSC.MQJMS_ENCODING_INTEGER_REVERSED JMSC.MQJMS_ENCODING_DECIMAL_NORMAL JMSC.MQJMS_ENCODING_DECIMAL_REVERSED JMSC.MQJMS_ENCODING_FLOAT_IEEE_NORMAL JMSC.MQJMS_ENCODING_FLOAT_IEEE_REVERSED JMSC.MQJMS_ENCODING_NATIVE
multicast	ASCF DISABLED NOTR RELIABLE ENABLED	JMSC.MQJMS_MULTICAST_AS_CF JMSC.MQJMS_MULTICAST_DISABLED JMSC.MQJMS_MULTICAST_NOT_RELIABLE JMSC.MQJMS_MULTICAST_RELIABLE JMSC.MQJMS_MULTICAST_ENABLED

See “The ENCODING property” on page 58 for a discussion of encoding.

Message types

JMS provides several message types, each of which embodies some knowledge of its content. To avoid referring to the vendor-specific class names for the message types, methods are provided on the Session object for message creation.

In the sample program, a text message is created in the following manner:

```
System.out.println( "Creating a TextMessage" );
TextMessage outMessage = session.createTextMessage();
System.out.println("Adding Text");
outMessage.setText(outString);
```

The message types that can be used are:

- BytesMessage
- MapMessage
- ObjectMessage
- StreamMessage
- TextMessage

Details of these types are in Chapter 15, “Package com.ibm.jms,” on page 421.

Receiving a message

Messages are received using a QueueReceiver. This is created from a Session by using the `createReceiver()` method. This method takes a Queue parameter that defines from where the messages are received. See "Sending a message" on page 317 for details of how to create a Queue object.

The sample program creates a receiver and reads back the test message with the following code:

```
QueueReceiver queueReceiver = session.createReceiver(ioQueue);
Message inMessage = queueReceiver.receive(1000);
```

The parameter in the receive call is a timeout in milliseconds. This parameter defines how long the method should wait if there is no message available immediately. You can omit this parameter, in which case, the call blocks indefinitely. If you do not want any delay, use the `receiveNoWait()` method.

The receive methods return a message of the appropriate type. For example, if a `TextMessage` is put on a queue, when the message is received the object that is returned is an instance of `TextMessage`.

To extract the content from the body of the message, it is necessary to cast from the generic Message class (which is the declared return type of the receive methods) to the more specific subclass, such as `TextMessage`. If the received message type is not known, you can use the `instanceof` operator to determine which type it is. It is good practice always to test the message class before casting, so that unexpected errors can be handled gracefully.

The following code illustrates the use of `instanceof`, and extraction of the content from a TextMessage:

```
if (inMessage instanceof TextMessage) {
  String replyString = ((TextMessage) inMessage).getText();
  .
  .
  .
} else {
  // Print error message if Message was not a TextMessage.
  System.out.println("Reply message was not a TextMessage");
}
```

Note: If an application sends a message within a transaction, the message is not delivered to its destination until the transaction is committed. This means that an application cannot send a message and receive a reply to the message within the same transaction.

Message selectors

JMS provides a mechanism to select a subset of the messages on a queue so that this subset is returned by a receive call. When creating a QueueReceiver, you can provide a string that contains an SQL (Structured Query Language) expression to determine which messages to retrieve. The selector can refer to fields in the JMS message header as well as fields in the message properties (these are effectively application-defined header fields). Details of the header field names, as well as the syntax for the SQL selector, are in Chapter 13, "JMS messages," on page 379.

The following example shows how to select for a user-defined property named `myProp`:

```
queueReceiver = session.createReceiver(ioQueue, "myProp = 'blue'");
```

Note: The JMS specification does not permit the selector associated with a receiver to be changed. Once a receiver is created, the selector is fixed for the lifetime of that receiver. This means that, if you require different selectors, you must create new receivers.

Asynchronous delivery

An alternative to making calls to `QueueReceiver.receive()` is to register a method that is called automatically when a suitable message is available. The following fragment illustrates the mechanism:

```
import javax.jms.*;

public class MyClass implements MessageListener
{
  // The method that will be called by JMS when a message
  // is available.
  public void onMessage(Message message)
  {
    System.out.println("message is "+message);

    // application specific processing here
    .
    .
    .
  }
}

 .
 .
 .
  // In Main program (possibly of some other class)
  MyClass listener = new MyClass();
  queueReceiver.setMessageListener(listener);

  // main program can now continue with other application specific
  // behavior.
```

Note: Use of asynchronous delivery with a QueueReceiver marks the entire Session as asynchronous. It is an error to make an explicit call to the `receive` methods of a QueueReceiver that is associated with a Session that is using asynchronous delivery.

Closing down

Garbage collection alone cannot release all WebSphere MQ resources in a timely manner, especially if the application needs to create many short-lived JMS objects at the Session level or lower. It is therefore important to call the `close()` methods of the various classes (QueueConnection, QueueSession, QueueSender, and QueueReceiver) when the resources are no longer required.

Java Virtual Machine hangs at shutdown

If an application using WebSphere MQ JMS finishes without calling `Connection.close()`, some JVMs appear to hang. If this problem occurs, either edit the application to include a call to `Connection.close()`, or terminate the JVM using the `Ctrl-C` keys.

Handling errors

Any runtime errors in a JMS application are reported by exceptions. The majority of methods in JMS throw JMSExceptions to indicate errors. It is good programming practice to catch these exceptions and display them on a suitable output.

A JMSException can contain a further exception embedded in it. For JMS, this can be a valuable way to pass important detail from the underlying transport. In the case of WebSphere MQ JMS, when WebSphere MQ raises an MQException, this exception is usually included as the embedded exception in a JMSException.

The implementation of JMSException does not include the embedded exception in the output of its `toString()` method. Therefore, it is necessary to check explicitly for an embedded exception and print it out, as shown in the following fragment:

```
try {
  .
  . code which may throw a JMSException
  .
} catch (JMSException je) {
  System.err.println("caught "+je);
  Exception e = je.getLinkedException();
  if (e != null) {
    System.err.println("linked exception: "+e);
  }
}
```

Exception listener

For asynchronous message delivery, the application code cannot catch exceptions raised by failures to receive messages. This is because the application code does not make explicit calls to `receive()` methods. To cope with this situation, it is possible to register an `ExceptionListener`, which is an instance of a class that implements the `onException()` method. When a serious error occurs, this method is called with the JMSException passed as its only parameter. Further details are in Sun's JMS documentation.

Using Secure Sockets Layer (SSL)

WebSphere MQ base Java client applications and WebSphere MQ JMS connections using TRANSPORT(CLIENT) support Secure Sockets Layer (SSL) encryption. SSL provides communication encryption, authentication, and message integrity. It is typically used to secure communications between any two peers on the Internet or within an intranet.

WebSphere MQ classes for Java uses Java Secure Socket Extension (JSSE) to handle SSL encryption, and so requires a JSSE provider. J2SE v1.4 JVMs have a JSSE provider built in. Details of how to manage and store certificates can vary from provider to provider. For information about this, refer to your JSSE provider's documentation.

This section assumes that your JSSE provider is correctly installed and configured, and that suitable certificates have been installed and made available to your JSSE provider.

SSL administrative properties

This section introduces the SSL administrative properties, as follows:

- "SSLCIPHERSUITE object property"
- "SSLPEERNAME object property"
- "SSLCERTSTORES object property" on page 325
- "SSLSocketFactory object property" on page 326

SSLCIPHERSUITE object property

To enable SSL encryption on a ConnectionFactory, use JMSAdmin to set the SSLCIPHERSUITE property to a CipherSuite supported by your JSSE provider. This must match the CipherSpec set on the target channel. However, CipherSuites are distinct from CipherSpecs and so have different names. Appendix D, "SSL CipherSpecs and CipherSuites," on page 645 contains a table mapping the CipherSpecs supported by WebSphere MQ to their equivalent CipherSuites as known to JSSE. Additionally, the named CipherSuite must be supported by your JSSE provider. For more information about CipherSpecs and CipherSuites with WebSphere MQ, see the *WebSphere MQ Security* book.

For example, to set a QueueConnectionFactory to connect to an SSL-enabled SVRCONN channel using a CipherSpec of RC4_MD5_EXPORT, issue the following command to JMSAdmin:

```
ALTER QCF(my.qcf) SSLCIPHERSUITE(SSL_RSA_EXPORT_WITH_RC4_40_MD5)
```

This can also be set from a program, using the setSSLCipherSuite() method on MQConnectionFactory.

For convenience, if a CipherSpec is specified on the SSLCIPHERSUITE property, JMSAdmin attempts to map the CipherSpec to an appropriate CipherSuite and issues a warning. This attempt to map is not made if the property is specified by a program.

SSLPEERNAME object property

A JMS application can ensure that it has connected to the correct queue manager, by specifying a distinguished name (DN) pattern. The connection succeeds only if the queue manager presents a DN that matches the pattern. For more details of the format of this pattern, refer to *WebSphere MQ Security* or the *WebSphere MQ Script (MQSC) Command Reference*.

The DN is set using the SSLPEERNAME property of ConnectionFactory. For example, the following JMSAdmin command sets the ConnectionFactory to expect the queue manager to identify itself with a Common Name beginning QMGR. with at least two Organizational Unit names, the first of which must be IBM and the second WEBSPHERE:

```
ALTER QCF(my.qcf) SSLPEERNAME(CN=QMGR.*, OU=IBM, OU=WEBSPHERE)
```

Checking is case-insensitive, and semicolons can be used in place of the commas. This can also be set from a program, using the setSSLPeerName() method on MQConnectionFactory. If this property is not set, no checking is performed on the Distinguished Name supplied by the queue manager. This property is ignored if no CipherSuite is set.

SSLCERTSTORES object property

It is common to use a certificate revocation list (CRL) to manage revocation of certificates that have become untrusted. These are typically hosted on LDAP servers; JMS allows an LDAP server to be specified for CRL checking under Java 2 v1.4 or later. The following JMSAdmin example directs JMS to use a CRL hosted on an LDAP server named crl1.ibm.com:

```
ALTER QCF(my.qcf) SSLCRL(ldap://crl1.ibm.com)
```

Note: To use a CertStore successfully with a CRL hosted on an LDAP server, make sure that your Java Software Development Kit (SDK) is compatible with the CRL. Some SDKs require that the CRL conforms to RFC 2587, which defines a schema for LDAP v2. Most LDAP v3 servers use RFC 2256 instead.

If your LDAP server is not running on the default port of 389, the port can be specified by appending a colon and the port number to the host name. If the certificate presented by the queue manager is present in the CRL hosted on crl1.ibm.com, the connection does not complete. To avoid single-point-of-failure, JMS allows multiple LDAP servers to be supplied, by supplying a space-delimited list of LDAP servers. For example:

```
ALTER QCF(my.qcf) SSLCRL(ldap://crl1.ibm.com ldap://crl2.ibm.com)
```

When multiple LDAP servers are specified, JMS tries each one in turn until it finds a server with which it can successfully verify the queue manager's certificate. Each server must contain identical information.

A string of this format can be supplied by a program on the MQConnectionFactory.setSSLCertStores() method. Alternatively, the application can create one or more java.security.cert.CertStore objects, place these in a suitable Collection object, and supply this Collection to the setSSLCertStores() method. In this way, the application can customize CRL checking. Refer to your JSSE documentation for details on constructing and using CertStore objects.

The certificate presented by the queue manager when a connection is being set up is validated as follows:

1. The first CertStore object in the Collection identified by sslCertStores is used to identify a CRL server.
2. An attempt is made to contact the CRL server.
3. If the attempt is successful, the server is searched for a match for the certificate.
   a. If the certificate is found to be revoked, the search process is over and the connection request fails with reason code MQRC_SSL_CERTIFICATE_REVOKED.
   b. If the certificate is not found, the search process is over and the connection is allowed to proceed.
4. If the attempt to contact the server is unsuccessful, the next CertStore object is used to identify a CRL server and the process repeats from step 2.

   If this was the last CertStore in the Collection, or if the Collection contains no CertStore objects, the search process has failed and the connection request fails with reason code MQRC_SSL_CERT_STORE_ERROR.

The Collection object determines the order in which CertStores are used.

If your application uses setSSLCertStores() to set a Collection of CertStore objects, the MQConnectionFactory can no longer be bound into a JNDI namespace.

Attempting to do so causes an exception. If the sslCertStores property is not set, no revocation checking is performed on the certificate provided by the queue manager. This property is ignored if no CipherSuite is set.

SSLSocketFactory object property

You might want to customize other aspects of the SSL connection for an application. For example, you might want to initialize cryptographic hardware or change the keystore and truststore in use. To do this, the application must first create a javax.net.ssl.SSLSocketFactory instance customized accordingly. Refer to your JSSE documentation for information on how to do this, as the customizable features vary from provider to provider. Once a suitable SSLSocketFactory has been obtained, use the MQConnectionFactory.setSSLSocketFactory() method to configure JMS to use the customized SSLSocketFactory.

If your application uses setSSLSocketFactory() to set a customized SSLSocketFactory, the MQConnectionFactory can no longer be bound into a JNDI namespace. Attempting to do so causes an exception. If this property is not set, the default SSLSocketFactory is used; refer to your JSSE documentation for details on the behavior of the default SSLSocketFactory. This property is ignored if no CipherSuite is set.

Important: Do not assume that use of the SSL properties ensures security when the ConnectionFactory is retrieved from a JNDI namespace that is not itself secure. Specifically, the standard LDAP implementation of JNDI is not secure; an attacker can imitate the LDAP server, misleading a JMS application into connecting to the wrong server without noticing. With suitable security arrangements in place, other implementations of JNDI (such as the fscontext implementation) are secure.

Chapter 11. Writing WebSphere MQ JMS publish/subscribe applications

You can write applications with WebSphere MQ JMS using two programming models:

- Point-to-point
- Publish/subscribe

This section considers publish/subscribe and how publish/subscribe messaging is implemented in WebSphere MQ JMS.

Introduction

With publish/subscribe messaging, one message producer can send messages to many message consumers at one time. The message producer need know nothing about the consumers receiving its messages, it needs to know only about the common destination. Similarly, the message consumers need to know only about the common destination. This common destination is called a *topic*.

A message producer that sends messages to a topic is a *publisher* and a message consumer that receives messages from a topic is a *subscriber*.

A message consumer receives messages on all topics to which it has subscribed. All messages sent to a topic are forwarded to all the message consumers subscribed to that topic at that time. Each consumer receives its own copy of each message.

JMS clients can establish durable subscriptions that allow consumers to disconnect and later reconnect and collect messages published while they were disconnected.

The connection between messages issued by publishers and the subscribers is made, in WebSphere MQ, by the publish/subscribe *broker*. The broker (sometimes referred to as the message broker) has a record of all the subscribers registered to a topic. When a message is published to a topic, the broker manages the forwarding of that message to the topic's subscribers.

To run a WebSphere MQ JMS publish/subscribe application, you must be able to connect to a message broker.

Getting started with WebSphere MQ JMS and publish/subscribe

Before you can start developing publish/subscribe applications, you need to choose the broker to use and set that broker up to run the WebSphere MQ JMS.

Choosing a broker

WebSphere MQ offers the following choice of brokers:

- WebSphere MQ Publish/Subscribe
- WebSphere MQ Integrator, Version 2 provides a broker that can be run in one of two modes. Compatibility mode, which provides a broker of equivalent functionality to the WebSphere MQ Publish/Subscribe broker; and native mode, which provides additional functionality. WebSphere MQ JMS can connect to

WebSphere MQ Integrator in native mode with JMS Version 5.2.1 and later. With earlier JMS versions, it can connect to WebSphere MQ Integrator in compatibility mode only.

Note, however, that broker based subscription stores are not supported by WebSphere MQ Integrator. For more information about subscription stores, see "Subscription stores" on page 363.

- WebSphere MQ Event Broker, Version 2.1, WebSphere Business Integration Event Broker, Version 5.0, and WebSphere Business Integration Message Broker, Version 5.0 each provide a broker that can be connected to in two different ways:

  Using message queues and WebSphere MQ
  With this connection, you can run the broker in either compatibility mode or native mode.

  Directly using a TCP/IP socket
  With this connection, you can run the broker only in native mode. Also there is no support for:
  – Persistent messages
  – Transacted messages
  – Durable subscriptions

  This has implications for the implementation of the JMS specification for direct connections to a broker:
  – Because there are no persistent messages, JMSDeliveryMode is always NON_PERSISTENT, and JMSExpiration has no meaning for messages received on direct connections.
  – Because there are no transacted messages, JMSRedelivered has no meaning for messages received on direct connections.

For specific information on each publish and subscribe interface, see Chapter 16, "Package com.ibm.mq.jms," on page 469.

Setting up the broker to run WebSphere MQ JMS

Broker setup depends on the broker you intend to use and how you intend to use it. Each broker provides its own documentation describing installation and setup. However, for convenience and because of WebSphere MQ JMS requirements, some setup instructions are given here.

Connecting to your broker using WebSphere MQ

This section applies to a WebSphere MQ Publish/Subscribe broker or a broker of WebSphere MQ Integrator. It also applies to a broker of WebSphere MQ Event Broker, WebSphere Business Integration Event Broker, or WebSphere Business Integration Message Broker if you choose to connect to the broker using WebSphere MQ.

Each broker requires its own queue manager. Refer to the broker's documentation regarding installation and setup.

For the WebSphere MQ JMS publish/subscribe implementation to work correctly, a number of system queues must be created on the queue manager on which the broker is running. Create these message queues on each queue manager for each broker you want to run WebSphere MQ JMS. WebSphere MQ JMS provides a script that creates these queues (see Create the WebSphere MQ JMS system queues).

Run the script to create the system queues. If you are using the WebSphere MQ Publish/Subscribe broker, your broker is now fully configured. To check that the

broker is correctly configured, run the publish/subscribe verification as described in "Publish/subscribe verification without JNDI" on page 30.

If you are using a broker of WebSphere MQ Integrator, WebSphere MQ Event Broker, WebSphere Business Integration Event Broker, or WebSphere Business Integration Message Broker, configure a publish/subscribe message flow in the broker for messages to be correctly routed. The method for creating the required message flow is similar in both cases. Refer to Appendix C, "Connecting to other products," on page 639 for details.

Connecting to your broker directly

This is possible only when you use the broker provided in WebSphere MQ Event Broker, WebSphere Business Integration Event Broker, or WebSphere Business Integration Message Broker. Because the connection to this broker is made directly, no system queues are required. However, you must set up a publish/subscribe message flow in the broker for messages to be correctly routed. Refer to Appendix C, "Connecting to other products," on page 639 for details.

Writing a simple publish/subscribe application connecting through WebSphere MQ

This section provides a walkthrough of a simple WebSphere MQ JMS application.

Here is the complete example. Individual sections are discussed after.

```
// ============================================================================
//
// Licensed Materials - Property of IBM
//
// 5724-H27, 5655-L82, 5724-L26
//
// (c) Copyright IBM Corp. 1995,2002,2005
//
// ============================================================================
//
// A TopicConnectionFactory object is retrieved from LDAP; this is used to
// create a TopicConnection. The TopicConnection is used to create a
// TopicSession, which creates two publishers and two subscribers. Both
// publishers subscribe to a topic; both subscribers then receive.
//
// @(#) jms/samples/PubSubSample.java, jms, j000 1.1 04/12/03 14:31:41

import java.util.Hashtable;

import javax.jms.*;
import javax.naming.*;
import javax.naming.directory.InitialDirContext;

public class PubSubSample {

    // To use LDAP the Initial Context Factory and the URL need to be specified
    // Change these to suit your particular installatoin
    static String icf = "com.sun.jndi.ldap.LdapCtxFactory";
    static String url = "ldap://edradour.hursley.ibm.com/cn=JMSData,dc=ibm,dc=uk";

    static String tcfLookup = "cn=ivtTCF"; // TopicConnectionFactory (TCF) lookup
    private static String tLookup = "cn=ivtT"; // topic lookup


    // Pub/Sub objects used by this program
    private static TopicConnectionFactory fact = null;
    private static Topic topic = null;
```

```
public static void main(String args[]) {
    // Initialise JNDI properties
    Hashtable env = new Hashtable();
    env.put(Context.INITIAL_CONTEXT_FACTORY, icf);
    env.put(Context.PROVIDER_URL, url);
    env.put(Context.REFERRAL, "throw");

    Context ctx = null;
    try {
        System.out.print("Initialising JNDI... ");
        ctx = new InitialDirContext(env);
        System.out.println("Done!");
    } catch (NamingException nx) {
        System.out.println("ERROR: " + nx);
        System.exit(-1);
    }

    // Lookup TCF
    try {
        System.out.print("Obtaining TCF from JNDI... ");
        fact = (TopicConnectionFactory) ctx.lookup(tcfLookup);
        System.out.println("Done!");
    } catch (NamingException nx) {
        System.out.println("ERROR: " + nx);
        System.exit(-1);
    }

    // Lookup Topic
    try {
        System.out.print("Obtaining topic T from JNDI... ");
        topic = (Topic) ctx.lookup(tLookup);
        System.out.println("Done!");
    } catch (NamingException nx) {
        System.out.println("ERROR: " + nx);
        System.exit(-1);
    }

    try {
        ctx.close();
    } catch (NamingException nx) {
        // Just ignore an exception on closing the context
    }

    try {
        // Create connection
        TopicConnection conn = fact.createTopicConnection();
        // Start connection
        conn.start();
        // Session
        TopicSession sess = conn.createTopicSession(false,
                Session.AUTO_ACKNOWLEDGE);

        // Create a topic dynamically
        Topic t = sess.createTopic("myTopic");
        // Publisher
        TopicPublisher pub = sess.createPublisher(t);
        // Subscriber
        TopicSubscriber sub = sess.createSubscriber(t);
        // Publisher
        TopicPublisher pubA = sess.createPublisher(topic);
        // Subscriber
        TopicSubscriber subA = sess.createSubscriber(topic);

        // Publish "Hello World"
        TextMessage hello = sess.createTextMessage();
        hello.setText("Hello World");
        pub.publish(hello);
```

```
            hello.setText("Hello World 2");
            pubA.publish(hello);

            // Receive message
            TextMessage m = (TextMessage) sub.receive();
            System.out.println("Message Text = " + m.getText());
            m = (TextMessage) subA.receive();
            System.out.println("Message Text = " + m.getText());

            // Close publishers and subscribers
            pub.close();
            pubA.close();
            sub.close();
            subA.close();

            // Close session and connection
            sess.close();
            conn.close();
            System.exit(0);
        } catch (JMSException je) {
            System.out.println("ERROR: " + je);
            System.out.println("LinkedException: " + je.getLinkedException());
            System.exit(-1);
        }
    }
}
```

Import required packages

The import statements for an application using WebSphere MQ classes for Java Message Service must include at least the following:

```
import javax.jms.*;                // JMS interfaces
import javax.naming.*;             // Used for JNDI lookup of
import javax.naming.directory.*;   //    administered objects
```

Obtain or create JMS objects

The next step is to obtain or create a number of JMS objects:

1. Obtain a TopicConnectionFactory
2. Create a TopicConnection
3. Create a TopicSession
4. Obtain a Topic from JNDI
5. Create TopicPublishers and TopicSubscribers

Many of these processes are similar to those that are used for point-to-point, as shown in the following:

Obtain a TopicConnectionFactory

The preferred way to do this is to use JNDI lookup, to maintain portability of the application code. The following code initializes a JNDI context:

```
String icf = "com.sun.jndi.ldap.LdapCtxFactory";  // initial context factory
String url = "ldap://server.company.com/o=company_us,c=us";  // url

// Initialise JNDI properties
Java.util.Hashtable env = new Hashtable();
env.put( Context.INITIAL_CONTEXT_FACTORY, icf );
env.put( Context.PROVIDER_URL, url );
env.put( Context.REFERRAL, "throw" );

Context ctx = null;
try {
      System.out.print( "Initialising JNDI... " );
```

```
      ctx = new InitialDirContext( env );
      System.out.println( "Done!" );
} catch ( NamingException nx ) {
      System.out.println( "ERROR: " + nx );
      System.exit(-1);
}
```

Note: Change the `icf` and `url` variables to suit your installation and your JNDI service provider.

The properties required by JNDI initialization are in a Hashtable, which is passed to the InitialDirContext constructor. If this connection fails, an exception is thrown to indicate that the administered objects required later in the application are not available.

Obtain a TopicConnectionFactory using a lookup key that the administrator has defined:

```
// LOOKUP TCF
try {
    System.out.print( "Obtaining TCF from JNDI... " );
    fact = (TopicConnectionFactory)ctx.lookup( tcfLookup );
    System.out.println( "Done!" );
} catch ( NamingException nx ) {
    System.out.println( "ERROR: " + nx );
    System.exit(-1);
}
```

If a JNDI namespace is not available, you can create a TopicConnectionFactory at runtime. You create a new com.ibm.mq.jms.MQTopicConnectionFactory as described in "Creating factories at runtime" on page 315.

Create a TopicConnection

This is created from the TopicConnectionFactory object. Connections are always initialized in a `stop` state and must be started with the following code:

```
// create connection
TopicConnection conn = fact.createTopicConnection();
//start connection
conn.start();
```

Create a TopicSession

This is created using the TopicConnection. This method takes two parameters: one to signify whether the session is transacted, and one to specify the acknowledgement mode:

```
TopicSession sess = conn.createTopicSession(false,
                                  Session.AUTO_ACKNOWLEDGE);
```

Obtain a Topic

This object can be obtained from JNDI, for use with TopicPublishers and TopicSubscribers that are created later. The following code retrieves a Topic:

```
Topic topic = null;
try {
    System.out.print( "Obtaining topic T from JNDI... " );
    topic = (Topic)ctx.lookup( tLookup );
    System.out.println( "Done!" );
}
catch ( NamingException nx ) {
    System.out.println( "ERROR: " + nx );
    System.exit(-1);
}
```

If a JNDI namespace is not available, you can create a Topic at runtime, as described in "Creating topics at runtime" on page 337.

The following code creates a Topic at runtime:

```
// topic
Topic t = sess.createTopic("myTopic");
```

Create consumers and producers of publications

Depending on the nature of the JMS client application that you write, a subscriber, a publisher, or both must be created. Use the `createPublisher` and `createSubscriber` methods as follows:

```
// publisher
TopicPublisher pub = sess.createPublisher(t);
// subscriber
TopicSubscriber sub = sess.createSubscriber(t);
// publisher
TopicPublisher pubA = sess.createPublisher(topic);
// subscriber
TopicSubscriber subA = sess.createSubscriber(topic);
```

Publish messages

The TopicPublisher object, `pub`, is used to publish messages, rather like a QueueSender is used in the point-to-point domain. The following fragment creates a TextMessage using the session, and then publishes the message:

```
// publish "hello world"
TextMessage  hello = sess.createTextMessage();
hello.setText("Hello World");
pub.publish(hello);
hello.setText("Hello World 2");
pubA.publish(hello);
```

Receive subscriptions

Subscribers must be able to read the subscriptions that are delivered to them, as in the following code:

```
// receive message
TextMessage m = (TextMessage) sub.receive();
System.out.println("Message Text = " + m.getText());
m = (TextMessage) subA.receive();
System.out.println("Message Text = " + m.getText());
```

This fragment of code performs a *get-with-wait*, which means that the receive call blocks until a message is available. Alternative versions of the receive call are available (such as receiveNoWait). For details, see "MQTopicSubscriber" on page 552.

Close down unwanted resources

It is important to free up all the resources used by the application when it terminates. Use the `close()` method on objects that can be closed (publishers, subscribers, sessions, and connections):

```
// close publishers and subscribers
pub.close();
pubA.close();
sub.close();
subA.close();
sess.close();
```

```
// close session and connection
sess.close();
conn.close();
```

TopicConnectionFactory administered objects

In the example, the TopicConnectionFactory object is obtained from JNDI name space. The TopicConnectionFactory in this case is an administered object that has been created and administered using the JMSAdmin tool. Use this method of obtaining TopicConnectionFactory objects because it ensures code portability.

The TopicConnectionFactory in the example is testTCF in JMSAdmin. Create testTCF in JMSAdmin **before** running the application. You must also create a Topic in JMSAdmin; see "Topic administered objects."

To create a TopicConnectionFactory object, invoke the JMSAdmin tool, as described in "Invoking the administration tool" on page 35, and execute one of the following commands, depending on the type of connection you want to make to the broker:

Bindings connection

```
InitCtx> def tcf(testTCF) transport(bind)
```

or, because this is the default transport type for TopicConnectionFactory objects:

```
InitCtx> def tcf(testTCF)
```

This creates a TopicConnectionFactory with default settings for bindings transport, connecting to the default queue manager.

Client connection

```
InitCtx> def tcf(testTCF) transport(client)
```

This creates a TopicConnectionFactory with default settings for the client transport type, connecting to localhost, on port 1414, using channel SYSTEM.DEF.SVRCONN.

Direct TCP/IP connection to a broker

```
InitCtx> def tcf(testTCF) transport(direct)
```

This creates a TopicConnectionFactory to make direct connections to a broker, connecting to localhost on port 1506.

Topic administered objects

In the example, one of the Topic objects has been obtained from JNDI name space. This Topic is an administered object that has been created and administered in the JMSAdmin tool. Use this method of obtaining Topic objects because it ensures code portability.

To run the example application above, create the Topic called testT in JMSAdmin **before** running the application.

To create a Topic object, invoke the JMSAdmin tool, as described in "Invoking the administration tool" on page 35, and execute one of the following commands, depending on the type of connection you want to make to the broker:

Compatibility mode, or WebSphere MQ Publish/Subscribe

```
InitCtx> def t(testT) bver(V1) topic(test/topic)
```

Native mode, or direct connection to a broker

```
InitCtx> def t(testT) bver(V2) topic(test/topic)
```

Using topics

This section discusses the use of JMS Topic objects in WebSphere MQ classes for Java Message Service applications.

Topic names

This section describes the use of topic names within WebSphere MQ classes for Java Message Service.

Note: The JMS specification does not specify exact details about the use and maintenance of topic hierarchies. Therefore, this area can vary from one provider to the next.

Topic names in WebSphere MQ JMS are arranged in a tree-like hierarchy, an example of which is shown in Figure 1.

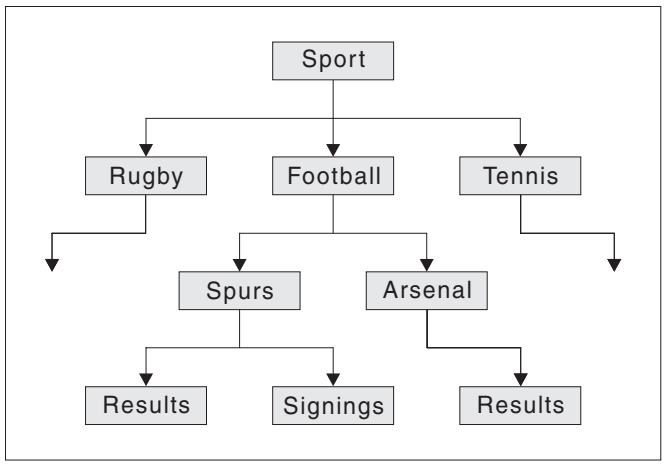

Figure 1. WebSphere MQ classes for Java Message Service topic name hierarchy

In a topic name, levels in the tree are separated by the / character. This means that the Signings node in Figure 1 is identified by the topic name:

```
Sport/Football/Spurs/Signings
```

A powerful feature of the topic system in WebSphere MQ classes for Java Message Service is the use of wildcards. These allow subscribers to subscribe to more than one topic at a time. Different brokers use different wildcard characters and different rules for their substitution. Use the broker version property of the topic (BROKERVER) to define which type of wildcards apply.

Note: The broker version of a topic must match the broker version of the topic connection factory you are using.

Broker Version 1 wildcards

The * wildcard matches zero or more characters; the ? wildcard matches a single character.

If a subscriber subscribes to the topic represented by the following topic name:

```
Sport/Football/*/Results
```

it receives publications on topics including:
- Sport/Football/Spurs/Results
- Sport/Football/Arsenal/Results

If the subscription topic is:

```
Sport/Football/Spurs/*
```

it receives publications on topics including:
- Sport/Football/Spurs/Results
- Sport/Football/Spurs/Signings

If the subscription topic is:

```
Sport/Football/*
```

it receives publications on topics including:
- Sport/Football/Arsenal/Results
- Sport/Football/Spurs/Results
- Sport/Football/Spurs/Signings

Broker Version 2 wildcards

The # wildcard matches multiple levels in a topic; the + wildcard matches a single level.

These wildcards can be used only to stand for complete levels within a topic; that is they can be preceded only by / or start-of-string, and they can be followed only by / or end-of-string.

If a subscriber subscribes to the topic represented by the following topic name:

```
Sport/Football/+/Results
```

it receives publications on topics including:
- Sport/Football/Spurs/Results
- Sport/Football/Arsenal/Results

If a subscriber subscribes to the topic represented by the following topic name:

```
Sport/#/Results
```

it receives publications on topics including:
- Sport/Football/Spurs/Results
- Sport/Football/Arsenal/Results

Although `Sport/Football/Spur?/Results` works with broker Version 1, there is no equivalent for broker Version 2, which does not support single character substitutions.

There is no need to administer the topic hierarchies that you use on the broker-side of your system explicitly. When the first publisher or subscriber on a given topic comes into existence, the broker automatically creates the state of the topics currently being published on, and subscribed to.

Unicode characters are supported.

Note: A publisher cannot publish on a topic whose name contains wildcards.

Creating topics at runtime

There are four ways to create Topic objects at runtime:

1. Construct a topic using the one-argument `MQTopic` constructor
2. Construct a topic using the default `MQTopic` constructor, and then call the `setBaseTopicName(..)` method
3. Use the session's `createTopic(..)` method
4. Use the session's `createTemporaryTopic()` method

Example 1: Using MQTopic(..)

This technique requires a reference to the WebSphere MQ implementation of the JMS Topic interface, and therefore renders the code non-portable.

The constructor takes one argument, which must be a uniform resource identifier (URI). For WebSphere MQ classes for Java Message Service Topics, this must be of the form:

```
topic://TopicName[?property=value[&property=value]*]
```

For further details on URIs and the permitted name-value pairs, see "Sending a message" on page 317.

The following code creates a topic for nonpersistent, priority 5 messages:

```
// Create a Topic using the one-argument MQTopic constructor
String tSpec = "Sport/Football/Spurs/Results?persistence=1&priority=5";
Topic rtTopic = new MQTopic( "topic://" + tSpec );
```

Example 2: Using MQTopic(), then setBaseTopicName(..)

This technique uses the default `MQTopic` constructor, and therefore renders the code non-portable.

After the object is created, set the `baseTopicName` property using the `setBaseTopicName(..)` method, passing in the required topic name.

Note: The topic name used here is the non-URI form, and cannot include name-value pairs. Set these by using the set methods, as described in "Setting properties with the set method" on page 319.

The following code uses this technique to create a topic:

```
// Create a Topic using the default MQTopic constructor
Topic rtTopic = new MQTopic();

// Set the object properties using the setter methods
((MQTopic)rtTopic).setBaseTopicName( "Sport/Football/Spurs/Results" );
((MQTopic)rtTopic).setPersistence(1);
((MQTopic)rtTopic).setPriority(5);
```

Example 3: Using session.createTopic(..)

You can also create a Topic object using the `createTopic` method of TopicSession, which takes a topic URI as follows:

```
// Create a Topic using the session factory method
Topic rtTopic = session.createTopic( "topic://Sport/Football/Spurs/Results" );
```

Although the createTopic method is in the JMS specification, the format of the string argument is vendor-specific. Therefore, using this technique might make your code non-portable.

Example 4: Using session.createTemporaryTopic()

A TemporaryTopic is a Topic that can be consumed only by subscribers that are created by the same TopicConnection. A TemporaryTopic is created as follows:

```
// Create a TemporaryTopic using the session factory method
Topic rtTopic = session.createTemporaryTopic();
```

Subscriber options

There are a number of different ways to use JMS subscribers. This section describes some examples of their use.

JMS provides two types of subscriber:

Non-durable subscribers

These subscribers receive messages on their chosen topic, only if the messages are published while the subscriber is active.

Durable subscribers

These subscribers receive all the messages published on a topic, including those that are published while the subscriber is inactive.

Creating non-durable subscribers

The subscriber created in Create consumers and producers of publications is non-durable and is created with the following code:

```
// Create a subscriber, subscribing on the given topic
 TopicSubscriber sub = session.createSubscriber( topic );
```

Creating durable subscribers

Durable subscribers cannot be configured with a direct connection to a broker.

Creating a durable subscriber is similar to creating a non-durable subscriber, but you must also provide a name that uniquely identifies the subscriber:

```
// Create a durable subscriber, supplying a uniquely-identifying name
TopicSubscriber sub = session.createDurableSubscriber( topic, "D_SUB_000001" );
```

Non-durable subscribers automatically deregister themselves when their `close()` method is called (or when they fall out of scope). However, if you want to terminate a durable subscription, you must explicitly notify the system. To do this, use the session's `unsubscribe()` method and pass in the unique name that created the subscriber:

```
// Unsubscribe the durable subscriber created above
session.unsubscribe( "D_SUB_000001" );
```

A durable subscriber is created at the queue manager specified in the MQTopicConnectionFactory queue manager parameter. If there is a subsequent attempt to create a durable subscriber with the same name at a different queue manager, a new and completely independent durable subscriber is returned.

Using message selectors

You can use message selectors to filter out messages that do not satisfy given criteria. For details about message selectors, see "Message selectors" on page 321. Message selectors are associated with a subscriber as follows:

```
// Associate a message selector with a non-durable subscriber
String selector = "company = 'IBM'";
TopicSubscriber sub = session.createSubscriber( topic, selector, false );
```

You can control whether the JMS client or the broker performs message filtering by setting the MessageSelection property on the TopicConnectionFactory. If the broker is capable of performing message selection, it is generally preferable to let the broker do it because it reduces the number of messages sent to your client. However, if the broker is very heavily loaded, it might be preferable to let the client perform message selection instead.

Suppressing local publications

You can create a subscriber that ignores publications that are published on the subscriber's own connection. Set the third parameter of the `createSubscriber` call to true, as follows:

```
// Create a non-durable subscriber with the noLocal option set
TopicSubscriber sub = session.createSubscriber( topic, null, true );
```

Combining the subscriber options

You can combine the subscriber variations, so that you can create a durable subscriber that applies a selector and ignores local publications. The following code fragment shows the use of the combined options:

```
// Create a durable, noLocal subscriber with a selector applied
String selector = "company = 'IBM'";
TopicSubscriber sub = session.createDurableSubscriber( topic, "D_SUB_000001",
                                                       selector, true );
```

Configuring the base subscriber queue

Subscriber queues cannot be configured for a direct connection to a broker.

There are two ways in which you can configure subscribers:

- Multiple queue approach

  Each subscriber has an exclusive queue assigned to it, from which it retrieves all its messages. JMS creates a new queue for each subscriber. This is the only approach available with WebSphere MQ JMS V1.1.

- Shared queue approach

  A subscriber uses a shared queue, from which it, and other subscribers, retrieve their messages. This approach requires only one queue to serve multiple subscribers. This is the default approach used with WebSphere MQ JMS.

You can choose which approach to use, and configure which queues to use.

In general, the shared queue approach gives a modest performance advantage. For systems with a high throughput, there are also large architectural and administrative advantages, because of the significant reduction in the number of queues required.

In some situations, there are still good reasons for using the multiple queue approach:

- The theoretical physical capacity for message storage is greater.

  There is an upper limit to the number of messages that a WebSphere MQ queue can hold and so, in the shared queue approach, the total number of messages for all the subscribers that share the queue cannot exceed this limit. This issue is more significant for durable subscribers, because the lifetime of a durable subscriber is usually much longer than that of a non-durable subscriber. Therefore, more messages might accumulate for a durable subscriber.

- External administration of subscription queues is easier.

For certain application types, administrators might want to monitor the state and depth of particular subscriber queues. This task is much simpler when there is one to one mapping between a subscriber and a queue.

Default configuration

The default configuration uses the following shared subscription queues:

- SYSTEM.JMS.ND.SUBSCRIBER.QUEUE for non-durable subscriptions
- SYSTEM.JMS.D.SUBSCRIBER.QUEUE for durable subscriptions

These are created for you when you run the MQJMS_PSQ.MQSC script.

If required, you can specify alternative physical queues. You can also change the configuration to use the multiple queue approach.

Configuring non-durable subscribers

You can set the non-durable subscriber queue name property in either of the following ways:

- Use the WebSphere MQ JMS administration tool (for JNDI retrieved objects) to set the BROKERSUBQ property
- Use the setBrokerSubQueue() method in your program

For non-durable subscriptions, the queue name you provide should start with the following characters:

- SYSTEM.JMS.ND.

To select a shared queue approach, specify an explicit queue name, where the named queue is the one to use for the shared queue. The queue that you specify must already physically exist before you create the subscription.

To select the multiple queue approach, specify a queue name that ends with the * character. Subsequently, each subscriber that is created with this queue name creates an appropriate dynamic queue, for exclusive use by that particular subscriber. MQ JMS uses its own internal model queue to create such queues. Therefore, with the multiple queue approach, all required queues are created dynamically.

When you use the multiple queue approach, you cannot specify an explicit queue name. However, you can specify the queue prefix. This enables you to create different subscriber queue domains. For example, you could use:

- SYSTEM.JMS.ND.MYDOMAIN.*

The characters that precede the * character are used as the prefix, so that all dynamic queues that are associated with this subscription have queue names that start with SYSTEM.JMS.ND.MYDOMAIN.

Configuring durable subscribers

As discussed earlier, there might still be good reasons to use the multiple queue approach for durable subscriptions. Durable subscriptions are likely to have a longer life span, so it is possible that a large number of unretrieved messages could accumulate on the queue.

Therefore, the durable subscriber queue name property is set in the Topic object (that is, at a more manageable level than TopicConnectionFactory). This enables you to specify a number of different subscriber queue names, without needing to re-create multiple objects starting from the TopicConnectionFactory.

You can set the durable subscriber queue name in either of the following ways:

- Use the WebSphere MQ JMS administration tool (for JNDI retrieved objects) to set the BROKERDURSUBQ property
- Use the setBrokerDurSubQueue() method in your program:

```
// Set the MQTopic durable subscriber queue name using
// the multi-queue approach
sportsTopic.setBrokerDurSubQueue("SYSTEM.JMS.D.FOOTBALL.*");
```

  Once the Topic object is initialized, it is passed into the TopicSession createDurableSubscriber() method to create the specified subscription:

```
// Create a durable subscriber using our earlier Topic
TopicSubscriber sub = new session.createDurableSubscriber
                                              (sportsTopic, "D_SUB_SPORT_001");
```

For durable subscriptions, the queue name you provide must start with the following characters:

- SYSTEM.JMS.D.

To select a shared queue approach, specify an explicit queue name, where the named queue is the one to use for the shared queue. The queue that you specify must already physically exist before you create the subscription.

To select the multiple queue approach, specify a queue name that ends with the * character. Subsequently, each subscriber that is created with this queue name creates an appropriate dynamic queue, for exclusive use by that subscriber. MQ JMS uses its own internal model queue to create such queues. Therefore, with the multiple queue approach, all required queues are created dynamically.

When you use the multiple queue approach, you cannot specify an explicit queue name. However, you can specify the queue prefix. This enables you to create different subscriber queue domains. For example, you could use:

- SYSTEM.JMS.D.MYDOMAIN.*

The characters that precede the * character are used as the prefix, so that all dynamic queues that are associated with this subscription have queue names that start with SYSTEM.JMS.D.MYDOMAIN.

You cannot change the queue used by a durable subscriber. To do so, for example to move from the multiple queue approach to the single queue approach, first delete the old subscriber (using the unsubscribe() method) and create the subscription again from new. This removes any unconsumed messages on the durable subscription.

Subscription stores

There is no subscription store with a direct connection to a broker.

WebSphere MQ JMS maintains a persistent store of subscription information, used to resume durable subscriptions and cleanup after failed non-durable subscribers. This information can be managed by the publish/subscribe broker.

The choice of subscription store is based on the SUBSTORE property of the TopicConnectionFactory. This takes one of three values: QUEUE, BROKER, or MIGRATE.

SUBSTORE(QUEUE)
Subscription information is stored on SYSTEM.JMS.ADMIN.QUEUE and SYSTEM.JMS.PS.STATUS.QUEUE on the local queue manager.

WebSphere MQ JMS maintains an extra connection for a long-running transaction used to detect failed subscribers. There is a connection to each queue manager in use. In a busy system, this might cause the queue manager logs to fill up, resulting in errors from both the queue manager and its applications.

If you experience these problems, the system administrator can add extra log files or datasets to the queue manager. Alternatively, reduce the STATREFRESHINT property on the TopicConnectionFactory. This causes the long-running transaction to be refreshed more frequently, allowing the logs to reset themselves.

After a non-durable subscriber has failed:

- Information is left on the two SYSTEM.JMS queues, which allows a later JMS application to clean up after the failed subscriber. See "Subscriber cleanup utility" on page 344 for more information.
- Messages continue to be delivered to the subscriber until a later JMS application is executed.

SUBSTORE(QUEUE) is provided for compatibility with versions of MQSeries JMS.

SUBSTORE(BROKER)
Subscription information is stored by the publish/subscribe broker used by the application. This option is designed to provide improved resilience.

When a non-durable subscriber fails, the subscription is deregistered from the broker as soon as possible. The broker adds a response to this deregistration onto the SYSTEM.JMS.REPORT.QUEUE, which is used to clean up after the failed subscriber. With SUBSTORE(BROKER), a separate cleanup thread is run regularly in the background of each JMS publish/subscribe application.

Cleanup is run once every 10 minutes by default, but you can change this using the CLEANUPINT property on the TopicConnectionFactory. To customize the actions performed by cleanup, use the CLEANUP property on the TopicConnectionFactory.

See "Subscriber cleanup utility" on page 344 for more information about the different behaviors of cleanup with SUBSTORE(BROKER).

SUBSTORE(MIGRATE)
MIGRATE is the default value for SUBSTORE.

This option dynamically selects the queue-based or broker-based subscription store based on the levels of queue manager and publish/subscribe broker installed. If both queue manager and broker are capable of supporting SUBSTORE(BROKER), this behaves as SUBSTORE(BROKER); otherwise it behaves as SUBSTORE(QUEUE). Additionally, SUBSTORE(MIGRATE) transfers durable subscription information from the queue-based subscription store to the broker-based store.

This provides an easy migration path from older versions of WebSphere MQ JMS, WebSphere MQ, and publish/subscribe broker. This migration occurs the first time the durable subscription is opened when both queue

manager and broker are capable of supporting the broker-based subscription store. Only information relating to the subscription being opened is migrated; information relating to other subscriptions is left in the queue-based subscription store.

Migration and coexistence considerations

Except when SUBSTORE(MIGRATE) is used, the two subscription stores are entirely independent.

A durable subscription is created in the store specified by the TopicConnectionFactory. If there is a subsequent attempt to create a durable subscriber with the same name and ClientID but with the other subscription store, a new durable subscription is created.

When a connection uses SUBSTORE(MIGRATE), subscription information is automatically transferred from the queue-based subscription store to the broker-based subscription store when createDurableSubscriber() is called. If a durable subscription with matching name and ClientID already exists in the broker-based subscription store, this migration cannot complete and an exception is thrown from createDurableSubscriber().

Once a subscription has been migrated, it is important not to access the subscription from an application using an older version of WebSphere MQ JMS, or an application running with SUBSTORE(QUEUE). This would create a subscription in the queue-based subscription store, and prevent an application running with SUBSTORE(MIGRATE) from using the subscription.

To recover from this situation, either use SUBSTORE(BROKER) from your application or unsubscribe from the subscription with SUBSTORE(QUEUE).

Solving publish/subscribe problems

This section describes some problems that can occur when you develop JMS client applications that use the publish/subscribe domain. It discusses problems that are specific to the publish/subscribe domain. Refer to “Handling errors” on page 323 and “Solving problems” on page 33 for more general troubleshooting guidance.

Incomplete publish/subscribe close down

It is important that JMS client applications surrender all external resources when they terminate. To do this, call the `close()` method on all objects that can be closed once they are no longer required. For the publish/subscribe domain, these objects are:

- TopicConnection
- TopicSession
- TopicPublisher
- TopicSubscriber

The WebSphere MQ classes for Java Message Service implementation eases this task by using a *cascading close*. With this process, a call to close on a TopicConnection results in calls to close on each of the TopicSessions it created. This in turn results in calls to close on all TopicSubscribers and TopicPublishers the sessions created.

To ensure the proper release of external resources, call `connection.close()` for each of the connections that an application creates.

There are some circumstances where this close procedure might not complete. These include:

- Loss of a WebSphere MQ client connection
- Unexpected application termination

In these circumstances, the `close()` is not called, and external resources remain open on the terminated application's behalf. The main consequences of this are:

Broker state inconsistency
The WebSphere MQ Message Broker might contain registration information for subscribers and publishers that no longer exist. This means that the broker might continue forwarding messages to subscribers that will never receive them.

Subscriber messages and queues remain
Part of the subscriber deregistration procedure is the removal of subscriber messages. If appropriate, the underlying WebSphere MQ queue that was used to receive subscriptions is also removed. If normal closure has not occurred, these messages and queues remain. If there is broker state inconsistency, the queues continue to fill up with messages that will never be read.

Additionally, if the broker resides on a queue manager other than the application's local queue manager, correct operation of WebSphere MQ JMS depends on the communication channels between the two queue managers. If these channels fail for any reason, problems such as the above can occur until the channels restart. When diagnosing problems relating to channels, be careful not to lose WebSphere MQ JMS control messages on the transmission queue.

Subscriber cleanup utility

To avoid the problems associated with non-graceful closure of subscriber objects, WebSphere MQ JMS includes a subscriber cleanup utility that attempts to detect any earlier WebSphere MQ JMS publish/subscribe problems that could have resulted from other applications. This utility runs transparently in the background and should not affect other WebSphere MQ JMS operations. If a large number of problems are detected against a given queue manager, you might see some performance degradation while resources are cleaned up.

Note: Close all subscriber objects gracefully whenever possible, to avoid a buildup of subscriber problems.

The exact behavior of the utility depends on the subscription store in use:

Subscriber cleanup with SUBSTORE(QUEUE)
When using the queue-based subscription store, cleanup runs on a queue manager when the first TopicConnection to use that physical queue manager initializes.

If all the TopicConnections on a given queue manager close, when the next TopicConnection initializes for that queue manager, the utility runs again.

The cleanup utility uses information found on the SYSTEM.JMS.ADMIN.QUEUE and SYSTEM.JMS.PS.STATUS.QUEUE to detect previously failed subscribers. If any are found, it cleans up

associated resources by deregistering the subscriber from the broker, and cleaning up any unconsumed messages or temporary queues associated with the subscription.

Subscriber cleanup with SUBSTORE(BROKER)

With the broker-based subscription store, cleanup runs regularly on a background thread while there is an open TopicConnection to a particular physical queue manager. One instance of the cleanup thread is created for each physical queue manager to which a TopicConnection exists within the JVM.

The cleanup utility uses information found on the SYSTEM.JMS.REPORT.QUEUE (typically responses from the publish/subscribe broker) to remove unconsumed messages and temporary queues associated with a failed subscriber. It can be a few seconds after the subscriber fails before the cleanup routine can remove the messages and queues.

Two properties on the TopicConnectionFactory control behavior of this cleanup thread: CLEANUP and CLEANUPINT. CLEANUPINT determines how often (in milliseconds) cleanup is executed against any given queue manager. CLEANUP takes one of four possible values:

CLEANUP(SAFE)

This is the default value.

The cleanup thread tries to remove unconsumed subscription messages or temporary queues for failed subscriptions. This mode of cleanup does not interfere with the operation of other JMS applications.

CLEANUP(STRONG)

The cleanup thread performs as CLEANUP(SAFE), but also clears the SYSTEM.JMS.REPORT.QUEUE of any unrecognized messages.

This mode of cleanup can interfere with the operation of JMS applications running with later versions of WebSphere MQ JMS. If multiple JMS applications are using the same queue manager, but using different versions of WebSphere MQ JMS, only clients using the most recent version of WebSphere MQ JMS must use this option.

CLEANUP(NONE)

In this special mode, no cleanup is performed, and no cleanup thread exists. Additionally, if the application is using the single-queue approach, unconsumed messages can be left on the queue.

This option can be useful if the application is distant from the queue manager, and especially if it only publishes rather than subscribes. However, one application must clean up the queue manager to deal with any unconsumed messages. This can be a JMS application with CLEANUP(SAFE) or CLEANUP(STRONG), or the manual cleanup utility described in “Manual cleanup” on page 346.

CLEANUP(ASPROP)

The style of cleanup to use is determined by the system property com.ibm.mq.jms.cleanup, which is queried at JVM startup.

This property can be set on the Java command-line using the -D option, to NONE, SAFE, or STRONG. Any other value causes an exception. If not set, the property defaults to SAFE.

This allows easy JVM-wide change to the cleanup level without updating every TopicConnectionFactory used by the system. This is useful for applications or application servers that use multiple TopicConnectionFactory objects.

Where multiple TopicConnections exist within a JVM against the same queue manager, but with differing values for CLEANUP and CLEANUPINT, the following rules are used to determine behavior:

1. A TopicConnection with CLEANUP(NONE) does not attempt to clean up immediately after its subscription has closed. However, if another TopicConnection is using SAFE or STRONG cleanup, the cleanup thread eventually cleans up after the subscription.
2. If any TopicConnection is using STRONG Cleanup, the cleanup thread operates at STRONG level. Otherwise, if any TopicConnection uses SAFE Cleanup, the cleanup thread operates at SAFE level. Otherwise, there is no cleanup thread.
3. The smallest value of CLEANUPINT for those TopicConnections with SAFE or STRONG Cleanup is used.

Manual cleanup

If you use the broker-based subscription store, you can operate cleanup manually from the command-line. The syntax for bindings attach is:

```
Cleanup [-m <qmgr>] [-r <interval>]
          [SAFE | STRONG | FORCE | NONDUR] [-t]
```

or, for client attach:

```
Cleanup -client [-m <qmgr>] -host <hostname> [-port <port>]
         [-channel <channel>] [-r <interval>]
         [SAFE | STRONG | FORCE | NONDUR] [-t]
```

Where:

- `qmgr`, `hostname`, `port`, and `channel` determine connection settings for the queue manager to clean up.
- `-r` sets the interval between executions of cleanup, in minutes. If not set, cleanup is performed once.
- `-t` enables tracing, to the mqjms.trc file.
- SAFE, STRONG, FORCE, and NONDUR set the cleanup level, as follows:
  - SAFE and STRONG cleanup behave like the CLEANUP(SAFE) and CLEANUP(STRONG) options discussed in “Subscriber cleanup utility” on page 344.
  - FORCE cleanup behaves like STRONG Cleanup. However, STRONG cleanup leaves messages that could not be processed on the SYSTEM.JMS.REPORT.QUEUE; FORCE cleanup removes all messages even if it encounters an error during processing.

    Warning: This is a dangerous option that can leave an inconsistent state between the queue manager and the broker. It cannot be run while any &mqjms; &pubsub; application is running against the queue manager; doing so causes the cleanup utility to abort.
  - NONDUR behaves like FORCE cleanup.

After clearing the SYSTEM.JMS.REPORT.QUEUE, it attempts to remove any remaining unconsumed messages sent to non-durable subscribers. If the queue manager's command server is running on any queue beginning SYSTEM.JMS.ND.*, messages are cleared and the queue itself might be deleted. Otherwise, only SYSTEM.JMS.ND.SUBSCRIBER.QUEUE and SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE are cleared of messages.

Cleanup from within a program

You can use a programming interface to the cleanup routines for use with SUBSTORE(BROKER), through the class com.ibm.mq.jms.Cleanup. Instances of this class have getter/setter methods for each of the connection properties; and also for the cleanup level and interval.

It exposes two methods:

cleanup()
Executes cleanup once

run() Runs cleanup at intervals determined by the properties of the cleanup object

This class allows complete customization of publish/subscribe Cleanup, but it is intended for use by system administration programs rather than application programs.

For more details, refer to "Cleanup" on page 470.

Handling broker reports

The WebSphere MQ JMS implementation uses report messages from the broker to confirm registration and deregistration commands. These reports are normally consumed by the WebSphere MQ classes for Java Message Service implementation, but under some error conditions, they might remain on the queue. These messages are sent to the `SYSTEM.JMS.REPORT.QUEUE` queue on the local queue manager.

A Java application, `PSReportDump`, is supplied with WebSphere MQ classes for Java Message Service, which dumps the contents of this queue in plain text format. The information can then be analyzed, either by you, or by IBM support staff. You can also use the application to clear the queue of messages after a problem is diagnosed or fixed.

The compiled form of the tool is installed in the `<MQ_JAVA_INSTALL_PATH>/bin` directory. To invoke the tool, change to this directory, then use the following command:

```
java -Djava.library.path=library_path
     PSReportDump [-m queueManager] [-clear]
```

where *library_path* is the path to the WebSphere MQ Java libraries (see "The WebSphere MQ Java libraries" on page 10), and:

-m *queueManager*
Specifies the name of the queue manager to use

-clear Causes all the messages on the queue to be deleted after their contents have been dumped

Attention: Do not use this option if you are using the broker-based subscription store. Instead, run the manual cleanup utility at FORCE level.

Output is sent to the screen, or you can redirect it to a file.

Other considerations

If a large number of JMS clients connect directly to a broker running on Windows, and the connections happen almost simultaneously, a java.net.BindException address in use exception might be thrown in response to a TopicConnection call. You can try to avoid this by catching the exception and retrying, or by pacing the connections.

Chapter 12. Writing WebSphere MQ JMS 1.1 applications

This chapter provides information to help you write WebSphere MQ JMS 1.1 applications. It covers information similar to that provided in Chapter 10, "Writing WebSphere MQ JMS applications," on page 313 and Chapter 11, "Writing WebSphere MQ JMS publish/subscribe applications," on page 327, but from a JMS 1.1 perspective.

The JMS 1.1 model

You can write a JMS application using two styles of messaging:

- Point-to-point
- Publish/subscribe

These styles of messaging are also referred to as *messaging domains* and you can combine both styles of messaging in one application.

With versions of JMS before JMS 1.1, programming for the point-to-point domain uses one set of interfaces and methods, and programming for the publish/subscribe domain uses another set. The two sets are similar, but separate. With JMS 1.1, you can use a common set of interfaces and methods that support both messaging domains. The common interfaces provide a *domain independent* view of each messaging domain. Table 16 lists the JMS 1.1 domain independent interfaces and their corresponding *domain specific* interfaces.

Table 16. The JMS 1.1 domain independent interfaces

Domain independent interfaces	**Domain specific interfaces for the point-to-point domain**	**Domain specific interfaces for the publish/subscribe domain**
ConnectionFactory	QueueConnectionFactory	TopicConnectionFactory
Connection	QueueConnection	TopicConnection
Destination	Queue	Topic
Session	QueueSession	TopicSession
MessageProducer	QueueSender	TopicPublisher
MessageConsumer	QueueReceiver QueueBrowser	TopicSubscriber

JMS 1.1 retains all the domain specific interfaces in JMS 1.0.2b, and so existing applications can still use these interfaces. For new applications, however, consider using the JMS 1.1 domain independent interfaces.

In the WebSphere MQ JMS implementation of JMS 1.1, the administered objects are the following:

- ConnectionFactory
- Queue
- Topic

Destination is an abstract superclass of Queue and Topic, and so an instance of Destination is either a Queue or a Topic object. The domain independent interfaces

treat a queue or a topic as a destination. The messaging domain for a MessageConsumer or a MessageProducer object is determined by whether the destination is a queue or a topic.

Building a connection

Connections are not created directly, but are built using a connection factory. ConnectionFactory objects can be stored in a JNDI namespace, insulating the JMS application from provider specific information. For information about how to create and store ConnectionFactory objects, see Chapter 5, "Using the WebSphere MQ JMS administration tool," on page 35.

Retrieving a connection factory from JNDI

To retrieve a ConnectionFactory object from a JNDI namespace, you must first set up an initial context as shown in the following code:

```
import javax.jms.*;
import javax.naming.*;
import javax.naming.directory.*;
 .
 .
 .
  java.util.Hashtable environment = new java.util.Hashtable();
  environment.put(Context.INITIAL_CONTEXT_FACTORY, icf);
  environment.put(Context.PROVIDER_URL, url);
  Context ctx = new InitialDirContext( environment );
```

In this code:

icf Defines a factory class for the initial context

url Defines a context specific URL

For more details about using a JNDI namespace, see Sun's JNDI documentation.

Note: Some combinations of the JNDI packages and LDAP service providers can result in an LDAP error 84. To resolve the problem, insert the following line before the call to InitialDirContext:

```
environment.put(Context.REFERRAL, "throw");
```

After an initial context is obtained, you can retrieve a ConnectionFactory object from the JNDI namespace by using the lookup() method. The following code retrieves a ConnectionFactory object named CF from an LDAP based namespace:

```
ConnectionFactory factory;
factory = (ConnectionFactory)ctx.lookup("cn=CF");
```

Creating a connection factory at runtime

If a JNDI namespace is not available, it is possible to create a ConnectionFactory object at runtime. However, using this method reduces the portability of a JMS application because the application must then include references to WebSphere MQ specific classes.

The following code creates a ConnectionFactory object with all the default settings:

```
factory = new com.ibm.mq.jms.MQConnectionFactory();
```

The default transport type is bindings. You can change the transport type for a connection factory by using the setTransportType() method. Here are some examples:

```
fact.setTransportType(JMSC.MQJMS_TP_BINDINGS_MQ);     // Bindings mode
fact.setTransportType(JMSC.MQJMS_TP_CLIENT_MQ_TCPIP); // Client mode
fact.setTransportType(JMSC.MQJMS_TP_DIRECT_TCPIP);    // Direct TCP/IP mode
```

For information about transport types in each of the specific messaging domains, see “Choosing client or bindings transport” on page 316, for the point-to-point domain, and “TopicConnectionFactory administered objects” on page 334, for the publish/subscribe domain.

Note that you cannot use the point-to-point style of messaging if the transport type is direct. If an application uses Connection and Session objects that are created from a ConnectionFactory object whose transport type is direct, the application can perform only publish/subscribe operations.

A ConnectionFactory object has the same properties as those of a QueueConnectionFactory object and a TopicConnectionFactory object combined. However, certain combinations of property settings are not valid for a ConnectionFactory object. See “Properties” on page 42 for more details.

Using a connection factory to create a connection

You can use the createConnection() method on a ConnectionFactory object to create a Connection object, as shown in the following example:

```
Connection connection;
connection = factory.createConnection();
```

Both QueueConnectionFactory and TopicConnectionFactory inherit the createConnection() method from ConnectionFactory. You can therefore use createConnection() to create a domain specific object, as shown in the following code:

```
QueueConnectionFactory QCF;
Connection connection;
connection = QCF.createConnection();
```

This code creates a QueueConnection object. An application can now perform a domain independent operation on this object, or an operation that is applicable only to the point-to-point domain. If the application attempts to perform an operation that is applicable only to the publish/subscribe domain, an IllegalStateException is thrown with the message `MQJMS1112: JMS 1.1 Invalid operation for a domain specific object.` This is because the connection was created from a domain specific connection factory.

Starting a connection

The JMS specification states that a connection is created in the `stopped` state. Until the connection starts, a message consumer that is associated with the connection cannot receive any messages. To start the connection, issue the following command:

```
connection.start();
```

Using a client channel definition table

As an alternative to creating a client connection channel definition by setting certain properties of a ConnectionFactory object, a WebSphere MQ JMS client application can use client connection channel definitions that are stored in a client channel definition table. These definitions are created by WebSphere MQ Script (MQSC) commands or WebSphere MQ Programmable Command Format (PCF) commands. When the application creates a Connection object, the WebSphere MQ

JMS client searches the client channel definition table for a suitable client connection channel definition, and uses the channel definition to start an MQI channel. For more information about client channel definition tables and how to construct one, see *WebSphere MQ Clients*.

To use a client channel definition table, the CCDTURL property of a ConnectionFactory object must be set to a URL object. The URL object encapsulates a uniform resource locator (URL) that identifies the name and location of the file containing the client channel definition table and specifies how the file can be accessed. You can set the CCDTURL property by using the WebSphere MQ JMS administration tool, or an application can set the property by creating a URL object and calling the setCCDTURL() method of the ConnectionFactory object.

For example, if the file ccdt1.tab contains a client channel definition table and is stored on the same system on which the application is running, the application can set the CCDTURL property in the following way:

```
java.net.URL chanTab1 = new URL("file:///home/admdata/ccdt1.tab");
factory.setCCDTURL(chanTab1);
```

As another example, suppose the file ccdt2.tab contains a client channel definition table and is stored on a system that is different to the one on which the application is running. If the file can be accessed using the FTP protocol, the application can set the CCDTURL property in the following way:

```
java.net.URL chanTab2 = new URL("ftp://ftp.server/admdata/ccdt2.tab");
factory.setCCDTURL(chanTab2);
```

In addition to setting the CCDTURL property of the ConnectionFactory object, the QMANAGER property of the same object must be set to one of the following values:

- The name of a queue manager
- An asterisk (*) followed by the name of a queue manager group
- An asterisk (*)
- An empty string, or a string containing all blank characters

These are the same values that can be used for the *QMgrName* parameter on an MQCONN call issued by a client application that is using Message Queue Interface (MQI). For more information about the meaning of these values therefore, see the *WebSphere MQ Application Programming Reference* and *WebSphere MQ Clients*. You can set the QMANAGER property by using the WebSphere MQ JMS administration tool, or an application can set the property by calling the setQueueManager() method of the ConnectionFactory object.

If an application then creates a Connection object from the ConnectionFactory object, the WebSphere MQ JMS client accesses the client channel definition table identified by the CCDTURL property, uses the QMANAGER property to search the table for a suitable client connection channel definition, and then uses the channel definition to start an MQI channel to a queue manager. The way that the WebSphere MQ JMS client uses the QMANAGER property to search the client channel definition table is also as described in the *WebSphere MQ Application Programming Reference* and *WebSphere MQ Clients*. If your application uses connection pooling, see also “Controlling the default connection pool” on page 82.

Note that the CCDTURL and CHANNEL properties of a ConnectionFactory object cannot both be set when the application calls the createConnection() method. If both properties are set, the method throws an exception. The CCDTURL or

CHANNEL property is considered to be set if its value is anything other than null, an empty string, or a string containing all blank characters.

When the WebSphere MQ JMS client finds a suitable client connection channel definition in the client channel definition table, it uses only the information extracted from the table to start an MQI channel. Any channel related properties of the ConnectionFactory object are ignored.

In particular, note the following points if you are using Secure Sockets Layer (SSL):

- An MQI channel uses SSL only if the channel definition extracted from the client channel definition table specifies the name of a CipherSpec supported by the WebSphere MQ JMS client.
- A client channel definition table also contains information about the location of Lightweight Directory Access Protocol (LDAP) servers that hold certificate revocation lists (CRLs). The WebSphere MQ JMS client uses only this information to access LDAP servers that hold CRLs.

For more information about using SSL with a client channel definition table, see *WebSphere MQ Clients*.

Note also the following points if you are using channel exits:

- An MQI channel uses only the channel exits and associated user data specified by the channel definition extracted from the client channel definition table.
- A channel definition extracted from a client channel definition table can specify channel exits that are written in Java. This means, for example, that the SCYEXIT parameter on the DEFINE CHANNEL command to create a client connection channel definition can specify the name of a class that implements the WebSphere MQ base Java interface, MQSecurityExit. Similarly, the SENDEXIT parameter can specify the name of a class that implements the MQSendExit interface, and the RCVEXIT parameter can specify the name of a class that implements the MQReceiveExit interface. For more information about how to write a channel exit in Java, see "Using channel exits" on page 77.

  The use of channel exits written in a language other than Java is also supported. For information about how to specify the SCYEXIT, SENDEXIT, and RCVEXIT parameters on the DEFINE CHANNEL command for channel exits written in another language, see the *WebSphere MQ Script (MQSC) Command Reference*.

Specifying a range of ports for client connections

When a WebSphere MQ JMS application attempts to connect to a WebSphere MQ queue manager in client mode, a firewall might allow only those connections that originate from specified ports or a range of ports. In this situation, you can use the LOCALADDRESS property of a ConnectionFactory, QueueConnectionFactory, or TopicConnectionFactory object to specify a port, or a range of ports, that the application can bind to.

You can set the LOCALADDRESS property by using the WebSphere MQ JMS administration tool, or by calling the setLocalAddress() method in a JMS application. Here is an example of setting the property from within an application:

```
mqConnectionFactory.setLocalAddress("9.20.0.1(2000,3000)");
```

When the application connects to a queue manager subsequently, the application binds to a local IP address and port number in the range 9.20.0.1(2000) to 9.20.0.1(3000).

In a system with more than one network interface, you can also use the LOCALADDRESS property to specify which network interface must be used for a connection.

For a direct connection to a broker, the LOCALADDRESS property is relevant only when multicast is used. In this case, you can use the property to specify which local network interface must be used for a connection, but the value of the property must not contain a port number, or a range of port numbers.

Connection errors might occur if you restrict the range of ports. If an error occurs, a JMSException is thrown with an embedded MQException that contains the WebSphere MQ reason code MQRC_Q_MGR_NOT_AVAILABLE and the following message:

```
Socket connection attempt refused due to LOCAL_ADDRESS_PROPERTY restrictions
```

An error might occur if all the ports in the specified range are in use, or if the specified IP address, host name, or port number is not valid (a negative port number, for example).

Because the WebSphere MQ JMS client might create connections other than those required by an application, always consider specifying a range of ports. In general, every session created by an application requires one port and the WebSphere MQ JMS client might require three or four additional ports. If a connection error does occur, increase the range of ports.

Connection pooling, which is used by default in WebSphere MQ JMS, might have an effect on the speed at which ports can be reused. As a result, a connection error might occur while ports are being freed.

Channel compression

Compressing the data that flows on a WebSphere MQ channel can improve the performance of the channel and reduce network traffic. Using function supplied with WebSphere MQ, you can compress the data that flows on message channels and MQI channels and, on either type of channel, you can compress header data and message data independently of each other. By default, no data is compressed on a channel. For a full description of channel compression, including how it is implemented in WebSphere MQ, see *WebSphere MQ Intercommunication*, for message channels, and *WebSphere MQ Clients*, for MQI channels.

A WebSphere MQ JMS application specifies the techniques that can be used for compressing header or message data on a connection by creating a java.util.Collection object. Each compression technique is an Integer object in the collection, and the order in which the application adds the compression techniques to the collection is the order in which the compression techniques are negotiated with the queue manager when the application creates the connection. The application can then pass the collection to a ConnectionFactory object by calling the setHdrCompList() method, for header data, or the setMsgCompList() method, for message data. When the application is ready, it can create the connection.

The following code fragments illustrate the approach just described. The first code fragment shows you how to implement header data compression:

```
Collection headerComp = new Vector();
headerComp.add(new Integer(JMSC.MQJMS_COMPHDR_SYSTEM));
.
.
.
```

```
((MQConnectionFactory) cf).setHdrCompList(headerComp);
.
.
.
connection = cf.createConnection();
```

The second code fragment shows you how to implement message data compression:

```
Collection msgComp = new Vector();
msgComp.add(new Integer(JMSC.MQJMS_COMPMSG_RLE));
msgComp.add(new Integer(JMSC.MQJMS_COMPMSG_ZLIB_HIGH));
.
.
.
((MQConnectionFactory) cf).setMsgCompList(msgComp);
.
.
.
connection = cf.createConnection();
```

In the second example, the compression techniques are negotiated in the order RLE, then ZLIB_HIGH, when the connection is created. The compression technique that is selected cannot be changed during the lifetime of the Connection object. Note that, to use compression on a connection, the setHdrCompList() and the setMsgCompList() methods must be called before creating the Connection object.

For more information about specifying compression techniques, and about which compression techniques are available, see "MQConnectionFactory" on page 478 and "JMSC" on page 566.

Obtaining a session

After a connection is made, use the createSession() method on the Connection object to obtain a session. The method has two parameters:

1. A boolean parameter that determines whether the session is transacted or non-transacted.
2. A parameter that determines the acknowledge mode.

The simplest case is obtaining a non-transacted session with an acknowledge mode of AUTO_ACKNOWLEDGE, as shown in the following code:

```
Session session;
.
.
.
boolean transacted = false;
session = connection.createSession(transacted, Session.AUTO_ACKNOWLEDGE);
```

Note: A connection is thread safe, but sessions (and the objects that are created from them) are not. The recommended practice for multithreaded applications is to use a separate session for each thread.

Destinations

The Destination interface is the abstract superclass of Queue and Topic. In the WebSphere MQ JMS implementation of JMS 1.1, Queue and Topic objects encapsulate addresses in WebSphere MQ and the broker. For example, a Queue object encapsulates the name of a WebSphere MQ queue.

For information about using Queue objects in the point-to-point domain, see “Sending a message” on page 317 and, for information about using Topic objects in the publish/subscribe domain, see “Using topics” on page 335. The following is a overview from a domain independent perspective.

Queue and Topic objects are retrieved from a JNDI namespace in the following way:

```
Queue ioQueue;
ioQueue = (Queue)ctx.lookup( qLookup );
.
.
.
Topic ioTopic;
ioTopic = (Topic)ctx.lookup( tLookup );
```

The WebSphere MQ JMS implementation of Queue and Topic interfaces are in the com.ibm.mq.jms.MQQueue and com.ibm.qm.jms.MQTopic classes respectively. These classes contain properties that control the behavior of WebSphere MQ and the broker but, in many cases, it is possible to use the default values. As well as being able to administer Queue and Topic objects in a JNDI namespace, JMS defines a standard way of specifying a destination at runtime that minimizes the WebSphere MQ specific code in the application. This mechanism uses the Session.createQueue() and Session.createTopic() methods, which take a string parameter that specifies the destination. The string is still in a provider specific format, but this approach is more flexible than referring directly to the provider classes.

WebSphere MQ JMS accepts two forms for the string parameter of createQueue():

- The first is the name of a WebSphere MQ queue:

  ```
  public static final String QUEUE = "SYSTEM.DEFAULT.LOCAL.QUEUE" ;
  .
  .
  .
  ioQueue = session.createQueue( QUEUE );
  ```

- The second, and more powerful, form is a uniform resource identifier (URI). This form allows you to specify a remote queue, which is a queue on a queue manager other than the one to which you are connected. It also allows you to set the other properties of a com.ibm.mq.jms.MQQueue object.

  The URI for a queue begins with the sequence `queue://`, followed by the name of the queue manager on which the queue resides. This is followed by a further forward slash (/), the name of the queue, and, optionally, a list of name-value pairs that set the remaining queue properties. For example, the URI equivalent of the previous example is the following:

  ```
  ioQueue = session.createQueue("queue:///SYSTEM.DEFAULT.LOCAL.QUEUE");
  ```

  The name of the queue manager is omitted. This is interpreted to mean as the queue manager to which the owning Connection object is connected at the time when the Queue object is used.

Note: When sending a message to a cluster, leave the queue manager field in the Queue object blank. This enables an MQOPEN call to be performed in BIND_NOT_FIXED mode, which allows the queue manager to be determined. Otherwise an exception is returned reporting that the Queue object cannot be found. This applies when using JNDI or defining a queue at runtime.

WebSphere MQ JMS accepts a topic URI for the string parameter of createTopic(), as shown in the following example:

```
Topic topic = session.createTopic( "topic://Sport/Football/Spurs/Results" );
```

Although the createTopic() method is in the JMS specification, the format of the string argument is provider specific. Therefore, using this method can make your code non-portable.

Other ways of creating a Topic object at runtime are as follows:

Using MQTopic(..)

This way requires a reference to the WebSphere MQ JMS implementation of the Topic interface, and therefore renders the code non-portable.

The constructor takes one argument, which must be a URI. For a WebSphere MQ JMS topic, this must be of the form:

```
topic://TopicName[?property=value[&property=value]*]
```

For example, the following code creates a topic for nonpersistent messages with a priority of 5:

```
// Create a Topic using the one-argument MQTopic constructor
String tSpec = "Sport/Football/Spurs/Results?persistence=1&priority=5";
Topic rtTopic = new MQTopic( "topic://" + tSpec );
```

Using MQTopic(), then setBaseTopicName(..)

This way uses the default MQTopic constructor, and therefore renders the code non-portable. Here is an example:

```
// Create a Topic using the default MQTopic constructor
Topic rtTopic = new MQTopic();
.
.
.
// Set the object properties using the setter methods
((MQTopic)rtTopic).setBaseTopicName( "Sport/Football/Spurs/Results" );
((MQTopic)rtTopic).setPersistence(1);
((MQTopic)rtTopic).setPriority(5);
```

Using session.createTemporaryTopic()

A temporary topic is created by a session, and only message consumers created by the same session can consume messages from the topic. A TemporaryTopic object is created as follows:

```
// Create a TemporaryTopic using the session factory method
Topic rtTopic = session.createTemporaryTopic();
```

Sending a message

An application sends messages using a MessageProducer object. A MessageProducer object is normally created for a specific destination so that all messages sent using that message producer are sent to the same destination. The destination is specified using either a Queue or a Topic object. Queue and Topic objects can be created at runtime, or built and stored in a JNDI namespace, as described in “Destinations” on page 355.

After a Queue or a Topic object is obtained, an application can pass the object to the createProducer() method to create a MessageProducer object, as shown in the following example:

```
MessageProducer messageProducer = session.createProducer(ioDestination);
```

The parameter ioDestination can be either a Queue or a Topic object.

The application can then use the send() method on the MessageProducer object to send messages. Here is an example:

```
messageProducer.send(outMessage);
```

You can use the send() method to send messages in either messaging domain. The nature of the destination determines the actual domain used. However, TopicPublisher, the sub-interface for MessageProducer that is specific to the publish/subscribe domain, uses a publish() method instead.

An application can create a MessageProducer object with no specified destination. In this case, the application must specify the destination in the send() method.

Note: If an application sends a message within a transaction, the message is not delivered to its destination until the transaction is committed. This means that an application cannot send a message and receive a reply to the message within the same transaction.

Message types

JMS provides several message types, each of which embodies some knowledge of its content. To avoid referring to the provider specific class names for the message types, methods for creating messages are provided on a Session object.

For example, a text message can be created in the following manner:

```
System.out.println( "Creating a TextMessage" );
TextMessage outMessage = session.createTextMessage();
System.out.println("Adding Text");
outMessage.setText(outString);
```

Here are the message types you can use:

- BytesMessage
- MapMessage
- ObjectMessage
- StreamMessage
- TextMessage

Details of these types are in Chapter 15, “Package com.ibm.jms,” on page 421.

Receiving a message

An application receives messages using a MessageConsumer object. The application creates a MessageConsumer object by using the createConsumer() method on a Session object. This method has a destination parameter that defines where the messages are received from. See “Destinations” on page 355 for details of how to create a destination, which is either a Queue or a Topic object.

In the point-to-point domain, the following code creates a MessageConsumer object and then uses the object to receive a message:

```
MessageConsumer messageConsumer = session.createConsumer(ioQueue);
Message inMessage = messageConsumer.receive(1000);
```

The parameter on the receive() call is a timeout in milliseconds. This parameter defines how long the method must wait if no message is available immediately. You can omit this parameter; in which case, the call blocks until a suitable message arrives. If you do not want any delay, use the receiveNoWait() method.

The receive() methods return a message of the appropriate type. For example, suppose a text message is put on a queue. When the message is received, the object that is returned is an instance of TextMessage.

To extract the content from the body of the message, it is necessary to cast from the generic Message class (which is the declared return type of the receive() methods) to the more specific subclass, such as TextMessage. If the received message type is not known, you can use the `instanceof` operator to determine which type it is. It is good practice always to test the message class before casting so that unexpected errors can be handled gracefully.

The following code uses the `instanceof` operator and shows how to extract the content of a text message:

```
if (inMessage instanceof TextMessage) {
  String replyString = ((TextMessage) inMessage).getText();
  .
  .
  .
} else {
  // Print error message if Message was not a TextMessage.
  System.out.println("Reply message was not a TextMessage");
}
```

Note: If an application sends a message within a transaction, the message is not delivered to its destination until the transaction is committed. This means that an application cannot send a message and receive a reply to the message within the same transaction.

JMS provides two types of message consumer:

Nondurable message consumer
A nondurable message consumer receives messages from its chosen destination only if the messages are available while the consumer is active.

In the point-to-point domain, whether a consumer receives messages that are sent while the consumer is inactive depends on how WebSphere MQ is configured to support that consumer. In the publish/subscribe domain, a consumer does not receive messages that are sent while the consumer is inactive.

Durable topic subscriber
A durable topic subscriber receives all the messages sent to a destination, including those sent while the consumer is inactive.

The following sections describe how to create a durable topic subscriber, and how to configure WebSphere MQ and the broker to support either type of message consumer.

Creating durable topic subscribers

You cannot create a durable topic subscriber if the transport type is direct.

Durable topic subscribers are used when an application needs to receive messages that are published even while the application is inactive.

Creating a durable topic subscriber is similar to creating a nondurable message consumer, but you must also provide a name that identifies the durable subscription, as in the following example:

```
// Create a durable subscriber, supplying a uniquely-identifying name
TopicSubscriber sub = session.createDurableSubscriber( topic, "D_SUB_000001" );
```

The session used to create a durable topic subscriber must have an associated client identifier. The name that identifies a durable subscription must be unique only within the client identifier, and therefore the client identifier forms part of the full, unique identifier of the durable subscription. To reuse a durable subscription that was created previously, an application must create a durable topic subscriber using a session with the same client identifier as that associated with the durable subscription.

The client identifier associated with a session is the same as that associated with the connection that is used to create the session. The client identifier can be specified by setting the CLIENTID property of the ConnectionFactory object. Alternatively, an application can specify the client identifier by calling the setClientID() method of the Connection object. For more information about client identifiers and their relationship with durable topic subscribers and durable subscriptions, see the *Java Message Service Specification, Version 1.1*.

A durable topic subscriber is created for the queue manager specified by the QMANAGER property of the ConnectionFactory object. If there is a subsequent attempt to create a durable topic subscriber with the same name for a different queue manager, a new and completely independent durable topic subscriber is returned.

Nondurable message consumers in the publish/subscribe domain automatically deregister themselves when their close() method is called, or when they fall out of scope. However, if you want to terminate a durable subscription, you must explicitly notify the broker. To do this, use the unsubscribe() method of the session and pass in the name that identifies the durable subscription:

```
// Unsubscribe the durable subscriber created above
session.unsubscribe( "D_SUB_000001" );
```

Message selectors

JMS allows an application to specify that only messages that satisfy certain criteria are returned by successive receive() calls. When creating a MessageConsumer object, you can provide a string that contains an SQL (Structured Query Language) expression, which determines which messages are retrieved. This SQL expression is called a *selector*. The selector can refer to fields in the JMS message header as well as fields in the message properties (these are effectively application defined header fields). Details of the header field names, as well as the syntax for an SQL selector, are in Chapter 13, “JMS messages,” on page 379.

The following example shows how to select messages based on a user defined property called myProp:

```
messageConsumer = session.createConsumer(ioQueue, "myProp = 'blue'");
```

Note: The JMS specification does not permit the selector associated with a message consumer to be changed. After a message consumer is created, the selector is fixed for the lifetime of that consumer. This means that, if you require different selectors, you must create new message consumers.

You can control whether the JMS client or the broker performs message filtering in the publish/subscribe domain by setting the MSGSELECTION property on the ConnectionFactory object. If the broker is capable of performing message selection, it is generally preferable to let the broker do it because it reduces the amount of

work done by the client. However, if the broker is very heavily loaded, it might be preferable to let the client perform message selection instead.

Suppressing local publications

You can create a message consumer that ignores publications published on the consumer's own connection. To do this, set the third parameter on the createConsumer() call to `true`, as shown in the following example:

```
// Create a nondurable message consumer with the noLocal option set
MessageConsumer con = session.createConsumer( topic, null, true );
```

The example that follows shows how to create a durable topic subscriber that applies a selector and ignores local publications:

```
// Create a durable, noLocal subscriber with a selector applied
String selector = "company = 'IBM'";
TopicSubscriber sub = session.createDurableSubscriber( topic, "D_SUB_000001",
                                                       selector, true );
```

Configuring the consumer queue

You cannot configure a consumer queue if the transport type is direct.

In the publish/subscribe messaging domain, you can configure message consumers in two ways:

- The multiple queue approach.

  Each consumer has its own exclusive queue and retrieves all its messages from this queue. JMS creates a new queue for each consumer.
- The shared queue approach.

  Each consumer retrieves its messages from a queue that is shared with other consumers. This approach requires only one queue to serve multiple consumers. It is the default approach used with WebSphere MQ JMS.

You can choose which approach to use, and configure which queues to use.

In general, there is a modest performance advantage if you use the shared queue approach. For systems with a high throughput, there are also large system management and administrative advantages because of the significant reduction in the number of queues required.

In some situations, however, there are good reasons for using the multiple queue approach:

- In theory, you can store more messages.

  There is an upper limit to the number of messages that a WebSphere MQ queue can hold and so, in the shared queue approach, the total number of messages for all the message consumers that share the queue cannot exceed this limit. This issue is more significant for durable topic subscribers, because the lifetime of a durable topic subscriber is usually much longer than that of a nondurable message consumer. Therefore, more messages might accumulate for a durable subscriber.
- The WebSphere MQ administration of consumer queues is easier.

  For certain applications, an administrator might want to monitor the state and depth of particular consumer queues. This task is much simpler when each consumer has its own queue.

Default configuration

The default WebSphere MQ JMS configuration for the publish/subscribe domain uses the following shared consumer queues:

- SYSTEM.JMS.ND.SUBSCRIBER.QUEUE for nondurable message consumers
- SYSTEM.JMS.D.SUBSCRIBER.QUEUE for durable topic subscribers

These are created for you when you run the MQJMS_PSQ.MQSC script.

If required, you can specify alternative WebSphere MQ queues. You can also change the configuration to use the multiple queue approach.

Configuring nondurable message consumers

You can specify the name of the consumer queue for nondurable message consumers in either of the following ways:

- Use the WebSphere MQ JMS administration tool to set the BROKERSUBQ property.
- Use the setBrokerSubQueue() method in your application.

The queue name you provide must start with the following characters:

SYSTEM.JMS.ND.

To use the shared queue approach, specify the complete name of the shared queue. The queue must exist before you can create a subscription.

To use the multiple queue approach, specify a queue name that ends with an asterisk (*). Subsequently, when an application creates a nondurable message consumer specifying this queue name, WebSphere MQ JMS creates a temporary dynamic queue for exclusive use by that consumer. With the multiple queue approach, therefore, all the required queues are created dynamically.

If you use the multiple queue approach, you cannot specify the complete name of a queue, only a prefix. This allows you to create different domains of consumer queues. For example, you can use:

SYSTEM.JMS.ND.MYDOMAIN.*

The characters that precede the asterisk (*) are used as the prefix, so that all dynamic queues for nondurable message consumers specifying this prefix have queue names that start with SYSTEM.JMS.ND.MYDOMAIN.

Configuring durable topic subscribers

As stated previously, there might still be good reasons to use the multiple queue approach for durable topic subscribers. Durable topic subscribers are likely to have a longer life span, and so it is possible for a large number of messages for a durable subscriber to accumulate on a queue.

The name of the consumer queue for a durable topic subscriber is a property of a Topic object. This allows you to specify a number of different consumer queue names without having to create multiple objects starting from a ConnectionFactory object.

You can specify the name of the consumer queue for durable topic subscribers in either of the following ways:

- Use the WebSphere MQ JMS administration tool to set the BROKERDURSUBQ property.
- Use the setBrokerDurSubQueue() method in your application.

The queue name you provide must start with the following characters:

SYSTEM.JMS.D.

To use the shared queue approach, specify the complete name of the shared queue. The queue must exist before you can create a subscription.

To use the multiple queue approach, specify a queue name that ends with an asterisk (*). Subsequently, when an application creates a durable topic subscriber specifying this queue name, WebSphere MQ JMS creates a permanent dynamic queue for exclusive use by that subscriber. With the multiple queue approach, therefore, all the required queues are created dynamically.

Here is an example of using the multiple queue approach:

```
// Set the MQTopic durable subscriber queue name using
// the multi-queue approach
sportsTopic.setBrokerDurSubQueue("SYSTEM.JMS.D.FOOTBALL.*");
```

After the Topic object is initialized, it can be passed into the createDurableSubscriber() method of a Session object to create a durable topic subscriber:

```
// Create a durable subscriber using our earlier Topic
TopicSubscriber sub = session.createDurableSubscriber(sportsTopic,
                                                       "D_SUB_SPORT_001");
```

If you use the multiple queue approach, you cannot specify the complete name of a queue, only a prefix. This allows you to create different domains of consumer queues. For example, you can use:

SYSTEM.JMS.D.MYDOMAIN.*

The characters that precede the asterisk (*) are used as the prefix, so that all dynamic queues for durable topic subscribers specifying this prefix have queue names that start with SYSTEM.JMS.D.MYDOMAIN.

You cannot change the consumer queue of a durable topic subscriber. If, for example, you want to move from the multiple queue approach to the single queue approach, you must first delete the old subscriber using the unsubscribe() method and then create a new subscriber. Deleting the old subscriber also deletes any unconsumed messages for that subscriber.

Subscription stores

A subscription store is not used if the transport type is direct.

When an application creates a message consumer in the publish/subscribe domain, information about the subscription is created at the same time. WebSphere MQ JMS maintains a persistent store of subscription information called a *subscription store*. A subscription store is used to reopen durable topic subscribers and to clean up after a nondurable message consumer fails. A subscription store can be managed by the local queue manager or by the publish/subscribe broker.

The SUBSTORE property of a ConnectionFactory object determines the location of a subscription store. SUBSTORE has three possible values:

SUBSTORE(QUEUE)
Subscription information is stored in the queues, SYSTEM.JMS.ADMIN.QUEUE and SYSTEM.JMS.PS.STATUS.QUEUE, on the local queue manager.

WebSphere MQ JMS maintains an extra connection to each queue manager used by subscribers. Each connection is used by a long running transaction that detects when a subscriber loses its connection to the queue manager and cleans up after the subscriber. In a busy system, a long running transaction might cause the queue manager log to fill up resulting in errors from both the queue manager and the applications connected to it.

If you experience these problems, your system administrator can add extra log files or data sets to the queue manager. Alternatively, you can reduce the value of the STATREFRESHINT property of the ConnectionFactory object. This causes the long running transaction to be refreshed more frequently, allowing the queue manager log to reset itself before it becomes full.

After a nondurable message consumer fails, the following occurs:

- Subscription information related to the failed consumer remains on the two queues implementing the subscription store. This information can be removed by a cleanup utility supplied with WebSphere MQ JMS. See "Consumer cleanup utility for the publish/subscribe domain" on page 367 for more information.
- Messages continue to be delivered to the consumer until the cleanup utility runs.

This option is provided for compatibility with versions of MQSeries JMS.

SUBSTORE(BROKER)
Subscription information is stored by the publish/subscribe broker used by the application, not in WebSphere MQ queues. This means that, if a JMS client fails, the broker can clean up the resources associated with the JMS client without having to wait for the JMS client to reconnect.

If a nondurable message consumer fails, the subscription is de-registered from the broker as soon as possible. In response to the de-registration, the broker puts a report message on the queue, SYSTEM.JMS.REPORT.QUEUE. This message is used to clean up after the failed consumer.

If you use this option, a separate cleanup thread is run in the background. By default, the cleanup utility runs once every 10 minutes, but you can change this interval by setting the CLEANUPINT property of the ConnectionFactory object. To customize the actions performed by the cleanup utility, use the CLEANUP property of the ConnectionFactory object. For more information about how the cleanup utility works, see "Consumer cleanup utility for the publish/subscribe domain" on page 367.

SUBSTORE(MIGRATE)
This is the default value.

This option dynamically selects a queue based or a broker based subscription store depending on the release levels of WebSphere MQ and the publish/subscribe broker that are installed. If both WebSphere MQ and the broker are capable of supporting the SUBSTORE(BROKER) option, this option behaves like the SUBSTORE(BROKER) option; otherwise, it behaves like the SUBSTORE(QUEUE) option.

If this option behaves like the SUBSTORE(BROKER) option, the option additionally migrates durable subscription information from the queue based subscription store to the broker based store. This migration occurs the first time a durable subscription is opened when both WebSphere MQ and the broker are capable of supporting a broker based subscription store.

Only information related to the subscription being opened is migrated. Information related to other subscriptions is left in the queue based subscription store. This option therefore provides an easy migration path from older versions of WebSphere MQ JMS, WebSphere MQ, and the publish/subscribe broker.

Migration and coexistence considerations

Except when the SUBSTORE(MIGRATE) option is used, a queue based subscription store and a broker based subscription store are entirely independent.

A durable subscription is created in the subscription store specified by the ConnectionFactory object. If there is a subsequent attempt to create a durable subscription with the same name and ClientID, but with the other subscription store, a new durable subscription is created.

When a connection uses the SUBSTORE(MIGRATE) option, subscription information is automatically migrated from the queue based subscription store to the broker based subscription store when the application calls the createDurableSubscriber() method. If a durable subscription with a matching name and ClientID already exists in the broker based subscription store, the migration cannot complete and an exception is thrown by the createDurableSubscriber() call.

After a subscription is migrated, it is important not to access the subscription from an application using an older version of WebSphere MQ JMS, or from an application running with the SUBSTORE(QUEUE) option. Doing either of these creates a subscription in the queue based subscription store and prevents an application running with the SUBSTORE(MIGRATE) option from using the subscription.

To recover from this situation if it occurs, run your application with the SUBSTORE(BROKER) option, or unsubscribe from the subscription that is held in the queue based subscription store.

JMS persistent messages

A WebSphere MQ queue has an attribute called *NonPersistentMessageClass*. The value of this attribute determines whether nonpersistent messages on the queue are discarded when the queue manager restarts.

You can set the attribute for a local queue by using the WebSphere MQ Script (MQSC) command, DEFINE QLOCAL, with either of the following parameters:

NPMCLASS(NORMAL)
Nonpersistent messages on the queue are discarded when the queue manager restarts. This is the default value.

NPMCLASS(HIGH)
Nonpersistent messages on the queue are not discarded when the queue manager restarts following a quiesced or immediate shutdown. Nonpersistent messages might be discarded, however, following a preemptive shutdown or a failure.

This section describes how WebSphere MQ JMS applications can use this queue attribute to provide better performance for JMS persistent messages.

The PERSISTENCE property of a Queue or Topic object can have the value HIGH. You can use the WebSphere MQ JMS administration tool to set this value, or an

application can call the Destination.setPersistence() method passing the value JMSC.MQJMS_PER_HIGH as a parameter.

If an application sends a JMS persistent message or a JMS nonpersistent message to a destination whose PERSISTENCE property has the value HIGH, and the underlying WebSphere MQ queue is set to NPMCLASS(HIGH), the message is put on the queue as a WebSphere MQ nonpersistent message. If the PERSISTENCE property of the destination does not have the value HIGH, or if the underlying queue is set to NPMCLASS(NORMAL), a JMS persistent message is put on the queue as a WebSphere MQ persistent message, and a JMS nonpersistent message is put on the queue as a WebSphere MQ nonpersistent message.

If a JMS persistent message is put on a queue as a WebSphere MQ nonpersistent message, and you want to ensure that the message is not discarded following a quiesced or immediate shutdown of a queue manager, all queues through which the message might be routed must be set to NPMCLASS(HIGH). In the publish/subscribe domain, these queues include subscriber queues. As an aid to enforcing this configuration, the WebSphere MQ JMS client throws an InvalidDestinationException if an application tries to create a message consumer for a destination whose PERSISTENCE property has the value HIGH and the underlying WebSphere MQ queue is set to NPMCLASS(NORMAL).

Setting the PERSISTENCE property of a destination to HIGH has no effect on how a message is received from that destination. A message sent as a JMS persistent message is received as a JMS persistent message, and a message sent as a JMS nonpersistent message is received as a JMS nonpersistent message.

When an application sends the first message to a destination whose PERSISTENCE property has the value HIGH, or when an application creates the first message consumer for a destination whose PERSISTENCE property has the value HIGH, the WebSphere MQ JMS client issues an MQINQ call to determine whether NPMCLASS(HIGH) is set on the underlying WebSphere MQ queue. The application must therefore have the authority to inquire on the queue. In addition, the WebSphere MQ JMS client preserves the result of the MQINQ call until the destination is deleted, and does not issue more MQINQ calls. Therefore, if you change the NPMCLASS setting on the underlying queue while the application is still using the destination, the WebSphere MQ JMS client does not notice the new setting.

By allowing JMS persistent messages to be put on WebSphere MQ queues as WebSphere MQ nonpersistent messages, you are gaining performance at the expense of some reliability. If you require maximum reliability for JMS persistent messages, do not send the messages to a destination whose PERSISTENCE property has the value HIGH.

Asynchronous delivery

An application can call the MessageConsumer.receive() method to receive messages. As an alternative, an application can register a method that is called automatically when a suitable message is available. This is called *asynchronous delivery* of messages. The following code illustrates the mechanism:

```
import javax.jms.*;

public class MyClass implements MessageListener
{
  // The method that will be called by JMS when a message
  // is available.
```

```
  public void onMessage(Message message)
  {
    System.out.println("message is "+message);

    // application specific processing here
    .
    .
    .
  }
}
 .
 .
 .
  // In Main program (possibly of some other class)
  MyClass listener = new MyClass();
  messageConsumer.setMessageListener(listener);

  // main program can now continue with other application specific
  // behavior.
```

Note: Using asynchronous delivery with a message consumer marks the entire session as using asynchronous delivery. An application cannot call the receive() methods of a message consumer if the message consumer is associated with a session that is using asynchronous delivery.

Consumer cleanup utility for the publish/subscribe domain

To avoid the problems associated with message consumer objects in the publish/subscribe domain not closing gracefully, WebSphere MQ JMS supplies a consumer cleanup utility that attempts to clean up the resources associated with a consumer that has failed. This utility runs in the background and does not affect other WebSphere MQ JMS operations. If the utility detects a large number of problems associated with a given queue manager, you might see some performance degradation while resources are being cleaned up.

Note: Whenever possible, close all message consumer objects gracefully to avoid an accumulation of these types of problems.

If applications use the domain independent classes, the cleanup utility is invoked only if the applications perform publish/subscribe operations, such as creating a Topic object, or creating a MessageConsumer object with a Topic object retrieved from a JNDI namespace. This is to prevent the cleanup utility from being invoked in an environment in which applications are performing only point-to-point operations.

The exact behavior of the cleanup utility depends on where the subscription store is located:

Queue based subscription store

For a queue based subscription store, the cleanup utility runs against a queue manager when the first Connection object to use that queue manager initializes. If all the Connection objects that use a given queue manager close, the utility runs again only when the next Connection object to use that queue manager initializes.

The cleanup utility uses the information in the queues, SYSTEM.JMS.ADMIN.QUEUE and SYSTEM.JMS.PS.STATUS.QUEUE, to detect nondurable message consumers that have failed previously. If it finds a failed consumer, the utility cleans up the resources associated with

the consumer by de-registering the consumer from the broker and deleting its consumer queue, provided it is not a shared queue, and any unconsumed messages on the queue.

Broker based subscription store

For a broker based subscription store, the cleanup utility runs at regular intervals on a background thread while there is at least one Connection object that uses a given queue manager. One cleanup thread is created for each queue manager for which a Connection object exists within the JVM.

The cleanup utility uses information in the queue, SYSTEM.JMS.REPORT.QUEUE (the messages in this queue are typically report messages from the publish/subscribe broker), to perform any necessary cleanup. This might involve deleting consumer queues and unconsumed messages that are no longer required.

Two properties of a ConnectionFactory object control the behavior of the cleanup thread: CLEANUPINT and CLEANUP. CLEANUPINT determines how often, in milliseconds, the cleanup utility is run against any given queue manager. CLEANUP has four possible values:

CLEANUP(SAFE)

This is the default value.

The cleanup thread tries to delete any consumer queues and unconsumed messages that are no longer required. This mode of cleanup does not interfere with the operation of other JMS applications.

CLEANUP(STRONG)

The cleanup thread performs like the CLEANUP(SAFE) option, but it also deletes any messages on the queue, SYSTEM.JMS.REPORT.QUEUE, that it does not recognize.

This mode of cleanup can interfere with the operation of JMS applications running with later versions of WebSphere MQ JMS. If multiple JMS applications are using the same queue manager, but using different versions of WebSphere MQ JMS, only applications using the most recent version of WebSphere MQ JMS must use this option.

CLEANUP(NONE)

In this special mode, no cleanup is performed, and consumer queues and unconsumed messages that are no longer required are not deleted.

This option can be useful if the application and the queue manager are on a different systems, especially if the application only sends messages and does not receive them. At some time, however, cleanup must be initiated to delete consumer queues and unconsumed messages that are no longer required. This can be done by a JMS application that uses a ConnectionFactory object with the property CLEANUP(SAFE) or CLEANUP(STRONG), or by using the manual cleanup utility, which is described in “Manual cleanup” on page 369.

CLEANUP(ASPROP)

The mode of cleanup is determined by the system property com.ibm.mq.jms.cleanup, which is queried when the JVM starts.

This property can be set on the Java command line by using the -D option. Its value can be SAFE, STRONG, or NONE. Any other value causes an exception. If the property not set, its value defaults to SAFE.

This option allows you to change the mode of cleanup within an entire JVM without having to update every ConnectionFactory object. This is useful for applications or application servers that use multiple ConnectionFactory objects.

If multiple Connection objects for the same queue manager exist within a JVM, but the Connection objects use different values for the CLEANUPINT and CLEANUP properties, the following rules determine the behavior of the cleanup utility:

1. If a Connection object using the CLEANUP(NONE) option fails, cleanup does not run. The cleanup thread eventually runs, however, if another Connection object is using the CLEANUP(SAFE) or CLEANUP(STRONG) option.
2. If any Connection object is using the CLEANUP(STRONG) option, the cleanup thread operates in STRONG mode. Otherwise, if any Connection object is using the CLEANUP(SAFE) option, the cleanup thread operates in SAFE mode. Otherwise, there is no cleanup thread.
3. The cleanup utility runs at intervals determined by the smallest value of the CLEANUPINT property of those Connections that are using the CLEANUP(SAFE) or CLEANUP(STRONG) option.

Manual cleanup

If you use a broker based subscription store, you can operate the cleanup utility manually from the command line. Here is the syntax of the command:

For a bindings connection:

```
Cleanup [-m <qmgr>] [-r <interval>]
          [SAFE | STRONG | FORCE | NONDUR] [-t]
```

For a client connection:

```
Cleanup -client [-m <qmgr>] -host <hostname> [-port <port>]
         [-channel <channel>] [-r <interval>]
         [SAFE | STRONG | FORCE | NONDUR] [-t]
```

The parameters of the command are as follows:

- `qmgr`, `hostname`, `port`, and `channel` enable the cleanup utility to connect to a queue manager.
- `-r` sets the interval, in minutes, between each run of the cleanup utility. If the parameter is not set, the cleanup utility runs once only.
- `-t` enables tracing. The output is sent to the file mqjms.trc.
- `SAFE`, `STRONG`, `FORCE`, or `NONDUR` sets the cleanup level as follows:
  - SAFE and STRONG behave like the CLEANUP(SAFE) and CLEANUP(STRONG) modes discussed in “Consumer cleanup utility for the publish/subscribe domain” on page 367.
  - FORCE behaves like STRONG mode. But, whereas STRONG mode leaves any messages that cannot be processed on the queue, SYSTEM.JMS.REPORT.QUEUE, FORCE mode deletes all the messages even if it encounters an error during processing.

Warning: This is a dangerous mode that can leave an inconsistent state between the queue manager and the broker. You cannot run the cleanup utility in this mode while any &mqjms; &pubsub; applications are connected to the queue manager. If you try to do so, the cleanup utility ends.

– NONDUR behaves like FORCE mode but, in addition, this mode deletes all the messages on queues whose names begin with the characters SYSTEM.JMS.ND. To do this successfully, the command server of the queue manager must be running.

Cleanup from within a program

You can use a programming interface to invoke the cleanup utility that is used with a broker based subscription store. Instances of the class com.ibm.mq.jms.Cleanup have getter and setter methods for each of the properties that are used to connect to a queue manager, and also for the cleanup level and cleanup interval. It exposes two additional methods:

cleanup()
Executes the cleanup utility once only.

run() Runs cleanup at intervals determined by cleanup interval property.

This class allows complete customization of the publish/subscribe cleanup utility, but it is intended for use by system administration programs rather than application programs.

For more details, see “Cleanup” on page 470.

Closing down

Garbage collection alone cannot release all WebSphere MQ resources in a timely manner, especially if an application creates many short lived JMS objects at the session level or lower. It is therefore important for an application to call the appropriate close() method to close a Connection, Session, MessageConsumer, or MessageProducer object when it is no longer required.

Java Virtual Machine hangs at shutdown

If an application using WebSphere MQ JMS finishes without calling Connection.close(), some JVMs appear to hang. If this problems occurs, you can end the JVM by entering Ctrl-C. To avoid the problem in the future, consider modifying the application to include a call to Connection.close().

Handling errors

Any runtime errors in a JMS application are reported by exceptions. The majority of JMS methods throw a JMSException to indicate an error. It is good programming practice to catch these exceptions and display them on a suitable output device.

Each JMS exception encapsulates a message identifier and some associated message text, which describes the error that has occurred. The message identifier has the format MQJMS*nnnn*, where *nnnn* is an integer in the range 0000 to 9999. For each message identifier, there is a corresponding field in the MQJMS_Messages class. In the definition of the MQJMS_Messages class, the description of the field contains an explanation of why the error occurred and a suggested user response.

To determine which field in the MQJMS_Messages class corresponds to a given message identifier, you must look in the specification of the WebSphere MQ JMS API that is supplied as HTML pages generated by the Javadoc tool. The MQJMS_Messages section of the Constant Field Values page contains a table that lists each message identifier and its corresponding field in the MQJMS_Messages class.

A JMSException can contain a further exception embedded within it. For JMS, this can be a valuable way to pass important information about the error from the underlying transport. In the case of WebSphere MQ JMS, an MQException is thrown in WebSphere MQ base Java whenever an error occurs in WebSphere MQ, and this exception is usually included as the embedded exception in a JMSException.

The implementation of JMSException does not include the embedded exception in the output of its toString() method. Therefore, you must check explicitly for an embedded exception and print it out, as shown in the following example:

```
try {
  .
  . code that might throw a JMSException
  .
} catch (JMSException je) {
  System.err.println("caught "+je);
  Exception e = je.getLinkedException();
  if (e != null) {
    System.err.println("linked exception: "+e);
  }
}
```

Exception listener

For asynchronous message delivery, the application code cannot catch exceptions raised by failures to receive messages. This is because the application code does not make explicit calls to receive() methods. To cope with this situation, you can register an ExceptionListener, which is an instance of a class that implements the onException() method. When a serious error occurs, this method is called with the JMSException passed as its only parameter. Further details are in Sun's JMS documentation.

Handling broker reports

The WebSphere MQ JMS implementation uses report messages from the broker to confirm whether registration and de-registration requests have been successful. These report messages are sent to the queue, SYSTEM.JMS.REPORT.QUEUE, on the local queue manager and are normally consumed by the WebSphere MQ JMS. Under some error conditions, however, they might remain on the queue.

WebSphere MQ JMS supplies a Java application, PSReportDump, which dumps the contents of the queue, SYSTEM.JMS.REPORT.QUEUE, in plain text format. The information in the dump can be analyzed by you or by IBM support staff. You can also use the application to delete all the messages in the queue after a problem is diagnosed or fixed.

The compiled form of the application is in the *<MQ_JAVA_INSTALL_PATH>*/bin directory. To start the application, change to this directory and use the following command:

```
java -Djava.library.path=library_path
     PSReportDump [-m queueManager] [-clear]
```

where *library_path* is the path to the WebSphere MQ Java libraries (see "The WebSphere MQ Java libraries" on page 10), and:

-m *queueManager*
Specifies the name of the queue manager to use

-clear Causes all the messages on the queue to be deleted after their contents have been dumped

Attention: Do not use this option if you are using a broker based subscription store. Instead, run the manual cleanup utility in FORCE mode.

The output from the application is sent to the screen, or you can redirect it to a file.

Other considerations

If a large number of JMS clients connect directly to a broker running on Windows, and the connections happen almost simultaneously, a java.net.BindException address in use exception might be thrown in response to a request to connect to the broker. You can try to avoid this by catching the exception and retrying, or by pacing the connections.

Using channel exits

A WebSphere MQ JMS application can use channel security, send, and receive exits on the MQI channel that starts when the application connects to a queue manager. The application can use exits written in Java, C, or C++. The application can also use a sequence of send or receive exits that are run in succession.

Only an application that connects to a queue manager in client mode can use channel exits. An application cannot use channel exits if it connects in bindings mode.

The SENDEXIT property of a ConnectionFactory object specifies the send exit, or exits, used by a connection. The value of the property is a string that comprises one or more items separated by commas. Each item identifies a send exit in one of the following ways:

- The name of a class that implements the WebSphere MQ base Java interface, MQSendExit (for a send exit written in Java)
- A string in the format *libraryName*(*entryPointName*) (for a send exit not written in Java)

You can set the SENDEXIT property by using the WebSphere MQ JMS administration tool, or an application can set the property by calling the setSendExit() method.

In a similar way, the RECEXIT property of a ConnectionFactory object specifies the receive exit, or exits, used by a connection, and the SECEXIT property specifies the security exit used by a connection.

The SENDEXITINIT property of a ConnectionFactory object specifies the user data that is passed to each send exit when it is called. The value of the property is a string that comprises one or more items of user data separated by commas. The position of each item of user data within the string determines which send exit, in a sequence of send exits, the user data is passed to. For example, the first item of user data in the string is passed to the first send exit in a sequence of send exits.

You can set the SENDEXITINIT property by using the WebSphere MQ JMS administration tool, or an application can set the property by calling the setSendExitInit() method.

In a similar way, the RECEXITINIT property of a ConnectionFactory object specifies the user data that is passed to each receive exit, and the SECEXITINIT property specifies the user data passed to a security exit.

Note the following rules when specifying user data that is passed to channel exits:

- If the number of items of user data in a string is more than the number of exits in a sequence, the excess items of user data are ignored.
- If the number of items of user data in a string is less than the number of exits in a sequence, each unspecified item of user data is set to an empty string. Two commas in succession within a string, or a comma at the beginning of a string, also denotes an unspecified item of user data.

On i5/OS, no user data can be passed to channel exit programs that are written in C or C++.

For information about how to write a channel exit in Java, and how to make sure that WebSphere MQ JMS can locate the JAR or class files containing channel exit classes, see "Using channel exits" on page 77.

For information about how to write a channel exit in C or C++, see *WebSphere MQ Intercommunication*. You must store channel exit programs that are not written in Java in the directory shown in Table 13 on page 79.

If your application uses a client channel definition table to connect to a queue manager, see "Using a client channel definition table" on page 351.

Using Secure Sockets Layer (SSL)

WebSphere MQ base Java client applications and WebSphere MQ JMS connections using TRANSPORT(CLIENT) support Secure Sockets Layer (SSL) encryption. SSL provides communication encryption, authentication, and message integrity. It is typically used to secure communications between any two peers on the Internet or within an intranet.

WebSphere MQ classes for Java uses Java Secure Socket Extension (JSSE) to handle SSL encryption, and so requires a JSSE provider. J2SE v1.4 JVMs have a JSSE provider built in. Details of how to manage and store certificates can vary from provider to provider. For information about this, refer to your JSSE provider's documentation.

This section assumes that your JSSE provider is correctly installed and configured, and that suitable certificates have been installed and made available to your JSSE provider.

If your WebSphere MQ JMS client application uses a client channel definition table to connect to a queue manager, see "Using a client channel definition table" on page 351.

SSL administrative properties

This section introduces the SSL administrative properties, as follows:

SSLCIPHERSUITE object property

To enable SSL encryption on a ConnectionFactory object, use JMSAdmin to set the SSLCIPHERSUITE property to a CipherSuite supported by your JSSE provider. This must match the CipherSpec set on the target channel. However, CipherSuites are distinct from CipherSpecs and so have different names. Appendix D, "SSL CipherSpecs and CipherSuites," on page 645 contains a table mapping the CipherSpecs supported by WebSphere MQ to their equivalent CipherSuites as known to JSSE. Additionally, the named CipherSuite must be supported by your JSSE provider. For more information about CipherSpecs and CipherSuites with WebSphere MQ, see *WebSphere MQ Security*.

For example, to set up a ConnectionFactory object that can be used to create a connection over an SSL enabled MQI channel with a CipherSpec of RC4_MD5_EXPORT, issue the following command to JMSAdmin:

```
ALTER CF(my.cf) SSLCIPHERSUITE(SSL_RSA_EXPORT_WITH_RC4_40_MD5)
```

This can also be set from an application, using the setSSLCipherSuite() method on an MQConnectionFactory object.

For convenience, if a CipherSpec is specified on the SSLCIPHERSUITE property, JMSAdmin attempts to map the CipherSpec to an appropriate CipherSuite and issues a warning. This attempt to map is not made if the property is specified by an application.

SSLFIPSREQUIRED object property

If you require a connection to use a CipherSuite that is supported by the IBM Java JSSE FIPS provider (IBMJSSEFIPS), set the SSLFIPSREQUIRED property of the connection factory to YES. The default value of this property is NO, which means that a connection can use any CipherSuite that is not supported by IBMJSSEFIPS.

If an application uses more than one connection, the value of SSLFIPSREQUIRED that is used when the application creates the first connection determines the value that is used when the application creates any subsequent connection. This means that the value of the SSLFIPSREQUIRED property of the connection factory that is used to create a subsequent connection is ignored. You must restart the application if you want to use a different value of SSLFIPSREQUIRED.

An application can set this property by calling the setSSLFipsRequired() method of a ConnectionFactory object. The property is ignored if no CipherSuite is set.

SSLPEERNAME object property

A JMS application can ensure that it connects to the correct queue manager by specifying a distinguished name (DN) pattern. The connection succeeds only if the queue manager presents a DN that matches the pattern. For more details of the format of this pattern, refer to *WebSphere MQ Security* or the *WebSphere MQ Script (MQSC) Command Reference*.

The DN is set using the SSLPEERNAME property of a ConnectionFactory object. For example, the following JMSAdmin command sets a ConnectionFactory object to expect the queue manager to identify itself with a Common Name beginning with the characters `QMGR.`, and with at least two Organizational Unit names, the first of which must be IBM and the second WEBSPHERE:

```
ALTER CF(my.cf) SSLPEERNAME(CN=QMGR.*, OU=IBM, OU=WEBSPHERE)
```

Checking is not case sensitive and semicolons can be used in place of commas. This can also be set from an application using the setSSLPeerName() method on an MQConnectionFactory object. If this property is not set, no checking is performed on the Distinguished Name supplied by the queue manager. This property is ignored if no CipherSuite is set.

SSLCERTSTORES object property

It is common to use a certificate revocation list (CRL) to identify certificates that are no longer trusted. CRLs are typically hosted on LDAP servers. JMS allows an LDAP server to be specified for CRL checking under Java 2 v1.4 or later. The following JMSAdmin example directs JMS to use a CRL hosted on an LDAP server named crl1.ibm.com:

```
ALTER CF(my.cf) SSLCRL(ldap://crl1.ibm.com)
```

Note: To use a CertStore successfully with a CRL hosted on an LDAP server, make sure that your Java Software Development Kit (SDK) is compatible with the CRL. Some SDKs require that the CRL conforms to RFC 2587, which defines a schema for LDAP v2. Most LDAP v3 servers use RFC 2256 instead.

If your LDAP server is not running on the default port of 389, the port can be specified by appending a colon (:) and the port number to the host name. If the certificate presented by the queue manager is present in the CRL hosted on crl1.ibm.com, the connection does not complete. To avoid a single point of failure, JMS allows multiple LDAP servers to be supplied by supplying a list of LDAP servers delimited by the space character. Here is an example:

```
ALTER CF(my.cf) SSLCRL(ldap://crl1.ibm.com ldap://crl2.ibm.com)
```

When multiple LDAP servers are specified, JMS tries each one in turn until it finds a server with which it can successfully verify the queue manager's certificate. Each server must contain identical information.

A string in this format can be supplied by an application on the MQConnectionFactory.setSSLCertStores() method. Alternatively, the application can create one or more java.security.cert.CertStore objects, place these in a suitable Collection object, and supply this Collection object to the setSSLCertStores() method. In this way, the application can customize CRL checking. Refer to your JSSE documentation for details on constructing and using CertStore objects.

The certificate presented by the queue manager when a connection is being set up is validated as follows:

1. The first CertStore object in the Collection identified by sslCertStores is used to identify a CRL server.
2. An attempt is made to contact the CRL server.
3. If the attempt is successful, the server is searched for a match for the certificate.
   a. If the certificate is found to be revoked, the search process is over and the connection request fails with reason code MQRC_SSL_CERTIFICATE_REVOKED.
   b. If the certificate is not found, the search process is over and the connection is allowed to proceed.
4. If the attempt to contact the server is unsuccessful, the next CertStore object is used to identify a CRL server and the process repeats from step 2.

   If this was the last CertStore in the Collection, or if the Collection contains no CertStore objects, the search process has failed and the connection request fails with reason code MQRC_SSL_CERT_STORE_ERROR.

The Collection object determines the order in which CertStores are used.

If your application uses setSSLCertStores() to set a Collection of CertStore objects, the MQConnectionFactory can no longer be bound into a JNDI namespace. Attempting to do so causes an exception. If the sslCertStores property is not set, no revocation checking is performed on the certificate provided by the queue manager. This property is ignored if no CipherSuite is set.

SSLRESETCOUNT object property

This property represents the total number of bytes sent and received by a connection before the secret key that is used for encryption is renegotiated. The number of bytes sent is the number before encryption, and the number of bytes received is the number after decryption. The number of bytes also includes control information sent and received by the WebSphere MQ JMS client.

For example, to configure a ConnectionFactory object that can be used to create a connection over an SSL enabled MQI channel whose secret key is renegotiated after 4 MB of data have flowed, issue the following command to JMSAdmin:

```
ALTER CF(my.cf) SSLRESETCOUNT(4194304)
```

An application can set this property by calling the setSSLResetCount() method of a ConnectionFactory object.

If the value of this property is zero, which is the default value, the secret key is never renegotiated. The property is ignored if no CipherSuite is set.

If you are using an HP or Sun Java 2 Software Development Kit (SDK) or Java Runtime Environment (JRE), do not set this property to a value other than zero. If you do set the property to a value other than zero, a connection fails when it attempts to renegotiate the secret key.

For more information about the secret key that is used for encryption on an SSL enabled channel, see *WebSphere MQ Security*.

SSLSocketFactory object property

You might want to customize other aspects of the SSL connection for an application. For example, you might want to initialize cryptographic hardware or change the keystore and truststore in use. To do this, the application must first

create a javax.net.ssl.SSLSocketFactory object that is customized accordingly. Refer to your JSSE documentation for information on how to do this, as the customizable features vary from provider to provider. After a suitable SSLSocketFactory object is obtained, use the MQConnectionFactory.setSSLSocketFactory() method to configure JMS to use the customized SSLSocketFactory object.

If your application uses the setSSLSocketFactory() method to set a customized SSLSocketFactory object, the MQConnectionFactory object can no longer be bound into a JNDI namespace. Attempting to do so causes an exception. If this property is not set, the default SSLSocketFactory object is used. Refer to your JSSE documentation for details on the behavior of the default SSLSocketFactory object. This property is ignored if no CipherSuite is set.

Important: Do not assume that the use of the SSL properties ensures security when a ConnectionFactory object is retrieved from a JNDI namespace that is not itself secure. Specifically, the standard LDAP implementation of JNDI is not secure. An attacker can imitate the LDAP server, misleading a JMS application into connecting to the wrong server without noticing. With suitable security arrangements in place, other implementations of JNDI (such as the fscontext implementation) are secure.

Making changes to the JSSE keystore or truststore

If you change the contents of the JSSE keystore or truststore, or change the location of the keystore or truststore file, WebSphere MQ JMS applications that are running at the time do not automatically pick up the changes. For the changes to take effect, the following actions must be performed:

- The applications must close all their connections, and destroy any unused connections in connection pools.
- If your JSSE provider caches information from the keystore and truststore, this information must be refreshed.

After these actions have been performed, the applications can then recreate their connections.

Depending on how you design your applications, and on the function provided by your JSSE provider, it might be possible to perform these actions without stopping and restarting your applications. However, stopping and restarting the applications might be the simplest solution.

Chapter 13. JMS messages

JMS messages are composed of the following parts:

Header
All messages support the same set of header fields. Header fields contain values that are used by both clients and providers to identify and route messages.

Properties
Each message contains a built-in facility to support application-defined property values. Properties provide an efficient mechanism to filter application-defined messages.

Body JMS defines several types of message body that cover the majority of messaging styles currently in use.

JMS defines five types of message body:

Stream
A stream of Java primitive values. It is filled and read sequentially.

Map A set of name-value pairs, where names are strings and values are Java primitive types. The entries can be accessed sequentially or randomly by name. The order of the entries is undefined.

Text A message containing a java.lang.String.

Object
a message that contains a serializable Java object

Bytes A stream of uninterpreted bytes. This message type is for literally encoding a body to match an existing message format.

The JMSCorrelationID header field is used to link one message with another. It typically links a reply message with its requesting message. JMSCorrelationID can hold a provider-specific message ID, an application-specific String, or a provider-native byte[] value.

Message selectors

A message contains a built-in facility to support application-defined property values. In effect, this provides a mechanism to add application-specific header fields to a message. Properties allow an application, using message selectors, to have a JMS provider select or filter messages on its behalf, using application-specific criteria. Application-defined properties must obey the following rules:

- Property names must obey the rules for a message selector identifier.
- Property values can be boolean, byte, short, int, long, float, double, and String.
- The JMSX and JMS_ name prefixes are reserved.

Property values are set before sending a message. When a client receives a message, the message properties are read-only. If a client attempts to set properties at this point, a MessageNotWriteableException is thrown. If clearProperties is called, the properties can now be both read from, and written to.

A property value might duplicate a value in a message's body. JMS does not define a policy for what should or should not be made into a property. However, application developers must be aware that JMS providers probably handle data in a message's body more efficiently than data in a message's properties. For best performance, applications must use message properties only when they need to customize a message's header. The primary reason for doing this is to support customized message selection.

A JMS message selector allows a client to specify the messages that it is interested in by using the message header. Only messages whose headers match the selector are delivered.

Message selectors cannot refer to message body values.

A message selector matches a message when the selector evaluates to true when the message's header field and property values are substituted for their corresponding identifiers in the selector.

A message selector is a String, whose syntax is based on a subset of the SQL92 conditional expression syntax. The order in which a message selector is evaluated is from left to right within a precedence level. You can use parentheses to change this order. Predefined selector literals and operator names are written here in upper case; however, they are not case-sensitive.

A selector can contain:

- Literals
  - A string literal is enclosed in single quotes. A doubled single quote represents a single quote. Examples are 'literal' and 'literal''s'. Like Java string literals, these use the Unicode character encoding.
  - An exact numeric literal is a numeric value without a decimal point, such as 57, -957, and +62. Numbers in the range of Java long are supported.
  - An approximate numeric literal is a numeric value in scientific notation, such as 7E3 or -57.9E2, or a numeric value with a decimal, such as 7., -95.7, or +6.2. Numbers in the range of Java double are supported.
  - The boolean literals TRUE and FALSE.
- Identifiers:
  - An identifier is an unlimited length sequence of Java letters and Java digits, the first of which must be a Java letter. A letter is any character for which the method Character.isJavaLetter returns true. This includes _ and $. A letter or digit is any character for which the method Character.isJavaLetterOrDigit returns true.
  - Identifiers cannot be the names NULL, TRUE, or FALSE.
  - Identifiers cannot be NOT, AND, OR, BETWEEN, LIKE, IN, or IS.
  - Identifiers are either header field references or property references.
  - Identifiers are case-sensitive.
  - Message header field references are restricted to:
    - JMSDeliveryMode
    - JMSPriority
    - JMSMessageID
    - JMSTimestamp
    - JMSCorrelationID

- JMSType

  JMSMessageID, JMSTimestamp, JMSCorrelationID, and JMSType values can be null, and if so, are treated as a NULL value.

  – Any name beginning with JMSX is a JMS-defined property name.
  – Any name beginning with JMS_ is a provider-specific property name.
  – Any name that does not begin with JMS is an application-specific property name. If there is a reference to a property that does not exist in a message, its value is NULL. If it does exist, its value is the corresponding property value.

- White space is the same as it is defined for Java: space, horizontal tab, form feed, and line terminator.
- Expressions:
  – A selector is a conditional expression. A selector that evaluates to true matches; a selector that evaluates to false or unknown does not match.
  – Arithmetic expressions are composed of themselves, arithmetic operations, identifiers (whose value is treated as a numeric literal), and numeric literals.
  – Conditional expressions are composed of themselves, comparison operations, and logical operations.
- Standard bracketing (), to set the order in which expressions are evaluated, is supported.
- Logical operators in precedence order: NOT, AND, OR.
- Comparison operators: =, >, >=, <, <=, <> (not equal).
  – Only values of the same type can be compared. One exception is that it is valid to compare exact numeric values and approximate numeric values. (The type conversion required is defined by the rules of Java numeric promotion.) If there is an attempt to compare different types, the selector is always false.
  – String and boolean comparison is restricted to = and <>. Two strings are equal only if they contain the same sequence of characters.
- Arithmetic operators in precedence order:
  – +, - unary.
  – *, /, multiplication, and division.
  – +, -, addition, and subtraction.
  – Arithmetic operations on a NULL value are not supported. If they are attempted, the complete selector is always false.
  – Arithmetic operations must use Java numeric promotion.
- arithmetic-expr1 [NOT] BETWEEN arithmetic-expr2 and arithmetic-expr3 comparison operator:
  – Age BETWEEN 15 and 19 is equivalent to age >= 15 AND age <= 19.
  – Age NOT BETWEEN 15 and 19 is equivalent to age < 15 OR age > 19.
  – If any of the expressions of a BETWEEN operation are NULL, the value of the operation is false. If any of the expressions of a NOT BETWEEN operation are NULL, the value of the operation is true.
- identifier [NOT] IN (string-literal1, string-literal2,...) comparison operator where identifier has a String or NULL value.
  – Country IN ('UK', 'US', 'France') is true for 'UK' and false for 'Peru'. It is equivalent to the expression (Country = 'UK') OR (Country = 'US') OR (Country = 'France').
  – Country NOT IN ('UK', 'US', 'France') is false for 'UK' and true for 'Peru'. It is equivalent to the expression NOT ((Country = 'UK') OR (Country = 'US') OR (Country = 'France')).

  - If the identifier of an IN or NOT IN operation is NULL, the value of the operation is unknown.
- identifier [NOT] LIKE pattern-value [ESCAPE escape-character] comparison operator, where identifier has a string value. pattern-value is a string literal, where _ stands for any single character and % stands for any sequence of characters (including the empty sequence). All other characters stand for themselves. The optional escape-character is a single character string literal, whose character is used to escape the special meaning of the _ and % in pattern-value.
  - phone LIKE '12%3' is true for 123 and 12993 and false for 1234.
  - word LIKE 'l_se' is true for lose and false for loose.
  - underscored LIKE '_%' ESCAPE '\' is true for _foo and false for bar.
  - phone NOT LIKE '12%3' is false for 123 and 12993 and true for 1234.
  - If the identifier of a LIKE or NOT LIKE operation is NULL, the value of the operation is unknown.
- identifier IS NULL comparison operator tests for a null header field value, or a missing property value.
  - prop_name IS NULL.
- identifier IS NOT NULL comparison operator tests for the existence of a non-null header field value or a property value.
  - prop_name IS NOT NULL.

The following message selector selects messages with a message type of car, color of blue, and weight greater than 2500 lbs:

```
"JMSType = 'car' AND color = 'blue' AND weight > 2500"
```

As noted above, property values can be NULL. The evaluation of selector expressions that contain NULL values is defined by SQL 92 NULL semantics. The following is a brief description of these semantics:

- SQL treats a NULL value as unknown.
- Comparison or arithmetic with an unknown value always yields an unknown value.
- The IS NULL and IS NOT NULL operators convert an unknown value into the respective TRUE and FALSE values.

Although SQL supports fixed decimal comparison and arithmetic, JMS message selectors do not. This is why exact numeric literals are restricted to those without a decimal. It is also why there are numerics with a decimal as an alternate representation for an approximate numeric value.

SQL comments are not supported.

Mapping JMS messages onto WebSphere MQ messages

This section describes how the JMS message structure that is described in the first part of this chapter is mapped onto a WebSphere MQ message. It is of interest to programmers who want to transmit messages between JMS and traditional WebSphere MQ applications. It is also of interest to people who want to manipulate messages transmitted between two JMS applications, for example, in a message broker implementation.

This section does not apply if you use a direct connection to a broker. When you use a direct connection, all communication is performed directly over TCP/IP; no queues or messages are involved.

WebSphere MQ messages are composed of three components:

- The WebSphere MQ Message Descriptor (MQMD)
- A WebSphere MQ MQRFH2 header
- The message body.

The MQRFH2 is optional, and its inclusion in an outgoing message is governed by a flag in the JMS Destination class. You can set this flag using the WebSphere MQ JMS administration tool. Because the MQRFH2 carries JMS-specific information, always include it in the message when the sender knows that the receiving destination is a JMS application. Normally, omit the MQRFH2 when sending a message directly to a non-JMS application. This is because such an application does not expect an MQRFH2 in its WebSphere MQ message.

If an incoming message does not have an MQRFH2 header, the Queue or Topic object derived from the JMSReplyTo header field of the message, by default, has this flag set so that a reply message sent to the queue or topic also does not have an MQRFH2 header. You can switch off this behavior of including an MQRFH2 header in a reply message only if the original message has an MQRFH2 header by setting the TARGCLIENTMATCHING property of the connection factory to NO.

Figure 2 shows how the structure of a JMS message is transformed to a WebSphere MQ message and back again:

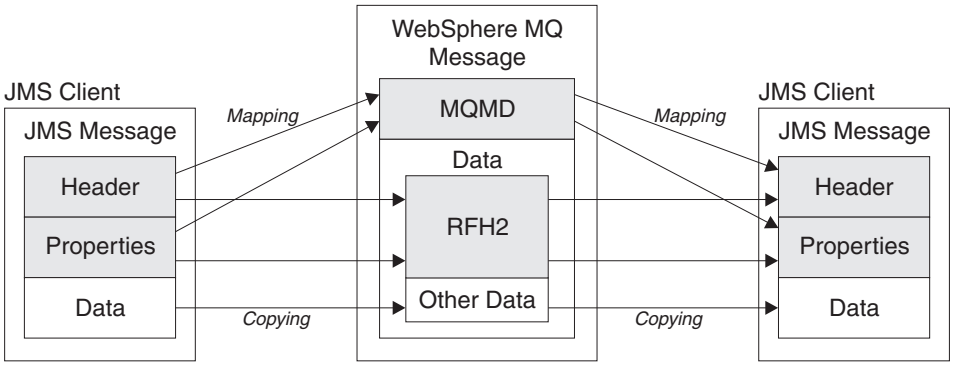

Figure 2. How messages are transformed between JMS and WebSphere MQ using the MQRFH2 header

The structures are transformed in two ways:

Mapping
Where the MQMD includes a field that is equivalent to the JMS field, the JMS field is mapped onto the MQMD field. Additional MQMD fields are exposed as JMS properties, because a JMS application might need to get or set these fields when communicating with a non-JMS application.

Copying
Where there is no MQMD equivalent, a JMS header field or property is passed, possibly transformed, as a field inside the MQRFH2.

The MQRFH2 header

This section describes the MQRFH Version 2 header, which carries JMS-specific data that is associated with the message content. The MQRFH2 Version 2 is an extensible header, and can also carry additional information that is not directly associated with JMS. However, this section covers only its use by JMS.

There are two parts of the header, a fixed portion and a variable portion.

Fixed portion

The fixed portion is modelled on the *standard* WebSphere MQ header pattern and consists of the following fields:

StrucId (MQCHAR4)

Structure identifier.

Must be MQRFH_STRUC_ID (value: “RFH ”) (initial value).

MQRFH_STRUC_ID_ARRAY (value: “R”,“F”,“H”,“ ”) is also defined in the usual way.

Version (MQLONG)

Structure version number.

Must be MQRFH_VERSION_2 (value: 2) (initial value).

StrucLength (MQLONG)

Total length of MQRFH2, including the NameValueData fields.

The value set into StrucLength must be a multiple of 4 (the data in the NameValueData fields can be padded with space characters to achieve this).

Encoding (MQLONG)

Data encoding.

Encoding of any numeric data in the portion of the message following the MQRFH2 (the next header, or the message data following this header).

CodedCharSetId (MQLONG)

Coded character set identifier.

Representation of any character data in the portion of the message following the MQRFH2 (the next header, or the message data following this header).

Format (MQCHAR8)

Format name.

Format name for the portion of the message following the MQRFH2.

Flags (MQLONG)

Flags.

MQRFH_NO_FLAGS =0. No flags set.

NameValueCCSID (MQLONG)

The coded character set identifier (CCSID) for the NameValueData character strings contained in this header. The NameValueData can be coded in a character set that differs from the other character strings that are contained in the header (StrucID and Format).

If the NameValueCCSID is a 2-byte Unicode CCSID (1200, 13488, or 17584), the byte order of the Unicode is the same as the byte ordering of the numeric fields in the MQRFH2. (For example, Version, StrucLength, and NameValueCCSID itself.)

The NameValueCCSID takes values from the following table:

Table 17. Possible values for NameValueCCSID field

Value	Meaning
1200	UCS2 open-ended
1208	UTF8
13488	UCS2 2.0 subset
17584	UCS2 2.1 subset (includes Euro symbol)

Variable portion

The variable portion follows the fixed portion. The variable portion contains a variable number of MQRFH2 folders. Each folder contains a variable number of elements or properties. Folders group together related properties. The MQRFH2 headers created by JMS can contain up to three folders:

The <mcd> folder

This contains properties that describe the *shape* or *format* of the message. For example, the `Msd` property identifies the message as being Text, Bytes, Stream, Map, Object, or null. This folder is always present in a JMS MQRFH2.

The <jms> folder

This is used to transport JMS header fields, and JMSX properties that cannot be fully expressed in the MQMD. This folder is always present in a JMS MQRFH2.

The <usr> folder

This is used to transport any application-defined properties associated with the message. This folder is present only if the application has set some application-defined properties.

The <mqext> folder

This is used to transport IBM defined properties that are used only by WebSphere Application Server. This folder is present only if the application has set at least one of these properties.

Table 18 shows a full list of property names.

Table 18. MQRFH2 folders and properties used by JMS

JMS field name	Java type	MQRFH2 folder name	Property name	Type/values
JMSDestination	Destination	jms	Dst	string
JMSExpiration	long	jms	Exp	i8
JMSPriority	int	jms	Pri	i4
JMSDeliveryMode	int	jms	Dlv	i4
JMSCorrelationID	String	jms	Cid	string
JMSReplyTo	Destination	jms	Rto	string
JMSTimestamp	long	jms	Tms	i8
JMSType	String	mcd	Type, Set, Fmt	string

Table 18. MQRFH2 folders and properties used by JMS (continued)

JMS field name	Java type	MQRFH2 folder name	Property name	Type/values
JMSXGroupID	String	jms	Gid	string
JMSXGroupSeq	int	jms	Seq	i4
xxx (user defined)	Any	usr	xxx	any
		mcd	Msd	jms_none jms_text jms_bytes jms_map jms_stream jms_object

The syntax used to express the properties in the variable portion is as follows:

NameValueLength (MQLONG)
Length in bytes of the NameValueData string that immediately follows this length field (it does not include its own length). The value set into NameValueLength is always a multiple of 4 (the NameValueData field is padded with space characters to achieve this).

NameValueData (MQCHARn)
A single character string, whose length in bytes is given by the preceding NameValueLength field. It contains a folder holding a sequence of properties. Each property is a name/type/value triplet, contained within an XML element whose name is the folder name, as follows:

```
<foldername> triplet1 triplet2 .....   tripletn </foldername>
```

The closing `</foldername>` tag can be followed by spaces as padding characters. Each triplet is encoded using an XML-like syntax:

```
<name dt='datatype'>value</name>
```

The `dt='datatype'` element is optional and is omitted for many properties, because the data type is predefined. If it is included, one or more space characters must be included before the `dt=` tag.

name is the name of the property; see Table 18 on page 385.

datatype
must match, after folding, one of the data types listed in Table 19 on page 387.

value is a string representation of the value to be conveyed, using the definitions in Table 19 on page 387.

A null value is encoded using the following syntax:

```
<name dt='datatype' xsi:nil='true'></name>
```

Do not use `xsi:nil='false'`.

Table 19. Property data types

Data type	Definition
string	Any sequence of characters excluding < and &
boolean	The character 0 or 1 (1 = ″true″)
bin.hex	Hexadecimal digits representing octets
i1	A number, expressed using digits 0..9, with optional sign (no fractions or exponent). Must lie in the range -128 to 127 inclusive
i2	A number, expressed using digits 0..9, with optional sign (no fractions or exponent). Must lie in the range -32768 to 32767 inclusive
i4	A number, expressed using digits 0..9, with optional sign (no fractions or exponent). Must lie in the range -2147483648 to 2147483647 inclusive
i8	A number, expressed using digits 0..9, with optional sign (no fractions or exponent). Must lie in the range -9223372036854775808 to 9223372036854775807 inclusive
int	A number, expressed using digits 0..9, with optional sign (no fractions or exponent). Must lie in the same range as i8. This can be used in place of one of the i* types if the sender does not want to associate a particular precision with the property
r4	Floating point number, magnitude <= 3.40282347E+38, >= 1.175E-37 expressed using digits 0..9, optional sign, optional fractional digits, optional exponent
r8	Floating point number, magnitude <= 1.7976931348623E+308, >= 2.225E-307 expressed using digits 0..9, optional sign, optional fractional digits, optional exponent

A string value can contain spaces. You must use the following escape sequences in a string value:

- `&` for the & character
- `<` for the < character

You can use the following escape sequences, but they are not required:

- `>` for the > character
- `&apos;` for the ' character
- `&quot;` for the " character

JMS fields and properties with corresponding MQMD fields

Table 20 lists the JMS header fields and Table 21 on page 388 lists the JMS properties that are mapped directly to MQMD fields. Table 22 on page 388 lists the provider specific properties and the MQMD fields that they are mapped to.

Table 20. JMS header fields mapping to MQMD fields

JMS header field	Java type	MQMD field	C type
JMSDeliveryMode	int	Persistence	MQLONG
JMSExpiration	long	Expiry	MQLONG
JMSPriority	int	Priority	MQLONG
JMSMessageID	String	MessageID	MQBYTE24

Table 20. JMS header fields mapping to MQMD fields (continued)

JMS header field	Java type	MQMD field	C type
JMSTimestamp	long	 PutDate PutTime	 MQCHAR8 MQCHAR8
JMSCorrelationID	String	CorrelId	MQBYTE24

Table 21. JMS properties mapping to MQMD fields

JMS property	Java type	MQMD field	C type
JMSXUserID	String	UserIdentifier	MQCHAR12
JMSXAppID	String	PutApplName	MQCHAR28
JMSXDeliveryCount	int	BackoutCount	MQLONG
JMSXGroupID	String	GroupId	MQBYTE24
JMSXGroupSeq	int	MsgSeqNumber	MQLONG

Table 22. JMS provider specific properties mapping to MQMD fields

JMS provider specific property	Java type	MQMD field	C type
JMS_IBM_Report_Exception	int	Report	MQLONG
JMS_IBM_Report_Expiration	int	Report	MQLONG
JMS_IBM_Report_COA	int	Report	MQLONG
JMS_IBM_Report_COD	int	Report	MQLONG
JMS_IBM_Report_PAN	int	Report	MQLONG
JMS_IBM_Report_NAN	int	Report	MQLONG
JMS_IBM_Report_Pass_Msg_ID	int	Report	MQLONG
JMS_IBM_Report_Pass_Correl_ID	int	Report	MQLONG
JMS_IBM_Report_Discard_Msg	int	Report	MQLONG
JMS_IBM_MsgType	int	MsgType	MQLONG
JMS_IBM_Feedback	int	Feedback	MQLONG
JMS_IBM_Format	String	Format[1]	MQCHAR8
JMS_IBM_PutApplType	int	PutApplType	MQLONG
JMS_IBM_Encoding	int	Encoding	MQLONG
JMS_IBM_Character_Set	String	CodedCharacterSetId	MQLONG
JMS_IBM_PutDate	String	PutDate	MQCHAR8
JMS_IBM_PutTime	String	PutTime	MQCHAR8
JMS_IBM_Last_Msg_In_Group	boolean	MsgFlags	MQLONG

Note:

1. JMS_IBM_Format represents the format of the message body. This can be defined by the application setting the JMS_IBM_Format property of the message (note that there is an 8 character limit), or can default to the WebSphere MQ format of the message body appropriate to the JMS message type. JMS_IBM_Format maps to the MQMD Format field only if the message contains no RFH or RFH2 sections. In a typical message, it will map to the Format field of the RFH2 immediately preceding the message body.

Mapping JMS fields onto WebSphere MQ fields (outgoing messages)

Table 23 shows how the JMS header fields are mapped into MQMD/RFH2 fields at send() or publish() time. Table 24 shows how JMS properties and Table 25 shows how JMS provider specific properties are mapped to MQMD fields at send() or publish() time,

For fields marked Set by Message Object, the value transmitted is the value held in the JMS message immediately before the send() or publish() operation. The value in the JMS message is left unchanged by the operation.

For fields marked Set by Send Method, a value is assigned when the send() or publish() is performed (any value held in the JMS message is ignored). The value in the JMS message is updated to show the value used.

Fields marked as Receive-only are not transmitted and are left unchanged in the message by send() or publish().

Table 23. Outgoing message field mapping

JMS header field name	MQMD field used for transmission	Header	Set by
JMSDestination		MQRFH2	Send Method
JMSDeliveryMode	Persistence	MQRFH2	Send Method
JMSExpiration	Expiry	MQRFH2	Send Method
JMSPriority	Priority	MQRFH2	Send Method
JMSMessageID	MessageID		Send Method
JMSTimestamp	PutDate/PutTime		Send Method
JMSCorrelationID	CorrelId	MQRFH2	Message Object
JMSReplyTo	ReplyToQ/ReplyToQMgr	MQRFH2	Message Object
JMSType		MQRFH2	Message Object
JMSRedelivered			Receive-only

Table 24. Outgoing message JMS property mapping

JMS property name	MQMD field used for transmission	Header	Set by
JMSXUserID	UserIdentifier		Send Method
JMSXAppID	PutApplName		Send Method
JMSXDeliveryCount			Receive-only
JMSXGroupID	GroupId	MQRFH2	Message Object
JMSXGroupSeq	MsgSeqNumber	MQRFH2	Message Object

Table 25. Outgoing message JMS provider specific property mapping

JMS provider specific property name	MQMD field used for transmission	Header	Set by
JMS_IBM_Report_Exception	Report		Message Object
JMS_IBM_Report_Expiration	Report		Message Object
JMS_IBM_Report_COA/COD	Report		Message Object

Table 25. Outgoing message JMS provider specific property mapping (continued)

JMS provider specific property name	MQMD field used for transmission	Header	Set by
JMS_IBM_Report_NAN/PAN	Report		Message Object
JMS_IBM_Report_Pass_Msg_ID	Report		Message Object
JMS_IBM_Report_Pass_Correl_ID	Report		Message Object
JMS_IBM_Report_Discard_Msg	Report		Message Object
JMS_IBM_MsgType	MsgType		Message Object
JMS_IBM_Feedback	Feedback		Message Object
JMS_IBM_Format	Format		Message Object
JMS_IBM_PutApplType	PutApplType		Send Method
JMS_IBM_Encoding	Encoding		Message Object
JMS_IBM_Character_Set	CodedCharacterSetId		Message Object
JMS_IBM_PutDate	PutDate		Send Method
JMS_IBM_PutTime	PutTime		Send Method
JMS_IBM_Last_Msg_In_Group	MsgFlags		Message Object

Mapping JMS header fields at send() or publish()

The following notes relate to the mapping of JMS fields at send() or publish():

JMSDestination to MQRFH2

This is stored as a string that serializes the salient characteristics of the destination object, so that a receiving JMS can reconstitute an equivalent destination object. The MQRFH2 field is encoded as URI (see uniform resource identifiers for details of the URI notation).

JMSReplyTo to MQMD ReplyToQ, ReplyToQMgr, MQRFH2

The queue and queue manager name are copied to the MQMD ReplyToQ and ReplyToQMgr fields respectively. The destination extension information (other useful details that are kept in the destination object) is copied into the MQRFH2 field. The MQRFH2 field is encoded as a URI (see uniform resource identifiers for details of the URI notation).

JMSDeliveryMode to MQMD Persistence

The JMSDeliveryMode value is set by the send() or publish() Method or MessageProducer, unless the Destination Object overrides it. The JMSDeliveryMode value is mapped to the MQMD Persistence field as follows:

- JMS value PERSISTENT is equivalent to MQPER_PERSISTENT
- JMS value NON_PERSISTENT is equivalent to MQPER_NOT_PERSISTENT

If the MQQueue persistence property is not set to JMSC.MQJMS_PER_QDEF, the delivery mode value is also encoded in the MQRFH2.

JMSExpiration to/from MQMD Expiry, MQRFH2

JMSExpiration stores the time to expire (the sum of the current time and the time to live), whereas MQMD stores the time to live. Also, JMSExpiration is in milliseconds, but MQMD.expiry is in centiseconds.

- If the send() method sets an unlimited time to live, MQMD Expiry is set to MQEI_UNLIMITED, and no JMSExpiration is encoded in the MQRFH2.
- If the send() method sets a time to live that is less than 214748364.7 seconds (about 7 years), the time to live is stored in MQMD. Expiry, and the expiration time (in milliseconds), are encoded as an i8 value in the MQRFH2.
- If the send() method sets a time to live greater than 214748364.7 seconds, MQMD.Expiry is set to MQEI_UNLIMITED. The true expiration time in milliseconds is encoded as an i8 value in the MQRFH2.

JMSPriority to MQMD Priority
Directly map JMSPriority value (0-9) onto MQMD priority value (0-9). If JMSPriority is set to a non-default value, the priority level is also encoded in the MQRFH2.

JMSMessageID from MQMD MessageID
All messages sent from JMS have unique message identifiers assigned by WebSphere MQ. The value assigned is returned in the MQMD messageId field after the MQPUT call, and is passed back to the application in the JMSMessageID field. The WebSphere MQ messageId is a 24-byte binary value, whereas the JMSMessageID is a string. The JMSMessageID is composed of the binary messageId value converted to a sequence of 48 hexadecimal characters, prefixed with the characters ID:. JMS provides a hint that can be set to disable the production of message identifiers. This hint is ignored, and a unique identifier is assigned in all cases. Any value that is set into the JMSMessageId field before a send() is overwritten.

JMSTimestamp to MQRFH2
During a send, the JMSTimestamp field is set according to the JVM's clock. This value is set into the MQRFH2. Any value that is set into the JMSTimestamp field before a send() is overwritten. See also the JMS_IBM_PutDate and JMS_IBM_PutTime properties.

JMSType to MQRFH2
This string is set into the MQRFH2 mcd.Type field. If it is in URI format, it can also affect mcd.Set and mcd.Fmt fields. See also Appendix C, "Connecting to other products," on page 639.

JMSCorrelationID to MQMD CorrelId, MQRFH2
The JMSCorrelationID can hold one of the following:

A provider specific message ID
This is a message identifier from a message previously sent or received, and so should be a string of 48 hexadecimal digits that are prefixed with ID:. The prefix is removed, the remaining characters are converted into binary, and then they are set into the MQMD CorrelId field. No CorrelId value is encoded in the MQRFH2.

A provider-native byte[] value
The value is copied into the MQMD CorrelId field - padded with nulls, or truncated to 24 bytes if necessary. No CorrelId value is encoded in the MQRFH2.

An application-specific string
The value is copied into the MQRFH2. The first 24 bytes of the string, in UTF8 format, are written into the MQMD CorrelID.

Mapping JMS property fields

These notes refer to the mapping of JMS property fields in WebSphere MQ messages:

JMSXUserID from MQMD UserIdentifier
JMSXUserID is set on return from send call.

JMSXAppID from MQMD PutApplName
JSMXAppID is set on return from send call.

JMSXGroupID to MQRFH2 (point-to-point)
For point-to-point messages, the JMSXGroupID is copied into the MQMD GroupID field. If the JMSXGroupID starts with the prefix ID:, it is converted into binary. Otherwise, it is encoded as a UTF8 string. The value is padded or truncated if necessary to a length of 24 bytes. The MQMF_MSG_IN_GROUP flag is set.

JMSXGroupID to MQRFH2 (publish/subscribe)
For publish/subscribe messages, the JMSXGroupID is copied into the MQRFH2 as a string.

JMSXGroupSeq MQMD MsgSeqNumber (point-to-point)
For point-to-point messages, the JMSXGroupSeq is copied into the MQMD MsgSeqNumber field. The MQMF_MSG_IN_GROUP flag is set.

JMSXGroupSeq MQMD MsgSeqNumber (publish/subscribe)
For publish/subscribe messages, the JMSXGroupSeq is copied into the MQRFH2 as an i4.

Mapping JMS provider-specific fields

The following notes refer to the mapping of JMS Provider specific fields into WebSphere MQ messages:

JMS_IBM_Report_<name> to MQMD Report
A JMS application can set the MQMD Report options, using the following JMS_IBM_Report_XXX properties. The single MQMD is mapped to several JMS_IBM_Report_XXX properties. The application must set the value of these properties to the standard WebSphere MQ MQRO_ constants (included in com.ibm.mq.MQC). So, for example, to request COD with full Data, the application must set JMS_IBM_Report_COD to the value MQC.MQRO_COD_WITH_FULL_DATA.

JMS_IBM_Report_Exception
MQRO_EXCEPTION or
MQRO_EXCEPTION_WITH_DATA or
MQRO_EXCEPTION_WITH_FULL_DATA

JMS_IBM_Report_Expiration
MQRO_EXPIRATION or
MQRO_EXPIRATION_WITH_DATA or
MQRO_EXPIRATION_WITH_FULL_DATA

JMS_IBM_Report_COA
MQRO_COA or
MQRO_COA_WITH_DATA or
MQRO_COA_WITH_FULL_DATA

JMS_IBM_Report_COD
MQRO_COD or
MQRO_COD_WITH_DATA or
MQRO_COD_WITH_FULL_DATA

JMS_IBM_Report_PAN
MQRO_PAN

JMS_IBM_Report_NAN
MQRO_NAN

JMS_IBM_Report_Pass_Msg_ID
MQRO_PASS_MSG_ID

JMS_IBM_Report_Pass_Correl_ID
MQRO_PASS_CORREL_ID

JMS_IBM_Report_Discard_Msg
MQRO_DISCARD_MSG

JMS_IBM_MsgType to MQMD MsgType
Value maps directly onto MQMD MsgType. If the application has not set an explicit value of JMS_IBM_MsgType, a default value is used. This default value is determined as follows:

- If JMSReplyTo is set to a WebSphere MQ queue destination, MSGType is set to the value MQMT_REQUEST
- If JMSReplyTo is not set, or is set to anything other than a WebSphere MQ queue destination, MsgType is set to the value MQMT_DATAGRAM

JMS_IBM_Feedback to MQMD Feedback
Value maps directly onto MQMD Feedback.

JMS_IBM_Format to MQMD Format
Value maps directly onto MQMD Format.

JMS_IBM_Encoding to MQMD Encoding
If set, this property overrides the numeric encoding of the Destination Queue or Topic.

JMS_IBM_Character_Set to MQMD CodedCharacterSetId
If set, this property overrides the coded character set property of the Destination Queue or Topic.

JMS_IBM_PutDate from MQMD PutDate
The value of this property is set, during send, directly from the PutDate field in the MQMD. Any value that is set into the JMS_IBM_PutDate property before a send is overwritten. This field is a String of eight characters, in the WebSphere MQ Date format of YYYYMMDD. This property can be used in conjunction with the JMS_IBM_PutTime property to determine the time the message was put according to the queue manager.

JMS_IBM_PutTime from MQMD PutTime
The value of this property is set, during send, directly from the PutTime field in the MQMD. Any value that is set into the JMS_IBM_PutTime property before a send is overwritten. This field is a String of eight characters, in the WebSphere MQ Time format of HHMMSSTH. This property can be used in conjunction with the JMS_IBM_PutDate property to determine the time the message was put according to the queue manager.

JMS_IBM_Last_Msg_In_Group to MQMD MsgFlags
For point-to-point messaging, this boolean value maps to the MQMF_LAST_MSG_IN_GROUP flag in the MQMD MsgFlags field. It is normally used in conjunction with the JMSXGroupID and JMSXGroupSeq

properties to indicate to a legacy WebSphere MQ application that this is the last message in a group. This property is ignored for publish/subscribe messaging.

Mapping WebSphere MQ fields onto JMS fields (incoming messages)

Table 26 shows how JMS header fields and Table 27 shows how JMS property fields are mapped into MQMD/MQRFH2 fields at send() or publish() time. Table 28 shows how JMS provider specific properties are mapped.

Table 26. Incoming message JMS header field mapping

JMS header field name	MQMD field retrieved from	MQRFH2 field retrieved from
JMSDestination		jms.Dst
JMSDeliveryMode	Persistence[1]	jms.Dlv[1]
JMSExpiration		jms.Exp
JMSPriority	Priority	
JMSMessageID	MessageID	
JMSTimestamp	PutDate[1] PutTime[1]	jms.Tms[1]
JMSCorrelationID	CorrelId[1]	jms.Cid[1]
JMSReplyTo	ReplyToQ[1] ReplyToQMgr[1]	jms.Rto[1]
JMSType		mcd.Type, mcd.Set, mcd.Fmt
JMSRedelivered	BackoutCount	

Note:

1. For properties that can have values retrieved from the MQRFH2 or the MQMD, if both are available, the setting in the MQRFH2 is used.

Table 27. Incoming message property mapping

JMS property name	MQMD field retrieved from	MQRFH2 field retrieved from
JMSXUserID	UserIdentifier	
JMSXAppID	PutApplName	
JMSXDeliveryCount	BackoutCount	
JMSXGroupID	GroupId[1]	jms.Gid[1]
JMSXGroupSeq	MsgSeqNumber[1]	jms.Seq[1]

Note:

1. For properties that can have values retrieved from the MQRFH2 or the MQMD, if both are available, the setting in the MQRFH2 is used.

Table 28. Incoming message provider specific JMS property mapping

JMS property name	MQMD field retrieved from	MQRFH2 field retrieved from
JMS_IBM_Report_Exception	Report	

Table 28. Incoming message provider specific JMS property mapping (continued)

JMS property name	MQMD field retrieved from	MQRFH2 field retrieved from
JMS_IBM_Report_Expiration	Report	
JMS_IBM_Report_COA	Report	
JMS_IBM_Report_COD	Report	
JMS_IBM_Report_PAN	Report	
JMS_IBM_Report_NAN	Report	
JMS_IBM_Report_ Pass_Msg_ID	Report	
JMS_IBM_Report_Pass_Correl_ID	Report	
JMS_IBM_Report_Discard_Msg	Report	
JMS_IBM_MsgType	MsgType	
JMS_IBM_Feedback	Feedback	
JMS_IBM_Format	Format	
JMS_IBM_PutApplType	PutApplType	
JMS_IBM_Encoding [1]	Encoding	
JMS_IBM_Character_Set [1]	CodedCharacterSetId	
JMS_IBM_PutDate	PutDate	
JMS_IBM_PutTime	PutTime	
JMS_IBM_Last_Msg_In_Group	MsgFlags	

1. Only set if the incoming message is a Bytes Message.

Mapping JMS to a native WebSphere MQ application

This section describes what happens if you send a message from a JMS client application to a traditional WebSphere MQ application with no knowledge of MQRFH2 headers. Figure 3 on page 396 shows the mapping.

The administrator indicates that the JMS client is communicating with such an application by setting the WebSphere MQ destination's TargetClient value to JMSC.MQJMS_CLIENT_NONJMS_MQ. This indicates that no MQRFH2 field is to be produced. Note that if this is not done, the receiving application must be able to handle the MQRFH2 field.

The mapping from JMS to MQMD targeted at a native WebSphere MQ application is the same as mapping from JMS to MQMD targeted at a true JMS client. If JMS receives a WebSphere MQ message with the MQMD format field set to other than MQFMT_RFH2, data is being received from a non-JMS application. If the format is MQFMT_STRING, the message is received as a JMS text message. Otherwise, it is received as a JMS bytes message. Because there is no MQRFH2, only those JMS properties that are transmitted in the MQMD can be restored.

If a WebSphere MQ JMS application receives a message that does not have an MQRFH2 header, the TARGCLIENT property of the Queue or Topic object derived from the JMSReplyTo header field of the message, by default, is set to MQ. This means that a reply message sent to the queue or topic also does not have an MQRFH2 header. You can switch off this behavior of including an MQRFH2

header in a reply message only if the original message has an MQRFH2 header by setting the TARGCLIENTMATCHING property of the connection factory to NO.

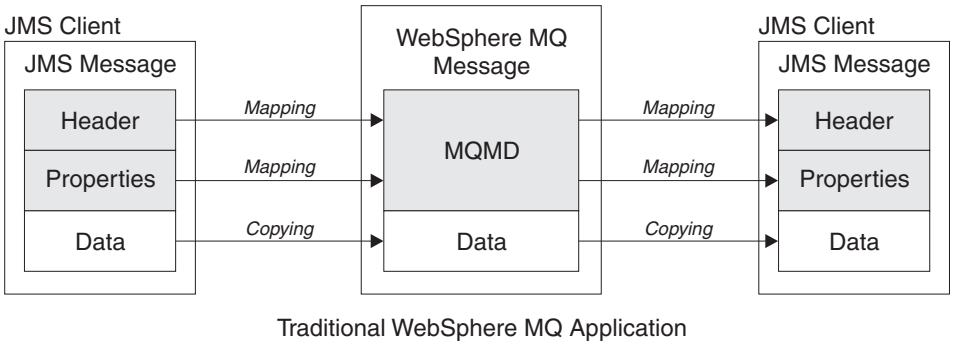

Figure 3. How JMS messages are transformed to WebSphere MQ messages (no MQRFH2 header)

Message body

This section discusses the encoding of the message body itself. The encoding depends on the type of JMS message:

ObjectMessage
is an object serialized by the Java Runtime in the normal way.

TextMessage
is an encoded string. For an outgoing message, the string is encoded in the character set given by the destination object. This defaults to UTF8 encoding (the UTF8 encoding starts with the first character of the message; there is no length field at the start). It is, however, possible to specify any other character set supported by WebSphere MQ Java. Such character sets are used mainly when you send a message to a non-JMS application.

If the character set is a double-byte set (including UTF16), the destination object's integer encoding specification determines the order of the bytes.

An incoming message is interpreted using the character set and encoding that are specified in the message itself. These specifications are in the last WebSphere MQ header (or MQMD if there are no headers). For JMS messages, the last header is usually the MQRFH2.

BytesMessage
is, by default, a sequence of bytes as defined by the JMS 1.0.2 specification and associated Java documentation.

For an outgoing message that was assembled by the application itself, the destination object's encoding property can be used to override the encodings of integer and floating point fields contained in the message. For example, you can request that floating point values are stored in S/390® rather than IEEE format).

An incoming message is interpreted using the numeric encoding specified in the message itself. This specification is in the rightmost WebSphere MQ header (or MQMD if there are no headers). For JMS messages, the rightmost header is usually the MQRFH2.

If a BytesMessage is received, and is re-sent without modification, its body is transmitted byte for byte, as it was received. The destination object's encoding property has no effect on the body. The only string-like entity that can be sent explicitly in a BytesMessage is a UTF8 string. This is

encoded in Java UTF8 format, and starts with a 2-byte length field. The destination object's character set property has no effect on the encoding of an outgoing BytesMessage. The character set value in an incoming WebSphere MQ message has no effect on the interpretation of that message as a JMS BytesMessage.

Non-Java applications are unlikely to recognize the Java UTF8 encoding. Therefore, for a JMS application to send a BytesMessage that contains text data, the application itself must convert its strings to byte arrays, and write these byte arrays into the BytesMessage.

MapMessage
is a string containing XML name/type/value triplets encoded as:

```
<map>
  <elt name="elementname1" dt="datatype1">value1</elt>
  <elt name="elementname2" dt="datatype2">value2</elt>
  ...
</map>
```

where `datatype` is one of the data types listed in Table 19 on page 387. The default data type is `string`, and so the attribute `dt="string"` is omitted for string elements.

Previous versions of the WebSphere MQ JMS client encoded the body of a map message in the following format:

```
<map>
  <elementname1 dt="datatype1">value1</elementname1>
  <elementname2 dt="datatype2">value2</elementname2>
  ...
</map>
```

The current version and Version 5.3 of the WebSphere MQ JMS client can interpret either format, but versions of the WebSphere MQ JMS client earlier than Version 5.3 cannot interpret the current format. If an application must be able to receive map messages encoded in the current format, the application must use the current version or Version 5.3 of the WebSphere MQ JMS client.

If an application needs to send map messages to another application that is using a WebSphere MQ JMS client earlier than Version 5.3, the sending application must call the connection factory method `setMapNameStyle(JMSC.MAP_NAME_STYLE_COMPATIBLE)` to specify that the map messages are sent in the previous format, which can be interpreted by any WebSphere MQ JMS client. By default, all map messages are sent in the current format.

The character set used to encode or interpret the XML string that forms the body of a map message is determined according to the rules that apply to a text message.

StreamMessage
is like a map message, but without element names:

```
<stream>
  <elt dt="datatype1">value1</elt>
  <elt dt="datatype2">value2</elt>
  ...
</stream>
```

where `datatype` is one of the data types listed in Table 19 on page 387. The default data type is `string`, and so the attribute `dt="string"` is omitted for string elements.

The character set used to encode or interpret the XML string that makes up the StreamMessage body is determined following the rules that apply to a TextMessage.

The MQRFH2.format field is set as follows:

MQFMT_NONE
for ObjectMessage, BytesMessage, or messages with no body.

MQFMT_STRING
for TextMessage, StreamMessage, or MapMessage.

Chapter 14. WebSphere MQ JMS Application Server Facilities

WebSphere MQ JMS supports the Application Server Facilities (ASF) that are specified in the *Java Message Service Specification, Version 1.1* (see Sun's Java Web site at `http://java.sun.com`). This specification identifies three roles within this programming model:

- **The JMS provider** supplies ConnectionConsumer and advanced Session functionality.
- **The application server** supplies ServerSessionPool and ServerSession functionality.
- **The client application** uses the functionality that the JMS provider and application server supply.

This chapter does not apply if you use a direct connection to a broker.

The following sections contain details about how WebSphere MQ JMS implements ASF:

- "ASF classes and functions" describes how WebSphere MQ JMS implements the ConnectionConsumer class and advanced functionality in the Session class.
- "Application server sample code" on page 406 describes the sample ServerSessionPool and ServerSession code that is supplied with WebSphere MQ JMS.
- "Examples of ASF use" on page 409 describes supplied ASF samples and examples of ASF use from the perspective of a client application.

ASF classes and functions

WebSphere MQ JMS implements the ConnectionConsumer class and advanced functionality in the Session class. For details, see:

- "MQQueueConnection" on page 526
- "MQSession" on page 533
- "MQTopicConnection" on page 546

ConnectionConsumer

The JMS specification enables an application server to integrate closely with a JMS implementation by using the `ConnectionConsumer` interface. This feature provides concurrent processing of messages. Typically, an application server creates a pool of threads, and the JMS implementation makes messages available to these threads. A JMS-aware application server (such as WebSphere Application Server) can use this feature to provide high-level messaging functionality, such as message driven beans.

Normal applications do not use the ConnectionConsumer, but expert JMS clients might use it. For such clients, the ConnectionConsumer provides a high-performance method to deliver messages concurrently to a pool of threads. When a message arrives on a queue or a topic, JMS selects a thread from the pool and delivers a batch of messages to it. To do this, JMS runs an associated MessageListener's `onMessage()` method.

You can achieve the same effect by constructing multiple Session and MessageConsumer objects, each with a registered MessageListener. However, the ConnectionConsumer provides better performance, less use of resources, and greater flexibility. In particular, fewer Session objects are required.

To help you develop applications that use ConnectionConsumers, WebSphere MQ JMS provides a fully-functioning example implementation of a pool. You can use this implementation without any changes, or adapt it to suit the specific needs of the application.

Planning an application

This section tells you how to plan an application including:

- "General principles for point-to-point messaging"
- "General principles for publish/subscribe messaging" on page 401
- "Handling poison messages" on page 402
- "Removing messages from the queue" on page 403

General principles for point-to-point messaging

When an application creates a ConnectionConsumer from a QueueConnection object, it specifies a JMS queue object and a selector string. The ConnectionConsumer then begins to provide messages to sessions in the associated ServerSessionPool. Messages arrive on the queue, and if they match the selector, they are delivered to sessions in the associated ServerSessionPool.

In WebSphere MQ terms, the queue object refers to either a QLOCAL or a QALIAS on the local queue manager. If it is a QALIAS, that QALIAS must refer to a QLOCAL. The fully-resolved WebSphere MQ QLOCAL is known as the *underlying QLOCAL*. A ConnectionConsumer is said to be *active* if it is not closed and its parent QueueConnection is started.

It is possible for multiple ConnectionConsumers, each with different selectors, to run against the same underlying QLOCAL. To maintain performance, unwanted messages must not accumulate on the queue. Unwanted messages are those for which no active ConnectionConsumer has a matching selector. You can set the QueueConnectionFactory so that these unwanted messages are removed from the queue (for details, see "Removing messages from the queue" on page 403). You can set this behavior in one of two ways:

- Use the JMS administration tool to set the QueueConnectionFactory to MRET(NO).
- In your program, use:

```
MQQueueConnectionFactory.setMessageRetention(JMSC.MQJMS_MRET_NO)
```

If you do not change this setting, the default is to retain such unwanted messages on the queue.

It is possible that ConnectionConsumers that target the same underlying QLOCAL could be created from multiple QueueConnection objects. However, for performance reasons, we recommend that multiple JVMs do not create ConnectionConsumers against the same underlying QLOCAL.

When you set up the WebSphere MQ queue manager, consider the following points:

- The underlying QLOCAL must be enabled for shared input. To do this, use the following MQSC command:

```
ALTER QLOCAL(your.qlocal.name) SHARE GET(ENABLED)
```

- Your queue manager must have an enabled dead-letter queue. If a ConnectionConsumer experiences a problem when it puts a message on the dead-letter queue, message delivery from the underlying QLOCAL stops. To define a dead-letter queue, use:

  ```
  ALTER QMGR DEADQ(your.dead.letter.queue.name)
  ```

- The user that runs the ConnectionConsumer must have authority to perform MQOPEN with MQOO_SAVE_ALL_CONTEXT and MQOO_PASS_ALL_CONTEXT. For details, see the WebSphere MQ documentation for your specific platform.
- If unwanted messages are left on the queue, they degrade the system performance. Therefore, plan your message selectors so that between them, the ConnectionConsumers will remove all messages from the queue.

For details about MQSC commands, see the *WebSphere MQ Script (MQSC) Command Reference*.

General principles for publish/subscribe messaging

When an application creates a ConnectionConsumer from a TopicConnection object, it specifies a Topic object and a selector string. The ConnectionConsumer then begins to receive messages that match the selector on that Topic.

Alternatively, an application can create a durable ConnectionConsumer that is associated with a specific name. This ConnectionConsumer receives messages that have been published on the Topic since the durable ConnectionConsumer was last active. It receives all such messages that match the selector on the Topic.

For non-durable subscriptions, a separate queue is used for ConnectionConsumer subscriptions. The CCSUB configurable option on the TopicConnectionFactory specifies the queue to use. Normally, the CCSUB specifies a single queue for use by all ConnectionConsumers that use the same TopicConnectionFactory. However, it is possible to make each ConnectionConsumer generate a temporary queue by specifying a queue name prefix followed by an asterisk (*). For information about how to form a valid prefix, see “Configuring nondurable message consumers” on page 362.

For durable subscriptions, the CCDSUB property of the Topic specifies the queue to use. Again, this can be a queue that already exists or a queue name prefix followed by an asterisk (*). If you specify a queue that already exists, all durable ConnectionConsumers that subscribe to the Topic use this queue. If you specify a queue name prefix followed by an asterisk (*), a queue is generated the first time that a durable ConnectionConsumer is created with a given name. This queue is reused later when a durable ConnectionConsumer is created with the same name. For information about how to form a valid prefix, see “Configuring durable topic subscribers” on page 362.

When you set up the WebSphere MQ queue manager, consider the following points:

- Your queue manager must have an enabled dead-letter queue. If a ConnectionConsumer experiences a problem when it puts a message on the dead-letter queue, message delivery from the underlying QLOCAL stops. To define a dead-letter queue, use:

  ```
  ALTER QMGR DEADQ(your.dead.letter.queue.name)
  ```

- The user that runs the ConnectionConsumer must have authority to perform MQOPEN with MQOO_SAVE_ALL_CONTEXT and MQOO_PASS_ALL_CONTEXT. For details, see the WebSphere MQ documentation for your platform.
- You can optimize performance for an individual ConnectionConsumer by creating a separate, dedicated, queue for it. This is at the cost of extra resource usage.

Handling poison messages

Sometimes, a badly-formatted message arrives on a queue. Such a message might make the receiving application fail and back out the receipt of the message. In this situation, such a message might be received, then returned to the queue, repeatedly. These messages are known as *poison messages*. The ConnectionConsumer must be able to detect poison messages and reroute them to an alternative destination.

When an application uses ConnectionConsumers, the circumstances in which a message is backed out depend on the session that the application server provides:

- When the session is non-transacted, with AUTO_ACKNOWLEDGE or DUPS_OK_ACKNOWLEDGE, a message is backed out only after a system error, or if the application terminates unexpectedly.
- When the session is non-transacted with CLIENT_ACKNOWLEDGE, unacknowledged messages can be backed out by the application server calling Session.recover().

  Typically, the client implementation of MessageListener or the application server calls Message.acknowledge(). Message.acknowledge() acknowledges all messages delivered on the session so far.
- When the session is transacted, unacknowledged messages can be backed out by the application server calling Session.rollback().
- If the application server supplies an XASession, messages are committed or backed out depending on a distributed transaction. The application server takes responsibility for completing the transaction.

The WebSphere MQ queue manager keeps a record of the number of times that each message has been backed out. When this number reaches a configurable threshold, the ConnectionConsumer requeues the message on a named requeue queue. If this requeue fails for any reason, the message is removed from the queue and either requeued to the dead-letter queue, or discarded. See “Removing messages from the queue” on page 403 for more details.

The threshold and the name of the requeue queue are attributes of the WebSphere MQ local queue. The names of the attributes are *BackoutThreshold* and *BackoutRequeueQName*. For point-to-point messaging, this is the underlying local queue. For publish/subscribe messaging, this is the CCSUB queue defined on the TopicConnectionFactory, or the CCDSUB queue defined on the Topic. To set the *BackoutThreshold* and *BackoutRequeueQName* attributes, issue the following MQSC command:

```
ALTER QLOCAL(your.queue.name) BOTHRESH(threshold) BOQUEUE(your.requeue.queue.name)
```

For publish/subscribe messaging, if your system creates a dynamic queue for each subscription, these settings are obtained from the WebSphere MQ JMS model queue. To alter these settings, you can use:

```
ALTER QMODEL(SYSTEM.JMS.MODEL.QUEUE) BOTHRESH(threshold) BOQUEUE(your.requeue.queue.name)
```

If the threshold is zero, poison message handling is disabled, and poison messages remain on the input queue. Otherwise, when the backout count reaches the threshold, the message is sent to the named requeue queue. If the backout count reaches the threshold, but the message cannot go to the requeue queue, the message is sent to the dead-letter queue or discarded. This situation occurs if the requeue queue is not defined, or if the ConnectionConsumer cannot send the message to the requeue queue. See "Removing messages from the queue" for further details.

The embedded JMS provider in WebSphere Application Server, Version 5.0 and Version 5.1 handles poison messages in a way that is different to that just described for WebSphere MQ JMS. For information about how the embedded JMS provider handles poison messages, see the relevant WebSphere Application Server information center.

Removing messages from the queue

When an application uses ConnectionConsumers, JMS might need to remove messages from the queue in a number of situations:

Badly formatted message
A message might arrive that JMS cannot parse.

Poison message
A message might reach the backout threshold, but the ConnectionConsumer fails to requeue it on the backout queue.

No interested ConnectionConsumer
For point-to-point messaging, when the QueueConnectionFactory is set so that it does not retain unwanted messages, a message arrives that is unwanted by any of the ConnectionConsumers.

In these situations, the ConnectionConsumer attempts to remove the message from the queue. The disposition options in the report field of the message's MQMD set the exact behavior. These options are:

MQRO_DEAD_LETTER_Q
The message is requeued to the queue manager's dead-letter queue. This is the default.

MQRO_DISCARD_MSG
The message is discarded.

The ConnectionConsumer also generates a report message, and this also depends on the report field of the message's MQMD. This message is sent to the message's ReplyToQ on the ReplyToQmgr. If there is an error while the report message is being sent, the message is sent to the dead-letter queue instead. The exception report options in the report field of the message's MQMD set details of the report message. These options are:

MQRO_EXCEPTION
A report message is generated that contains the MQMD of the original message. It does not contain any message body data.

MQRO_EXCEPTION_WITH_DATA
A report message is generated that contains the MQMD, any MQ headers, and 100 bytes of body data.

MQRO_EXCEPTION_WITH_FULL_DATA
A report message is generated that contains all data from the original message.

default
No report message is generated.

When report messages are generated, the following options are honored:

- MQRO_NEW_MSG_ID
- MQRO_PASS_MSG_ID
- MQRO_COPY_MSG_ID_TO_CORREL_ID
- MQRO_PASS_CORREL_ID

If a ConnectionConsumer cannot follow the disposition options or exception report options in the message's MQMD, its action depends on the persistence of the message. If the message is non-persistent, the message is discarded and no report message is generated. If the message is persistent, delivery of all messages from the QLOCAL stops.

It is important to define a dead-letter queue, and to check it regularly to ensure that no problems occur. Particularly, ensure that the dead-letter queue does not reach its maximum depth, and that its maximum message size is large enough for all messages.

When a message is requeued to the dead-letter queue, it is preceded by a WebSphere MQ dead-letter header (MQDLH). See the *WebSphere MQ Application Programming Reference* for details about the format of the MQDLH. You can identify messages that a ConnectionConsumer has placed on the dead-letter queue, or report messages that a ConnectionConsumer has generated, by the following fields:

- PutApplType is `MQAT_JAVA (0x1C)`
- PutApplName is "`MQ JMS ConnectionConsumer`"

These fields are in the MQDLH of messages on the dead-letter queue, and the MQMD of report messages. The feedback field of the MQMD, and the Reason field of the MQDLH, contain a code describing the error. For details about these codes, see "Error handling." Other fields are as described in the *WebSphere MQ Application Programming Reference*.

Error handling

This section covers various aspects of error handling, including "Recovering from error conditions" and "Reason and feedback codes" on page 405.

Recovering from error conditions

If a ConnectionConsumer experiences a serious error, message delivery to all ConnectionConsumers with an interest in the same QLOCAL stops. Typically, this occurs if the ConnectionConsumer cannot requeue a message to the dead-letter queue, or it experiences an error when reading messages from the QLOCAL.

When this occurs, any ExceptionListener that is registered with the affected Connection is notified.

You can use these to identify the cause of the problem. In some cases, the system administrator must intervene to resolve the problem.

There are two ways in which an application can recover from these error conditions:

- Call `close()` on all affected ConnectionConsumers. The application can create new ConnectionConsumers only after all affected ConnectionConsumers are closed and any system problems are resolved.
- Call `stop()` on all affected Connections. Once all Connections are stopped and any system problems are resolved, the application should be able to `start()` all Connections successfully.

Reason and feedback codes

To determine the cause of an error, you can use:

- The feedback code in any report messages
- The reason code in the MQDLH of any messages in the dead-letter queue

ConnectionConsumers generate the following reason codes.

MQRC_BACKOUT_THRESHOLD_REACHED (0x93A; 2362)

Cause The message has reached the Backout Threshold defined on the QLOCAL, but no Backout Queue is defined.

On platforms where you cannot define the Backout Queue, the message has reached the JMS-defined backout threshold of 20.

Action

If this is not wanted, define the Backout Queue for the relevant QLOCAL. Also look for the cause of the multiple backouts.

MQRC_MSG_NOT_MATCHED (0x93B; 2363)

Cause In point-to-point messaging, there is a message that does not match any of the selectors for the ConnectionConsumers monitoring the queue. To maintain performance, the message is requeued to the dead-letter queue.

Action

To avoid this situation, ensure that ConnectionConsumers using the queue provide a set of selectors that deal with all messages, or set the QueueConnectionFactory to retain messages.

Alternatively, investigate the source of the message.

MQRC_JMS_FORMAT_ERROR (0x93C; 2364)

Cause JMS cannot interpret the message on the queue.

Action

Investigate the origin of the message. JMS usually delivers messages of an unexpected format as a BytesMessage or TextMessage. Occasionally, this fails if the message is very badly formatted.

Other codes that appear in these fields are caused by a failed attempt to requeue the message to a Backout Queue. In this situation, the code describes the reason that the requeue failed. To diagnose the cause of these errors, refer to the *WebSphere MQ Application Programming Reference*.

If the report message cannot be put on the ReplyToQ, it is put on the dead-letter queue. In this situation, the feedback field of the MQMD is filled in as described above. The reason field in the MQDLH explains why the report message could not be placed on the ReplyToQ.

Application server sample code

Figure 4 summarizes the principles of ServerSessionPool and ServerSession functionality.

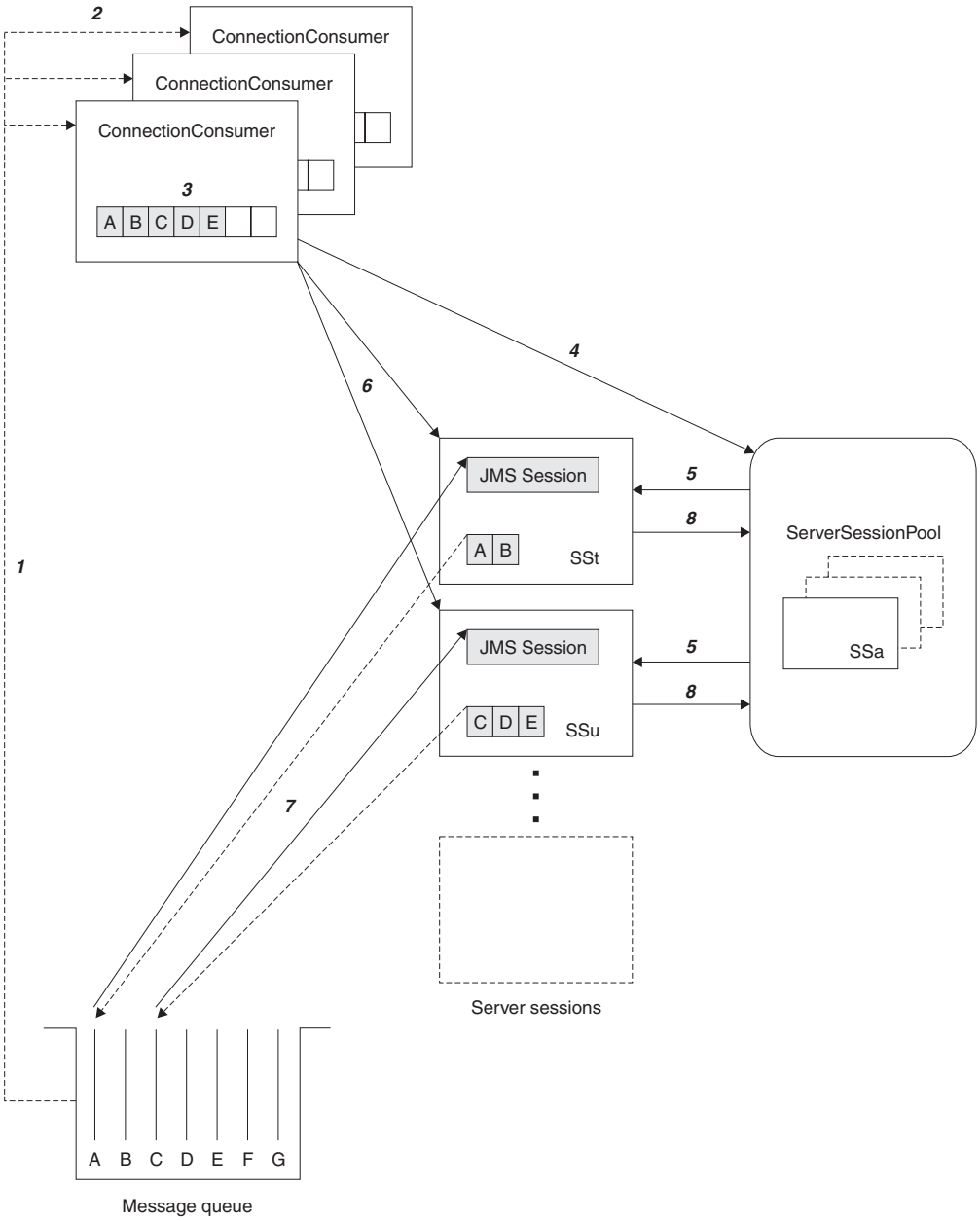

Figure 4. ServerSessionPool and ServerSession functionality

1. The ConnectionConsumers get message references from the queue.
2. Each ConnectionConsumer selects specific message references.
3. The ConnectionConsumer buffer holds the selected message references.
4. The ConnectionConsumer requests one or more ServerSessions from the ServerSessionPool.
5. ServerSessions are allocated from the ServerSessionPool.
6. The ConnectionConsumer assigns message references to the ServerSessions and starts the ServerSession threads running.

7. Each ServerSession retrieves its referenced messages from the queue. It passes them to the `onMessage` method from the MessageListener that is associated with the JMS Session.
8. After it completes its processing, the ServerSession is returned to the pool.

Normally, the application server supplies ServerSessionPool and ServerSession functionality. However, WebSphere MQ JMS is supplied with a simple implementation of these interfaces, with program source. These samples are stored in a subdirectory of the WebSphere MQ Java samples directory. The subdirectory is called jms\asf on Windows and jms/asf on the other platforms. To determine the WebSphere MQ Java samples directory for your platform, see Table 3 on page 8.

These samples enable you to use the WebSphere MQ JMS ASF in a standalone environment (that is, you do not need a suitable application server). Also, they provide examples of how to implement these interfaces and take advantage of the WebSphere MQ JMS ASF. These examples are intended to aid both WebSphere MQ JMS users, and vendors of other application servers.

MyServerSession.java

This class implements the `javax.jms.ServerSession` interface. It associates a thread with a JMS session. Instances of this class are pooled by a ServerSessionPool (see "MyServerSessionPool.java"). As a ServerSession, it must implement the following two methods:

- `getSession()`, which returns the JMS Session associated with this ServerSession
- `start()`, which starts this ServerSession's thread and results in the JMS Session's `run()` method being invoked

MyServerSession also implements the `Runnable` interface. Therefore, the creation of the ServerSession's thread can be based on this class, and does not need a separate class.

The class uses a `wait()-notify()` mechanism that is based on the values of two boolean flags, ready and quit. This mechanism means that the ServerSession creates and starts its associated thread during its construction. However, it does not automatically execute the body of the `run()` method. The body of the `run()` method is executed only when the ready flag is set to true by the `start()` method. The ASF calls the `start()` method when it is necessary to deliver messages to the associated JMS session.

For delivery, the `run()` method of the JMS session is called. The WebSphere MQ JMS ASF will have already loaded the `run()` method with messages.

After delivery completes, the ready flag is reset to false, and the owning ServerSessionPool is notified that delivery is complete. The ServerSession then remains in a wait state until either the `start()` method is called again, or the `close()` method is invoked and ends this ServerSession's thread.

MyServerSessionPool.java

This class implements the `javax.jms.ServerSessionPool` interface, creating and controlling access to a pool of ServerSessions.

In this implementation, the pool consists of a static array of ServerSession objects that are created during the construction of the pool. The following four parameters are passed into the constructor:

- `javax.jms.Connection` *`connection`*
  The connection used to create JMS sessions.
- `int` *`capacity`*

  The size of the array of MyServerSession objects.
- `int` *`ackMode`*

  The required acknowledge mode of the JMS sessions.
- `MessageListenerFactory` *`mlf`*
  The MesssageListenerFactory that creates the message listener that is supplied to the JMS sessions. See "MessageListenerFactory.java."

The pool's constructor uses these parameters to create an array of MyServerSession objects. The supplied connection is used to create JMS sessions of the given acknowledge mode and correct domain (QueueSessions for point-to-point and TopicSessions for publish/subscribe). The sessions are supplied with a message listener. Finally, the ServerSession objects, based on the JMS sessions, are created.

This sample implementation is a static model. That is, all the ServerSessions in the pool are created when the pool is created, and after this the pool cannot grow or shrink. This approach is just for simplicity. It is possible for a ServerSessionPool to use a sophisticated algorithm to create ServerSessions dynamically, as needed.

MyServerSessionPool keeps a record of which ServerSessions are currently in use by maintaining an array of boolean values called inUse. These booleans are all initialized to false. When the `getServerSession` method is invoked and requests a ServerSession from the pool, the inUse array is searched for the first false value. When one is found, the boolean is set to true and the corresponding ServerSession is returned. If there are no false values in the inUse array, the `getServerSession` method must `wait()` until notification occurs.

Notification occurs in either of the following circumstances:

- The pool's `close()` method is called, indicating that the pool must be shut down.
- A ServerSession that is currently in use completes its workload and calls the `serverSessionFinished` method. The `serverSessionFinished` method returns the ServerSession to the pool, and sets the corresponding inUse flag to false. The ServerSession then becomes eligible for reuse.

MessageListenerFactory.java

In this sample, a message listener factory object is associated with each ServerSessionPool instance. The MessageListenerFactory class represents a very simple interface that is used to obtain an instance of a class that implements the `javax.jms.MessageListener` interface. The class contains a single method:

```
javax.jms.MessageListener createMessageListener();
```

An implementation of this interface is supplied when the ServerSessionPool is constructed. This object is used to create message listeners for the individual JMS sessions that back up the ServerSessions in the pool. This architecture means that each separate implementation of the `MessageListenerFactory` interface must have its own ServerSessionPool.

WebSphere MQ JMS includes a sample MessageListenerFactory implementation, which is discussed in "CountingMessageListenerFactory.java" on page 410.

Examples of ASF use

A set of classes, supplied with WebSphere MQ, use the WebSphere MQ JMS application server facilities described in “ASF classes and functions” on page 399 within the sample standalone application server environment described in “Application server sample code” on page 406. These classes are stored in a subdirectory of the WebSphere MQ Java samples directory. The subdirectory is called jms\asf on Windows and jms/asf on the other platforms. To determine the WebSphere MQ Java samples directory for your platform, see Table 3 on page 8.

These samples provide examples of ASF use from the perspective of a client application:

- A simple point-to-point example uses:
  - ASFClient1.java
  - Load1.java
  - CountingMessageListenerFactory.java
- A more complex point-to-point example uses:
  - ASFClient2.java
  - Load2.java
  - CountingMessageListenerFactory.java
  - LoggingMessageListenerFactory.java
- A simple publish/subscribe example uses:
  - ASFClient3.java
  - TopicLoad.java
  - CountingMessageListenerFactory.java
- A more complex publish/subscribe example uses:
  - ASFClient4.java
  - TopicLoad.java
  - CountingMessageListenerFactory.java
  - LoggingMessageListenerFactory.java
- A publish/subscribe example using a durable ConnectionConsumer uses:
  - ASFClient5.java
  - TopicLoad.java

The following sections describe each class in turn.

Load1.java

This class is a generic JMS application that loads a given queue with a number of messages, then terminates. It can either retrieve the required administered objects from a JNDI namespace, or create them explicitly, using the WebSphere MQ JMS classes that implement these interfaces. The administered objects that are required are a QueueConnectionFactory and a queue. You can use the command line options to set the number of messages with which to load the queue, and the sleep time between individual message puts.

This application has two versions of the command line syntax.

For use with JNDI, the syntax is:

```
java -Djava.library.path=library_path
     Load1 [-icf jndiICF] [-url jndiURL] [-qcfLookup qcfLookup]
           [-qLookup qLookup] [-sleep sleepTime] [-msgs numMsgs]
```

where *library_path* is the path to the WebSphere MQ Java libraries (see "The WebSphere MQ Java libraries" on page 10).

For use without JNDI, the syntax is:

```
java -Djava.library.path=library_path
     Load1 -nojndi [-qmgr qMgrName] [-q qName]
                   [-sleep sleepTime] [-msgs numMsgs]
```

Table 29 describes the parameters and gives their defaults.

Table 29. Load1 parameters and defaults

Parameter	Meaning	Default
jndiICF	Initial context factory class used for JNDI	com.sun.jndi.ldap.LdapCtxFactory
jndiURL	Provider URL used for JNDI	ldap://localhost/o=ibm,c=us
qcfLookup	JNDI lookup key used for QueueConnectionFactory	cn=qcf
qLookup	JNDI lookup key used for Queue	cn=q
qMgrName	Name of queue manager to connect to	"" (use the default queue manager)
qName	Name of queue to load	SYSTEM.DEFAULT.LOCAL.QUEUE
sleepTime	Time (in milliseconds) to pause between message puts	0 (no pause)
numMsgs	Number of messages to put	1000

If there are any errors, an error message is displayed and the application terminates.

You can use this application to simulate message load on a WebSphere MQ queue. In turn, this message load can trigger the ASF-enabled applications described in the following sections. The messages put to the queue are simple JMS TextMessage objects. These objects do not contain user-defined message properties, which could be useful to make use of different message listeners. The source code is supplied so that you can modify this load application if necessary.

CountingMessageListenerFactory.java

This file contains definitions for two classes:

- CountingMessageListener
- CountingMessageListenerFactory

CountingMessageListener is a very simple implementation of the `javax.jms.MessageListener` interface. It keeps a record of the number of times its `onMessage` method has been invoked, but does nothing with the messages it is passed.

CountingMessageListenerFactory is the factory class for CountingMessageListener. It is an implementation of the `MessageListenerFactory` interface described in "MessageListenerFactory.java" on page 408. This factory keeps a record of all the

message listeners that it produces. It also includes a method, `printStats()`, which displays usage statistics for each of these listeners.

ASFClient1.java

This application acts as a client of the WebSphere MQ JMS ASF. It sets up a single ConnectionConsumer to consume the messages in a single WebSphere MQ queue. It displays throughput statistics for each message listener that is used, and terminates after one minute.

The application can either retrieve the required administered objects from a JNDI namespace, or create them explicitly, using the WebSphere MQ JMS classes that implement these interfaces. The administered objects that are required are a QueueConnectionFactory and a queue.

This application has two versions of the command line syntax:

For use with JNDI, the syntax is:

```
java -Djava.library.path=library_path
     ASFClient1 [-icf jndiICF] [-url jndiURL] [-qcfLookup qcfLookup]
                [-qLookup qLookup] [-poolSize poolSize] [-batchSize batchSize]
```

where *library_path* is the path to the WebSphere MQ Java libraries (see "The WebSphere MQ Java libraries" on page 10).

For use without JNDI, the syntax is:

```
java -Djava.library.path=library_path
     ASFClient1 -nojndi [-qmgr qMgrName] [-q qName]
                        [-poolSize poolSize] [-batchSize batchSize]
```

Table 30 describes the parameters and gives their defaults.

Table 30. ASFClient1 parameters and defaults

Parameter	Meaning	Default
jndiICF	Initial context factory class used for JNDI	com.sun.jndi.ldap.LdapCtxFactory
jndiURL	Provider URL used for JNDI	ldap://localhost/o=ibm,c=us
qcfLookup	JNDI lookup key used for QueueConnectionFactory	cn=qcf
qLookup	JNDI lookup key used for Queue	cn=q
qMgrName	Name of queue manager to connect to	"" (use the default queue manager)
qName	Name of queue to consume from	SYSTEM.DEFAULT.LOCAL.QUEUE
poolSize	The number of ServerSessions created in the ServerSessionPool being used	5
batchSize	The maximum number of message that can be assigned to a ServerSession at a time	10

The application obtains a QueueConnection from the QueueConnectionFactory.

A ServerSessionPool, in the form of a MyServerSessionPool, is constructed using:

- The QueueConnection that was created previously

- The required poolSize
- An acknowledge mode, AUTO_ACKNOWLEDGE
- An instance of a CountingMessageListenerFactory, as described in "CountingMessageListenerFactory.java" on page 410

The connection's `createConnectionConsumer` method is invoked, passing in:

- The queue that was obtained earlier
- A null message selector (indicating that all messages should be accepted)
- The ServerSessionPool that was just created
- The batchSize that is required

The consumption of messages is then started by invoking the connection's `start()` method.

The client application displays throughput statistics for each message listener that is used, displaying statistics every 10 seconds. After one minute, the connection is closed, the server session pool is stopped, and the application terminates.

Load2.java

This class is a JMS application that loads a given queue with a number of messages, then terminates, in a similar way to Load1.java. The command line syntax is also similar to that for Load1.java (substitute Load2 for Load1 in the syntax). For details, see "Load1.java" on page 409.

The difference is that each message contains a user property called `value`, which takes a randomly selected integer value between 0 and 100. This property means that you can apply message selectors to the messages. Consequently, the messages can be shared between the two consumers that are created in the client application described in "ASFClient2.java."

LoggingMessageListenerFactory.java

This file contains definitions for two classes:

- LoggingMessageListener
- LoggingMessageListenerFactory

LoggingMessageListener is an implementation of the `javax.jms.MessageListener` interface. It takes the messages that are passed to it and writes an entry to the log file. The default log file is `./ASFClient2.log`. You can inspect this file and check the messages that are sent to the connection consumer that is using this message listener.

LoggingMessageListenerFactory is the factory class for LoggingMessageListener. It is an implementation of the `MessageListenerFactory` interface described in "MessageListenerFactory.java" on page 408.

ASFClient2.java

ASFClient2.java is a slightly more complicated client application than ASFClient1.java. It creates two ConnectionConsumers that feed off the same queue, but that apply different message selectors. The application uses a CountingMessageListenerFactory for one consumer, and a LoggingMessageListenerFactory for the other. Use of two different message listener factories means that each consumer must have its own server session pool.

The application displays statistics that relate to one ConnectionConsumer on screen, and writes statistics that relate to the other ConnectionConsumer to a log file.

The command line syntax is similar to that for "ASFClient1.java" on page 411 (substitute ASFClient2 for ASFClient1 in the syntax). Each of the two server session pools contains the number of ServerSessions set by the poolSize parameter.

There should be an uneven distribution of messages. The messages loaded onto the source queue by Load2 contain a user property, where the value is between 0 and 100, evenly and randomly distributed. The message selector `value>75` is applied to highConnectionConsumer, and the message selector `value≤75` is applied to normalConnectionConsumer. The highConnectionConsumer's messages (approximately 25% of the total load) are sent to a LoggingMessageListener. The normalConnectionConsumer's messages (approximately 75% of the total load) are sent to a CountingMessageListener.

When the client application runs, statistics that relate to the normalConnectionConsumer, and its associated CountingMessageListenerFactories, are printed to screen every 10 seconds. Statistics that relate to the highConnectionConsumer, and its associated LoggingMessageListenerFactories, are written to the log file.

You can inspect the screen and the log file to see the real destination of the messages. Add the totals for each of the CountingMessageListeners. As long as the client application does not terminate before all the messages are consumed, this accounts for approximately 75% of the load. The number of log file entries accounts for the remainder of the load. (If the client application terminates before all the messages are consumed, you can increase the application timeout.)

TopicLoad.java

This class is a JMS application that is a publish/subscribe version of the Load2 queue loader described in "Load2.java" on page 412. It publishes the required number of messages under the given topic, then terminates. Each message contains a user property called `value`, which takes a randomly selected integer value between 0 and 100.

To use this application, ensure that the broker is running and that the required setup is complete. For details, see "Additional setup for publish/subscribe mode" on page 23.

This application has two versions of the command line syntax.

For use with JNDI, the syntax is:

```
java -Djava.library.path=library_path
     TopicLoad [-icf jndiICF] [-url jndiURL] [-tcfLookup tcfLookup]
               [-tLookup tLookup] [-sleep sleepTime] [-msgs numMsgs]
```

where *library_path* is the path to the WebSphere MQ Java libraries (see "The WebSphere MQ Java libraries" on page 10).

For use without JNDI, the syntax is:

```
java -Djava.library.path=library_path
     TopicLoad -nojndi [-qmgr qMgrName] [-t tName]
                       [-sleep sleepTime] [-msgs numMsgs]
```

Table 31 describes the parameters and gives their defaults.

Table 31. TopicLoad parameters and defaults

Parameter	Meaning	Default
jndiICF	Initial context factory class used for JNDI	com.sun.jndi.ldap.LdapCtxFactory
jndiURL	Provider URL used for JNDI	ldap://localhost/o=ibm,c=us
tcfLookup	JNDI lookup key used for TopicConnectionFactory	cn=tcf
tLookup	JNDI lookup key used for Topic	cn=t
qMgrName	Name of queue manager to connect to, and broker queue manager to publish messages to	"" (use the default queue manager)
tName	Name of topic to publish to	MQJMS/ASF/TopicLoad
sleepTime	Time (in milliseconds) to pause between message puts	0 (no pause)
numMsgs	Number of messages to put	200

If there are any errors, an error message is displayed and the application terminates.

ASFClient3.java

ASFClient3.java is a client application that is a publish/subscribe version of "ASFClient1.java" on page 411. It sets up a single ConnectionConsumer to consume the messages published on a single Topic. It displays throughput statistics for each message listener that is used, and terminates after one minute.

This application has two versions of the command line syntax.

For use with JNDI, the syntax is:

```
java -Djava.library.path=library_path
     ASFClient3 [-icf jndiICF] [-url jndiURL] [-tcfLookup tcfLookup]
                [-tLookup tLookup] [-poolsize poolSize] [-batchsize batchSize]
```

where *library_path* is the path to the WebSphere MQ Java libraries (see "The WebSphere MQ Java libraries" on page 10).

For use without JNDI, the syntax is:

```
java -Djava.library.path=library_path
     ASFClient3 -nojndi [-qmgr qMgrName] [-t tName]
                        [-poolsize poolSize] [-batchsize batchSize]
```

Table 32 describes the parameters and gives their defaults.

Table 32. ASFClient3 parameters and defaults

Parameter	Meaning	Default
jndiICF	Initial context factory class used for JNDI	com.sun.jndi.ldap.LdapCtxFactory
jndiURL	Provider URL used for JNDI	ldap://localhost/o=ibm,c=us
tcfLookup	JNDI lookup key used for TopicConnectionFactory	cn=tcf

Table 32. ASFClient3 parameters and defaults (continued)

Parameter	Meaning	Default
tLookup	JNDI lookup key used for Topic	cn=t
qMgrName	Name of queue manager to connect to, and broker queue manager to publish messages to	"" (use the default queue manager)
tName	Name of topic to consume from	MQJMS/ASF/TopicLoad
poolSize	The number of ServerSessions created in the ServerSessionPool being used	5
batchSize	The maximum number of message that can be assigned to a ServerSession at a time	10

Like ASFClient1, the client application displays throughput statistics for each message listener that is used, displaying statistics every 10 seconds. After one minute, the connection is closed, the server session pool is stopped, and the application terminates.

ASFClient4.java

ASFClient4.java is a more complex publish/subscribe client application. It creates three ConnectionConsumers that all feed off the same topic, but each one applies different message selectors.

The first two consumers use *high* and *normal* message selectors, in the same way as described for the application "ASFClient2.java" on page 412. The third consumer does not use any message selector. The application uses two CountingMessageListenerFactories for the two selector-based consumers, and a LoggingMessageListenerFactory for the third consumer. Because the application uses different message listener factories, each consumer must have its own server session pool.

The application displays statistics that relate to the two selector-based consumers on screen. It writes statistics that relate to the third ConnectionConsumer to a log file.

The command line syntax is similar to that for "ASFClient3.java" on page 414 (substitute ASFClient4 for ASFClient3 in the syntax). Each of the three server session pools contains the number of ServerSessions set by the poolSize parameter.

When the client application runs, statistics that relate to the normalConnectionConsumer and the highConnectionConsumer, and their associated CountingMessageListenerFactories, are printed to screen every 10 seconds. Statistics that relate to the third ConnectionConsumer, and its associated LoggingMessageListenerFactories, are written to the log file.

You can inspect the screen and the log file to see the real destination of the messages. Add the totals for each of the CountingMessageListeners and inspect the number of log file entries.

The distribution of messages is different from the distribution obtained by a point-to-point version of the same application (ASFClient2.java). This is because, in the publish/subscribe domain, each consumer of a topic obtains its own copy of

each message published on that topic. In this application, for a given topic load, the high and normal consumers receive approximately 25% and 75% of the load, respectively. The third consumer still receives 100% of the load. Therefore, the total number of messages received is greater than 100% of the load originally published on the topic.

ASFClient5.java

This sample exercises the durable publish/subscribe ConnectionConsumer functionality in WebSphere MQ JMS.

You invoke it with the same command-line options as the ASFClient4 sample, and, as with the other samples, the TopicLoad sample application can be used to trigger the consumer that is created. For details of TopicLoad, see "TopicLoad.java" on page 413.

When invoked, ASFClient5 displays a menu of three options:

```
1. Create/reactivate a durable ConnectionConsumer
2. Unsubscribe a durable ConnectionConsumer
X. Exit
```

If you choose option 1, and this is the first time this sample has been run, a new durable ConnectionConsumer is created using the given name. It then displays one minute's worth of throughput statistics, rather like the other samples, before closing the connection and terminating.

Having created a durable consumer, messages published on the topic in question continues to arrive at the consumer's destination even though the consumer is inactive.

This can be confirmed by running ASFClient5 again, and selecting option 1. This reactivates the named durable consumer, and the statistics displayed show that any relevant messages published during the period of inactivity were subsequently delivered to the consumer.

If you run ASFClient5 again and select option 2, this unsubscribes the named durable ConnectionConsumer and discards any outstanding messages delivered to it. Do this to ensure that the broker does not continue to deliver unwanted messages.

Part 5. WebSphere MQ JMS API reference

Introduction

This part documents the WebSphere MQ JMS application programming interface. The same information is provided in the file mqjmsapi.jar, which contains the HTML pages generated by the Javadoc tool.

Chapter 15. Package com.ibm.jms

This package comprises a set of classes and interfaces which are relevant to JMS messages.

JMSBytesMessage

public class **JMSBytesMessage**
extends JMSMessage
implements BytesMessage

```
java.lang.Object
   |
   +----com.ibm.jms.JMSMessage
           |
           +----com.ibm.jms.JMSBytesMessage
```

JMSBytesMessage is used to send a message containing a stream of uninterpreted bytes. The receiver of the message supplies the interpretation of the bytes.

Its methods are based largely on those found in java.io.DataInputStream and java.io.DataOutputStream.

This message type is for client encoding of existing message formats. If possible, use one of the other self-defining message types instead.

Although JMS allows the use of message properties with byte messages, it is typically not done since the inclusion of properties affects the format.

The primitive types can be written explicitly using methods for each type; this is the recommended method. They can also be written generically as objects. For example, a call to BytesMessage.writeInt(6) is equivalent to BytesMessage.writeObject(new Integer(6)). Both forms are provided because the explicit form is convenient for static programming and the object form is needed when types are not known at compile time.

When the message is first created, and when clearBody() is called, the body of the message is in write-only mode. After the first call to the reset method has been made, the message is in read-only mode. When a message has been sent, the provider always calls reset to read its content. Likewise, when a message has been received, the provider calls reset so that the message is in read-only mode for the client.

If clearBody is called on a message in read-only mode, the message body is cleared and the message is in write-only mode.

If a client attempts to read a message in write-only mode, a MessageNotReadableException is thrown.

If a client attempts to write a message in read-only mode, a MessageNotWriteableException is thrown.

JMSBytesMessage can be used by a JMS application to read or write a message that is sent to or from a non-Java application. As this non-Java application might be hosted on a platform with different integer or floating point encoding conventions, JMSBytesMessage class includes routines to represent its numeric data types in a number of different ways.

The only character set supported by JMSBytesMessage is the Java version of UTF-8. This includes a two-byte length prefix, and is limited to strings less than 65536 bytes in long. Applications wanting to send a string in different character set have a choice of two methods:

1. Send the message as a JMSTextMessage - if it is entirely made up of strings.
2. Convert the String to a byte array and then write it into JMSBytesMessage using the writeBytes() method.

The type of numeric encoding to be used can be set by the transport code when importing or exporting the message as a byte array. The type of encoding is specified using an int, which is effectively the sum of two separate enums, as defined by the ENC_* fields. This follows the convention laid down by WebSphere MQ in the MQMD.encoding field. For convenience, the constants defined here take precisely the same values as their MQENC counterparts defined in com.ibm.mq.MQC.

Methods

clearBody

```
public void clearBody() throws JMSException;
```

Clears out the message body. All other parts of the message are left untouched.

Exceptions

- JMSException - if an internal error occurs.

getBodyLength

```
public long getBodyLength() throws JMSException,
            MessageNotReadableException;
```

Gets the number of bytes of the message body when the message is in read-only mode. The value returned can be used to allocate a byte array. The value returned is the entire length of the message body, regardless of where the pointer for reading the message is currently located.

Returns

- number of bytes in the message

Exceptions

- JMSException - with reasons
  - MQJMS_EXCEPTION_MESSAGE_NOT_READABLE
  - MQJMS_EXCEPTION_RESOURCE_ALLOCATION

readBoolean

```
public boolean readBoolean() throws JMSException;
```

Reads a `boolean` from the bytes message.

Returns

- the boolean value read.

Exceptions

- JMSException - with reasons
  - MQJMS_EXCEPTION_MESSAGE_NOT_READABLE
  - MQJMS_EXCEPTION_MESSAGE_EOF

readByte

```
public byte readByte() throws JMSException;
```

Reads a signed 8-bit value from the bytes message.

Returns

- the next byte from the bytes message as a signed 8-bit `byte`.

Exceptions

- JMSException - with reasons
  - MQJMS_EXCEPTION_MESSAGE_NOT_READABLE
  - MQJMS_EXCEPTION_MESSAGE_EOF

readBytes

```
public int readBytes(byte[] value) throws JMSException;
```

Reads a byte array from the bytes message. If there are sufficient bytes remaining in the stream the entire buffer is filled. If not, the buffer is partially filled.

Parameters

- value - the buffer into which the data is read.

Returns

- the total number of bytes read into the buffer, or -1 if there is no more data because the end of the stream has been reached.

Exceptions

- JMSException - with reason MQJMS_EXCEPTION_MESSAGE_NOT_READABLE
- MessageEOFException - if end of message stream has been reached

readBytes

```
public int readBytes(byte[] value, int length) throws JMSException;
```

Reads a portion of the bytes message.

Parameters

- value - the buffer into which the data is read.
- length - the number of bytes to read.

Returns

- the total number of bytes read into the buffer, or -1 if there is no more data because the end of the stream has been reached.

Exceptions

- JMSException - with reason MQJMS_EXCEPTION_MESSAGE_NOT_READABLE
- IndexOutOfBoundsException - if length is inconsistent with value.
- MessageEOFException - if end of message stream has been reached

readChar

```
public char readChar() throws JMSException;
```

Reads a Unicode character value from the bytes message.

Returns

- the next two bytes from the bytes message as a Unicode character.

Exceptions

- JMSException - with reasons
  - MQJMS_EXCEPTION_MESSAGE_NOT_READABLE

- MQJMS_EXCEPTION_MESSAGE_EOF

readDouble

```
public double readDouble() throws JMSException;
```

Reads a `double` from the bytes message.

Returns

- the next eight bytes from the bytes message, interpreted as a `double`.

Exceptions

- JMSException - with reasons
  - MQJMS_EXCEPTION_MESSAGE_FORMAT
  - MQJMS_EXCEPTION_MESSAGE_NOT_READABLE
  - MQJMS_EXCEPTION_MESSAGE_EOF

readFloat

```
public float readFloat() throws JMSException;
```

Reads a `float` from the bytes message.

Returns

- the next four bytes from the bytes message, interpreted as a `float`.

Exceptions

- JMSException - with reasons
  - MQJMS_EXCEPTION_MESSAGE_NOT_READABLE
  - MQJMS_EXCEPTION_MESSAGE_EOF
  - MQJMS_EXCEPTION_UNEXPECTED_ERROR
  - MQJMS_EXCEPTION_MESSAGE_FORMAT

readInt

```
public int readInt() throws JMSException;
```

Reads a signed 32-bit integer from the bytes message.

Returns

- the next four bytes from the bytes message, interpreted as an `int`.

Exceptions

- JMSException - with reasons
  - MQJMS_EXCEPTION_MESSAGE_NOT_READABLE
  - MQJMS_EXCEPTION_MESSAGE_EOF
- IOException - if an I/O error has occurred.

readLong

```
public long readLong() throws JMSException;
```

Reads a signed 64-bit integer from the bytes message.

Returns

- the next eight bytes from the bytes message, interpreted as a `long`.

Exceptions

- JMSException - with reasons
  - MQJMS_EXCEPTION_MESSAGE_NOT_READABLE

  - MQJMS_EXCEPTION_MESSAGE_EOF
- IOException - if an I/O error has occurred.

readShort

```
public short readShort() throws JMSException;
```

Reads a signed 16-bit number from the bytes message.

Returns

- the next two bytes from the bytes message, interpreted as a signed 16-bit number.

Exceptions

- MessageEOFException - if end of message stream
- JMSException - with reasons
  - MQJMS_EXCEPTION_MESSAGE_NOT_READABLE
  - MQJMS_EXCEPTION_MESSAGE_EOF
- IOException - if an I/O error has occurred.

readUnsignedByte

```
public int readUnsignedByte() throws JMSException;
```

Reads an unsigned 8-bit number from the bytes message.

Returns

- the next byte from the bytes message, interpreted as an unsigned 8-bit number.

Exceptions

- MessageEOFException - if end of message stream
- JMSException - with reasons
  - MQJMS_EXCEPTION_MESSAGE_NOT_READABLE
  - MQJMS_EXCEPTION_MESSAGE_EOF
- IOException - if an I/O error has occurred.

readUnsignedShort

```
public int readUnsignedShort() throws JMSException;
```

Reads an unsigned 16-bit number from the bytes message.

Returns

- the next two bytes from the bytes message, interpreted as an unsigned 16-bit integer.

Exceptions

- JMSException - with reasons
  - MQJMS_EXCEPTION_MESSAGE_NOT_READABLE
  - MQJMS_EXCEPTION_MESSAGE_EOF
- IOException - if an I/O error has occurred.

readUTF

```
public String readUTF() throws JMSException;
```

Reads a string that has been encoded using a modified UTF-8 format from the bytes message.

For more information on the UTF-8 format, see *"File System Safe UCS Transformation Format (FSS_UFT)"*, X/Open Preliminary Specification, X/Open Company Ltd., Document Number: P316. This information also appears in ISO/IEC 10646, Annex P.

Returns

- a Unicode string from the bytes message.

Exceptions

- JMSException - with reasons
  - MQJMS_EXCEPTION_MESSAGE_NOT_READABLE
  - MQJMS_EXCEPTION_MESSAGE_EOF
  - MQJMS_EXCEPTION_MESSAGE_FORMAT
- IOException - if an I/O error has occurred.

reset

```
public void reset() throws JMSException;
```

Puts the message in read-only mode, and repositions the stream of bytes to the beginning.

Exceptions

- JMSException - if JMS fails to reset the message due to some internal JMS error.
- MessageFormatException - if message has an invalid format

toString

```
public String toString();
```

Returns a `String` containing a formatted version of the Message.

Returns

- java.lang.String

writeBoolean

```
public void writeBoolean(boolean value) throws JMSException;
```

Writes a `boolean` to the bytes message as a 1-byte value. The value `true` is written out as the value `(byte)1`; the value `false` is written out as the value `(byte)0`.

Parameters

- value - the `boolean` value to be written.

Exceptions

- JMSException - with reasons
  - MQJMS_EXCEPTION_MESSAGE_NOT_WRITABLE
  - MQJMS_EXCEPTION_RESOURCE_ALLOCATION

writeByte

```
public void writeByte(byte value) throws JMSException;
```

Writes a `byte` to the bytes message as a 1-byte value.

Parameters

- value - the `byte` value to be written.

Exceptions

- JMSException - with reasons
  - MQJMS_EXCEPTION_MESSAGE_NOT_WRITABLE
  - MQJMS_EXCEPTION_RESOURCE_ALLOCATION

writeBytes

```
public void writeBytes(byte[] value) throws JMSException;
```

Writes a byte array to the bytes message.

Parameters

- value - the byte array to be written.

Exceptions

- JMSException - with reason
  - MQJMS_EXCEPTION_MESSAGE_NOT_WRITABLE
  - MQJMS_EXCEPTION_RESOURCE_ALLOCATION

writeBytes

```
public void writeBytes(byte[] value, int offset, int length)
             throws JMSException;
```

Writes a portion of a byte array to the bytes message.

Parameters

- value - the byte array value to be written.
- offset - the initial offset within the byte array.
- length - the number of bytes to use.

Exceptions

- JMSException - with reason
  - MQJMS_EXCEPTION_MESSAGE_NOT_WRITABLE
  - MQJMS_EXCEPTION_RESOURCE_ALLOCATION

writeChar

```
public void writeChar(char value) throws JMSException;
```

Writes a `char` to the bytes message as a 2-byte value, high byte first.

Parameters

- value - the `char` value to be written.

Exceptions

- JMSException - with reason
  - MQJMS_EXCEPTION_MESSAGE_NOT_WRITABLE
  - MQJMS_EXCEPTION_RESOURCE_ALLOCATION

writeDouble

```
public void writeDouble(double value) throws JMSException;
```

Converts the double argument to a `long` using the doubleToLongBits() method in class `Double`, and then writes that `long` value to the stream message as an 8-byte quantity, high byte first.

Parameters

- value - the `double` value to be written.

Exceptions

- JMSException - with reason
  - MQJMS_EXCEPTION_MESSAGE_NOT_WRITABLE
  - MQJMS_EXCEPTION_RESOURCE_ALLOCATION

writeFloat

```
public void writeFloat(float value) throws JMSException;
```

Converts the float argument to an `int` using the floatToIntBits() method in class `Float` , and then writes that `int` value to the stream message as a 4-byte quantity, high byte first.

Parameters

- value - the `float` value to be written.

Exceptions

- JMSException - with reason
  - MQJMS_EXCEPTION_MESSAGE_NOT_WRITABLE
  - MQJMS_EXCEPTION_RESOURCE_ALLOCATION

writeInt

```
public void writeInt(int value) throws JMSException;
```

Writes an `int` to the bytes message as four bytes, high byte first.

Parameters

- value - the `int` to be written.

Exceptions

- JMSException - with reason
  - MQJMS_EXCEPTION_MESSAGE_NOT_WRITABLE
  - MQJMS_EXCEPTION_RESOURCE_ALLOCATION

writeLong

```
public void writeLong(long value) throws JMSException;
```

Writes a `long` to the bytes message as eight bytes, high byte first.

Parameters

- value - the `long` to be written.

Exceptions

- JMSException - with reason
  - MQJMS_EXCEPTION_MESSAGE_NOT_WRITABLE
  - MQJMS_EXCEPTION_RESOURCE_ALLOCATION

writeObject

```
public void writeObject(Object value) throws JMSException;
```

Writes a Java object to the bytes message.

Note that this method only works for the 'objectified' primitive object types (Integer, Double, Long...), Strings and byte arrays.

Parameters

- value - the Java object to be written.

Exceptions

- JMSException - with reason
  - MQJMS_EXCEPTION_MESSAGE_NOT_WRITABLE
  - MQJMS_EXCEPTION_RESOURCE_ALLOCATION
  - MQJMS_EXCEPTION_MESSAGE_FORMAT

writeShort

```
public void writeShort(short value) throws JMSException;
```

Writes a `short` to the bytes message as two bytes, high byte first.

Parameters

- value - the `short` to be written.

Exceptions

- JMSException - with reason
  - MQJMS_EXCEPTION_MESSAGE_NOT_WRITABLE
  - MQJMS_EXCEPTION_RESOURCE_ALLOCATION

writeUTF

```
public void writeUTF(String value) throws JMSException;
```

Write a string to the bytes message using UTF-8 encoding in a machine-independent manner.

For more information on the UTF-8 format, see *"File System Safe UCS Transformation Format (FSS_UFT)"*, X/Open Preliminary Specification, X/Open Company Ltd., Document Number: P316. This information also appears in ISO/IEC 10646, Annex P.

Parameters

- value - the `String` value to be written.

Exceptions

- JMSException - with reason
  - MQJMS_EXCEPTION_MESSAGE_NOT_WRITABLE
  - MQJMS_EXCEPTION_RESOURCE_ALLOCATION

JMSMapMessage

public class **JMSMapMessage**
extends JMSMessage
implements MapMessage

```
java.lang.Object
   |
   +----com.ibm.jms.JMSMessage
           |
           +----com.ibm.jms.JMSMapMessage
```

JMSMapMessage is used to send a set of name/type/value triplets (see "Message body" on page 396 for details). The entries can be accessed sequentially or randomly by name. The order of the entries is undefined. It adds a map message body.

The primitive types can be read or written explicitly using methods for each type. They can also be read or written generically as objects. For example, the call MapMessage.setInt("foo", 6) is equivalent to MapMessage.setObject("foo", new Integer(6)). Both forms are provided because the explicit form is convenient for static programming and the object form is needed when types are not known at compile time.

When a client receives a MapMessage it is in read-only mode. If a client attempts to write to the message at this point a MessageNotWriteableException is thrown. If clearBody() is called the message can then be both read from and written to.

Map messages support the following conversion table. The marked cases are supported and the unmarked cases throw a JMSException. The String to primitive conversions might throw a runtime exception if the primitives valueOf() method does not accept it as a valid String representation of the primitive.

A value written as the row type can be read as the column type.

	boolean	byte	short	char	int	long	float	double	String	byte[]
boolean	X								X	
byte		X	X		X	X			X	
short			X		X	X			X	
char				X					X	
int					X	X			X	
long						X			X	
float							X	X	X	
double								X	X	
String	X	X	X	X	X	X	X	X	X	
byte[]										X

Methods

clearBody

```
public void clearBody() throws JMSException;
```

Clears the message body. All other parts of the message are left untouched.

Exceptions

- JMSException - if there is an internal JMS error.

getBoolean

```
public boolean getBoolean(String name) throws JMSException;
```

Gets the boolean value of the named key.

Parameters

- name - the name of the key

Returns

- the boolean value.

Exceptions

- JMSException - if JMS fails to read the message due to an internal error.
- MessageFormatException - if this type conversion is not valid.

getByte

```
public byte getByte(String name) throws JMSException;
```

Gets the byte value of the named key.

Parameters

- name - the name of the key

Returns

- the byte value.

Exceptions

- JMSException - if JMS fails to read the message due to an internal error.
- MessageFormatException - if this type conversion is not valid.

getBytes

```
public byte[] getBytes(String name) throws JMSException;
```

Gets the byte array value of the named key.

Parameters

- name - the name of the key

Returns

- the byte array.

Exceptions

- JMSException - if JMS fails to read the message due to an internal error.
- MessageFormatException - if this type conversion is not valid.

getChar

```
public char getChar(String name) throws JMSException;
```

Gets the Unicode char value of the named key.

Parameters

- name - the name of the boolean key

Returns

- the char.

Exceptions

- JMSException - if JMS fails to read the message due to an internal error.

- MessageFormatException - if this type conversion is not valid.

getDouble

```
public double getDouble(String name) throws JMSException;
```

Gets the double value of the named key.

Parameters

- name - the name of the key

Returns

- the double value.

Exceptions

- JMSException - if JMS fails to read the message due to an internal error.
- MessageFormatException - if this type conversion is not valid.

getFloat

```
public float getFloat(String name) throws JMSException;
```

Gets the float value of the named key.

Parameters

- name - the name of the key

Returns

- the float value.

Exceptions

- JMSException - if JMS fails to read the message due to an internal error.
- MessageFormatException - if this type conversion is not valid.

getInt

```
public int getInt(String name) throws JMSException;
```

Gets the integer value of the named key.

Parameters

- name - the name of the key

Returns

- the integer value.

Exceptions

- JMSException - if JMS fails to read the message due to an internal error.
- MessageFormatException - if this type conversion is not valid.

getLong

```
public long getLong(String name) throws JMSException;
```

Gets the long value of the named key.

Parameters

- name - the name of the key

Returns

- the long value.

Exceptions

- JMSException - if JMS fails to read the message due to an internal error.
- MessageFormatException - if this type conversion is not valid.

getMapNames

```
public Enumeration getMapNames() throws JMSException;
```

Gets an Enumeration of all the MapMessage's names.

Returns

- an enumeration of all the names in this MapMessage.

Exceptions

- JMSException - if JMS fails to read message due to an internal error.

getObject

```
public Object getObject(String name) throws JMSException;
```

Gets the Java object with the given name.

This method can be used to return, as a class instance, an object that had been stored in the Map with the equivalent setObject() method call, or its equivalent primitive setter method.

Parameters

- name - the name of the Java object

Returns

- a class which represents the object referred to by the given name. If there is no item by this name, a null value is returned.

Exceptions

- JMSException - if JMS fails to read message due to an internal error.

getShort

```
public short getShort(String name) throws JMSException;
```

Gets the short value of the named key.

Parameters

- name - the name of the key

Returns

- the short value.

Exceptions

- JMSException - if JMS fails to read the message due to an internal error.
- MessageFormatException - if this type conversion is not valid.

getString

```
public String getString(String name) throws JMSException;
```

Gets the String value of the named key.

Parameters

- name - the name of the key

Returns

- the String value.

Exceptions

- JMSException - if JMS fails to read the message due to an internal error.
- MessageFormatException - if this type conversion is not valid.

itemExists

```
public boolean itemExists(String name) throws JMSException;
```

Checks whether an item exists in this MapMessage.

Parameters

- name - the name of the item to test

Returns

- **true**if the item does exist.

Exceptions

- JMSException - if a JMS error occurs.

setBoolean

```
public void setBoolean(String name, boolean value) throws JMSException;
```

Sets a boolean value with the given name in the map.

Parameters

- name - the name of the boolean
- value - the boolean value to set in the map.

Exceptions

- JMSException - if JMS fails to write the message due to an internal error.
- MessageNotWriteableException - if the message in read-only mode.

setByte

```
public void setByte(String name, byte value) throws JMSException;
```

Sets a byte value with the given name in the map.

Parameters

- name - the name of the byte
- value - the byte value to set in the map.

Exceptions

- JMSException - if JMS fails to write the message due to an internal error.
- MessageNotWriteableException - if the message in read-only mode.

setBytes

```
public void setBytes(String name, byte[] value) throws JMSException;
```

Sets a byte array with the given name in the map.

Parameters

- name - the name of the array
- value - the byte array to set in the map.

Exceptions

- JMSException - if JMS fails to write the message due to an internal error.
- MessageNotWriteableException - if the message in read-only mode.

setBytes

```
public void setBytes(String name, byte[] value, int offset, int length)
             throws JMSException;
```

Sets a portion of a byte array in the map with the given name.

Parameters

- name - the name of the byte array
- value - the byte array to set in the Map.
- offset - the initial offset within the byte array.
- length - the number of bytes to use.

Exceptions

- JMSException - if JMS fails to write the message due to an internal error.
- MessageNotWriteableException - if the message in read-only mode.

setChar

```
public void setChar(String name, char value) throws JMSException;
```

Sets a Unicode character with the given name in the map.

Parameters

- name - the name of the character
- value - the value to set in the Map.

Exceptions

- JMSException - if JMS fails to write the message due to an internal error.
- MessageNotWriteableException - if the message in read-only mode.

setDouble

```
public void setDouble(String name, double value) throws JMSException;
```

Sets a double value with the given name in the map.

Parameters

- name - the name of the double
- value - the value to set in the Map.

Exceptions

- JMSException - if JMS fails to write the message due to an internal error.
- MessageNotWriteableException - if the message is in read-only mode.

setFloat

```
public void setFloat(String name, float value) throws JMSException;
```

Sets a floating point variable with the given name in the map.

Parameters

- name - the name of the float
- value - the value to set in the map.

Exceptions

- JMSException - if JMS fails to write the message due to an internal error.
- MessageNotWriteableException - if the message in read-only mode.

setInt

```
public void setInt(String name, int value) throws JMSException;
```

Sets an integer with the given name in the map.

Parameters

- name - the name of the integer
- value - the value to set in the Map.

Exceptions

- JMSException - if JMS fails to write the message due to an internal error.
- MessageNotWriteableException - if the message in read-only mode.

setLong

```
public void setLong(String name, long value) throws JMSException;
```

Sets a long with the given name in the map.

Parameters

- name - the name of the long
- value - the value to set in the Map.

Exceptions

- JMSException - if JMS fails to write the message due to an internal error.
- MessageNotWriteableException - if the message in read-only mode.

setObject

```
public void setObject(String name, Object value) throws JMSException;
```

Sets a Java object with the given name in the map.

This method only works for the 'objectified' primitive object types (Integer, Double, Long...), String, and byte arrays.

Parameters

- name - the name of the object
- value - the value to set in the Map.

Exceptions

- JMSException - if JMS fails to write the message due to an internal error.
- MessageNotWriteableException - if the message in read-only mode.

setShort

```
public void setShort(String name, short value) throws JMSException;
```

Sets a short value with the given name in the map.

Parameters

- name - the name of the short
- value - the short value to set in the map.

Exceptions

- JMSException - if JMS fails to write the message due to an internal error.
- MessageNotWriteableException - if the message in read-only mode.

setString

```
public void setString(String name, String value) throws JMSException;
```

Sets a String value with the given name in the map.

Parameters

- name - the name of the String
- value - the String value to set in the Map.

Exceptions

- JMSException - if JMS fails to write the message due to an internal error.
- MessageNotWriteableException - if the message in read-only mode.

toString

```
public String toString();
```

Creates a String which contains a formatted version of the Message.

Returns

- the formatted version of the Message.

JMSMessage

public abstract class **JMSMessage**
extends Object
implements MessageSerializable

```
java.lang.Object
   |
   +----com.ibm.jms.JMSMessage
```

The Message interface is the root interface of all JMS messages. It defines the JMS header and the acknowledge() method used for all messages. The header contains fields used for message routing and identification. The payload contains the application data being sent.

JMS Messages are composed of the following parts:

- Header - All messages support the same set of header fields. Header fields contain values used by both clients and providers to identify and route messages.
- Properties - Each message contains a built-in facility for supporting application-defined property values. Properties provide an efficient mechanism for supporting application defined message filtering.
- Body - JMS defines several types of message body which cover the majority of messaging styles currently in use.

JMS defines five types of message body:

- Stream - a stream of Java primitive values. It is filled and read sequentially.
- Map - a set of name-value pairs where names are Strings and values are Java primitive types. The entries can be accessed sequentially or randomly by name. The order of the entries is undefined.
- Text - a message containing a java.util.StringBuffer. The inclusion of this message type allows XML to represent content of all kinds, including the content of JMS messages.
- Object - a message that contains a serializable Java object
- Bytes - a stream of uninterpreted bytes. This message type is for literally encoding a body to match an existing message format. In many cases, it will be possible to use one of the other, easier to use, body types instead. Although JMS allows the use of message properties with byte messages, it is typically not done since the inclusion of properties might affect the format.

The JMSCorrelationID header field is used to link one message with another. It typically links a reply message with its requesting message.

JMSCorrelationID can hold either a provider-specific message ID, an application-specific String, or a provider-specific byte[] value.

A Message contains a built-in facility for supporting application-defined property values. In effect, this provides a mechanism for adding application-specific header fields to a message.

Properties allow an application, via message selectors, to have a JMS provider select and filter messages on its behalf using application-specific criteria.

Property names must obey the rules for the selector identifier of a message.

Property values can be boolean, byte, short, int, long, float, double, and String.

Property values are set prior to sending a message. When a client receives a message, its properties are in read-only mode. If a client attempts to set properties at this point, a MessageNotWriteableException is thrown. If clearProperties() is called, the properties can then be both read from and written to.

A property value might duplicate a value in a message's body or it might not. Although JMS does not define a policy for what should or should not be made a property, JMS handles data in a message's body more efficiently than data in a message's properties. For best performance, only use message properties when you need to customize a message's header. The primary reason for doing this is to support customized message selection.

Message properties support the following conversion table. The marked cases are supported. The unmarked cases throw a JMSException. The String to primitive conversions might throw a runtime exception if the primitives valueOf() method does not accept it as a valid String representation of the primitive.

A value written as the row type can be read as the column type.

	boolean	byte	short	int	long	float	double	String
boolean	X							X
byte		X	X	X	X			X
short			X	X	X			X
int				X	X			X
long					X			X
float						X	X	X
double							X	X
String	X	X	X	X	X	X	X	X

In addition to the type-specific set and get methods for properties, JMS provides the setObjectProperty() and getObjectProperty() methods. These support the same set of property types using the 'objectified' primitive values. Their purpose is to allow the property type to be decided at execution time rather than at compile time. They support the same property value conversions.

The setObjectProperty() method accepts values of class Boolean, Byte, Short, Integer, Long, Float, Double and String. An attempt to use any other class throws a JMSException.

The getObjectProperty() method only returns values of class Boolean, Byte, Short, Integer, Long, Float, Double and String.

The order of property values is not defined. To iterate through a message's property values, use getPropertyNames() to retrieve a property name Enumeration and then use the various property getter methods to retrieve their values.

A message's properties are deleted by the clearProperties() method. This leaves the message with an empty set of properties.

Getting a property value for a name which has not been set returns a null value. Only the getStringProperty() and getObjectProperty() methods can return a null value. The other property get methods throw a java.lang.NullPointerException if they are used to get a nonexistent property.

JMS reserves the `JMSX' property name prefix for JMS defined properties. The full set of these properties is defined in the Java Message Service specification. The String[] ConnectionMetaData.getJMSXPropertyNames() method returns the names of the JMSX properties supported by a connection.

JMSX properties can be referenced in message selectors whether or not they are supported by a connection. They are treated like any other absent property.

JSMX properties `set by provider on send' are available to both the producer and the consumers of the message. JSMX properties `set by provider on receive' are only available to the consumers.

JMSXGroupID and JMSXGroupSeq are simply standard properties that clients can use if they want to group messages. All providers must support them. Unless specifically noted, the values and semantics of the JMSX properties are undefined.

JMS reserves the `JMS_' property name prefix.

If an application tries to set a message property with a name that commences *JMS*, but the name is not one of the names in the following list, WebSphere MQ JMS throws an exception:

- JMSXGroupID
- JMSXGroupSeq
- JMS_IBM_Format
- JMS_IBM_MsgType
- JMS_IBM_Feedback
- JMS_IBM_PutApplType
- JMS_IBM_Report_Exception
- JMS_IBM_Report_Expiration
- JMS_IBM_Report_COA
- JMS_IBM_Report_COD
- JMS_IBM_Report_PAN
- JMS_IBM_Report_NAN
- JMS_IBM_Report_Pass_Msg_ID
- JMS_IBM_Report_Pass_Correl_ID
- JMS_IBM_Report_Discard_Msg
- JMS_IBM_Last_Msg_In_Group
- JMS_IBM_PutDate
- JMS_IBM_PutTime

Methods

acknowledge

```
public void acknowledge() throws JMSException;
```

Acknowledges this and all previous messages received. Calling this method also acknowledges all other messages received by the **session** that received this message.

Exceptions

- JMSException - with the following reasons;

- MQJMS_E_SESSION_IS_TRANSACTED
- MQJMS_E_SESSION_CLOSED
- MQJMS_EXCEPTION_MQ_NULL_QMGR
- MQJMS_EXCEPTION_MQ_QM_COMMIT_FAILED

clearProperties

```
public void clearProperties() throws JMSException;
```

Clears a message's properties.

Exceptions

- JMSException - if an internal error occurs.

getBooleanProperty

```
public boolean getBooleanProperty(String name) throws JMSException;
```

Gets the boolean property value with the given name.

Parameters

- name - the name of the boolean property.

Returns

- the boolean property value with the given name.

Exceptions

- MessageFormatException - if this type conversion is invalid.
- JMSException - with reasons
  - MQJMS_EXCEPTION_NULL_PROPERTY_NAME
  - MQJMS_E_INTERNAL_ERROR
- NullPointerException - if name is null.

getByteProperty

```
public byte getByteProperty(String name) throws JMSException;
```

Gets the byte property value with the given name.

Parameters

- name - the name of the byte property.

Returns

- the byte property value with the given name.

Exceptions

- MessageFormatException - if this type conversion is not valid.
- JMSException - with reasons
  - MQJMS_EXCEPTION_NULL_PROPERTY_NAME
  - MQJMS_E_INTERNAL_ERROR
- NullPointerException - if name is null.

getDoubleProperty

```
public double getDoubleProperty(String name) throws JMSException;
```

Gets the double property value with the given name.

Parameters

- name - the name of the double property.

Returns

- the double property value with the given name.

Exceptions

- MessageFormatException - if this type conversion is not valid.
- JMSException - with reasons
  - MQJMS_EXCEPTION_NULL_PROPERTY_NAME
  - MQJMS_E_INTERNAL_ERROR
- NullPointerException - if name is null.

getFloatProperty

```
public float getFloatProperty(String name) throws JMSException;
```

Gets the float property value with the given name.

Parameters

- name - the name of the float property.

Returns

- the float property value with the given name.

Exceptions

- MessageFormatException - if this type conversion is not valid.
- JMSException - with reasons
  - MQJMS_EXCEPTION_NULL_PROPERTY_NAME
  - MQJMS_E_INTERNAL_ERROR
- NullPointerException - if name is null.

getIntProperty

```
public int getIntProperty(String name) throws JMSException;
```

Gets the integer property value with the given name.

Parameters

- name - the name of the integer property

Returns

- the integer property value with the given name.

Exceptions

- MessageFormatException - if this type conversion is not valid.
- JMSException - with reasons
  - MQJMS_EXCEPTION_NULL_PROPERTY_NAME
  - MQJMS_E_INTERNAL_ERROR
- NullPointerException - if name is null.

getJMSCorrelationID

```
public String getJMSCorrelationID() throws JMSException;
```

Gets the correlation ID for the message.

This method is used to return correlation ID values that are either provider-specific message IDs or application-specific Strings.

Returns

- the correlation ID of a message as a String.

Exceptions

- JMSException - with reason MQJMS_E_NO_UTF8

getJMSCorrelationIDAsBytes

```
public byte[] getJMSCorrelationIDAsBytes() throws JMSException;
```

Gets the correlation ID as an array of bytes for the message.

The use of a byte[] value for JMSCorrelationID is not portable.

Returns

- the correlation ID of a message as an array of bytes.

Exceptions

- JMSException - if an internal error occurs.

getJMSDeliveryMode

```
public int getJMSDeliveryMode() throws JMSException;
```

Gets the delivery mode for this message.

Returns

- the delivery mode of this message.

Exceptions

- JMSException - if an internal error occurs.

getJMSDestination

```
public Destination getJMSDestination() throws JMSException;
```

Gets the destination for this message.

The destination field contains the destination to which the message is being sent.

When a message is sent, this value is ignored. After completion of the send method it holds the destination specified by the send.

When a message is received, its destination value must be equivalent to the value assigned when it was sent.

Returns

- the destination of this message.

Exceptions

- JMSException - with reason MQJMS_EXCEPTION_INVALID_DESTINATION

getJMSExpiration

```
public long getJMSExpiration() throws JMSException;
```

Gets the message's expiration value.

When a message is sent, expiration is left unassigned. After completion of the send method, it holds the expiration time of the message. This is the time-to-live value specified by the client added to the time (GMT) when the message was sent.

If the time-to-live is specified as zero, expiration is set to zero which indicates the message does not expire.

When a message's expiration time is reached, a provider will discard it. JMS does not define any form of notification of message expiration.

Clients should not receive messages that have expired, although JMS does not guarantee that this will not happen.

Returns

- the time (GMT) when the message expires.

Exceptions

- JMSException - if an internal error occurs.

getJMSMessageID

```
public String getJMSMessageID() throws JMSException;
```

Gets the message ID.

The messageID header field contains a value that uniquely identifies each message sent by a provider.

When a message is sent, messageID can be ignored. When the send() method returns, it contains a provider-assigned value.

A JMSMessageID is a String value which functions as a unique key for identifying messages in a historical repository. The exact scope of uniqueness is provider defined. It must at least cover all messages for a specific installation of a provider where an installation is some connected set of message routers.

All JMSMessageID values must start with the prefix `ID:'. Uniqueness of message ID values across different providers is not required.

There is an overhead in creating a message ID and it also increases a message's size. Some JMS providers can optimize this overhead if they are given a hint that the message ID is not used by an application. JMS message Producers provide a hint to disable the message ID. When a client sets a Producer to disable the message ID it indicates they are saying that they do not depend on the value of the message ID for the messages it produces. These messages must either have message ID set to null or, if the hint is ignored, the messageID must be set to its normal unique value.

Returns

- the message ID.

Exceptions

- JMSException - if an internal error occurs.

getJMSPriority

```
public int getJMSPriority() throws JMSException;
```

Gets the message priority.

JMS defines ten levels of priority with 0 as the lowest and 9 as the highest. In addition, clients should consider priorities 0-4 as gradations of normal priority and priorities 5-9 as gradations of expedited priority.

JMS does not require that a provider strictly implement priority ordering of messages; however, it must do its best to deliver expedited messages ahead of normal messages.

Returns

- the default message priority

Exceptions

- JMSException - if an internal error occurs.

getJMSRedelivered

```
public boolean getJMSRedelivered() throws JMSException;
```

Gets an indication of whether this message is being re-delivered.

If a client receives a message with the re-delivered indicator set, it is likely, but not guaranteed, that this message was delivered to the client earlier but the client did not acknowledge its receipt at that earlier time.

Returns

- **true** if this message is being re-delivered, or **false** if not.

Exceptions

- JMSException - if an internal error occurs.

getJMSReplyTo

```
public Destination getJMSReplyTo() throws JMSException;
```

Gets the destination to which a reply to this message can be sent.

Returns

- where to send a response to this message

Exceptions

- JMSException - with reason MQJMS_EXCEPTION_INVALID_DESTINATION

getJMSTimestamp

```
public long getJMSTimestamp() throws JMSException;
```

Gets the message timestamp.

The JMSTimestamp header field contains the time a message was handed to a provider to be sent. It is not the time the message was actually transmitted because that might occur later due to transactions or other client-side queueing of messages.

When a message is sent, JMSTimestamp is ignored. When the send method returns, it contains a time somewhere between the call and the return. It is in the format of a normal Java millisecond time value.

Returns

- the message timestamp

Exceptions

- JMSException - if an internal error occurs.

getJMSType

```
public String getJMSType() throws JMSException;
```

Gets the message type.

Returns

- the message type

Exceptions

- JMSException - if an internal error occurs.

getLongProperty

```
public long getLongProperty(String name) throws JMSException;
```

Gets the long property value with the given name.

Parameters

- name - the name of the long property

Returns

- the long property value with the given name.

Exceptions

- JMSException - with reasons
  - MQJMS_EXCEPTION_NULL_PROPERTY_NAME
  - MQJMS_E_INTERNAL_ERROR
- NullPointerException - if name is null

getObjectProperty

```
public Object getObjectProperty(String name) throws JMSException;
```

Gets the Java object property value with the given name.

This method can be used to return, in objectified format, an object that had been stored as a property in the Message with the equivalent setObject() method call, or its equivalent primitive setter method.

Parameters

- name - the name of the Java object property.

Returns

- the Java object property value with the given name in objectified format (that is, if it set as an `int` , then an `Integer` is returned). If there is no property with this name, a null value is returned.

Exceptions

- JMSException - with reasons
  - MQJMS_EXCEPTION_NULL_PROPERTY_NAME
  - MQJMS_E_INTERNAL_ERROR
- NullPointerException - if name is null

getPropertyNames

```
public Enumeration getPropertyNames() throws JMSException;
```

Gets an `Enumeration` of all the property names.

Returns

- an Enumeration of all the property names.

Exceptions

- JMSException - if an internal error occurs

getShortProperty

```
public short getShortProperty(String name) throws JMSException;
```

Gets the `short` property value with the given name.

Parameters

- name - the name of the short property.

Returns

- the `short` property value with the given name.

Exceptions

- JMSException - with reasons
  - MQJMS_EXCEPTION_NULL_PROPERTY_NAME
  - MQJMS_E_INTERNAL_ERROR

getStringProperty

```
public String getStringProperty(String name) throws JMSException;
```

Gets the `String` property value with the given name.

Parameters

- name - the name of the String property to retrieve.

Returns

- the `String` property value with the given name.

Exceptions

- JMSException - with reason
  - MQJMS_EXCEPTION_NULL_PROPERTY_NAME
  - MQJMS_E_INTERNAL_ERROR

propertyExists

```
public boolean propertyExists(String name) throws JMSException;
```

Indicates whether a named property exists in the message properties Hashtable.

Parameters

- name - the name of the property to test.

Returns

- **true** if the property exists, **false** if it does not.

Exceptions

- JMSException - with reason MQJMS_EXCEPTION_NULL_PROPERTY_NAME.

setBooleanProperty

```
public void setBooleanProperty(String name, boolean value)
            throws JMSException;
```

Sets a `boolean` property value with the given name in the message.

Parameters

- name - the name of the `boolean` property.
- value - the `boolean` property value to set in the Message.

Exceptions

- MessageNotWriteableException - if properties are marked read-only
- JMSException - with reasons
  - MQJMS_E_BAD_PROPERTY_NAME

setByteProperty

```
public void setByteProperty(String name, byte value) throws JMSException;
```

Sets a `byte` property value with the given name in the message.

Parameters

- name - the name of the `byte` property.
- value - the `byte` property value to set in the Message.

Exceptions

- MessageNotWriteableException - if properties are marked read-only
- JMSException - with reasons
  - MQJMS_E_BAD_PROPERTY_NAME

setDoubleProperty

```
public void setDoubleProperty(String name, double value)
             throws JMSException;
```

Sets a `double` property value with the given name in the message.

Parameters

- name - the name of the `double` property.
- value - the `double` property value to set in the Message.

Exceptions

- MessageNotWriteableException - if properties are marked read-only
- JMSException - with reasons
  - MQJMS_E_BAD_PROPERTY_NAME

setFloatProperty

```
public void setFloatProperty(String name, float value)
             throws JMSException;
```

Sets a `float` property value with the given name in the message.

Parameters

- name - the name of the `float` property.
- value - the `float` property value to set in the Message.

Exceptions

- MessageNotWriteableException - if properties are marked read-only
- JMSException - with reasons
  - MQJMS_E_BAD_PROPERTY_NAME

setIntProperty

```
public void setIntProperty(String name, int value) throws JMSException;
```

Sets an integer property value with the given name in the message.

Parameters

- name - the name of the integer property.
- value - the integer property value to set in the Message.

Exceptions

- MessageNotWriteableException - if properties are marked read-only
- JMSException - with reasons
  - MQJMS_E_BAD_PROPERTY_NAME

setJMSCorrelationID

```
public void setJMSCorrelationID(String correlationID) throws JMSException;
```

Sets the correlation ID for the message.

A client can use the JMSCorrelationID header field to link one message with another. A typical use is to link a response message with its request message.

JMSCorrelationID can hold one of the following:

- A provider-specific message ID
- An application-specific String
- A provider-native byte[] value.

Since each message sent by a JMS provider is assigned a message ID, it is convenient to link messages using their message IDs. All message ID values must start with the 'ID:' prefix.

In some cases, an application (made up of several clients) needs to use an application-specific value for linking messages. For example, an application might use JMSCorrelationID to hold a value referencing some external information. Application-specified values must not start with the 'ID:' prefix; this is reserved for provider-generated message ID values.

If a provider supports the native concept of correlation ID, a JMS client might need to assign specific JMSCorrelationID values to match those expected by non-JMS clients. A byte[] value is used for this purpose. JMS providers without native correlation ID values are not required to support byte[] values. The use of a byte[] value for JMSCorrelationID is not portable.

Parameters

- correlationID - the message ID of a message being referred to.

Exceptions

- JMSException - with reasons
  - MQJMS_E_NO_UTF8
  - MQJMS_E_INVALID_HEX_STRING.

setJMSCorrelationIDAsBytes

```
public void setJMSCorrelationIDAsBytes(byte[] correlID)
             throws JMSException;
```

Sets the correlation ID as an array of bytes for the message.

If a provider supports the native concept of correlation id, a JMS client might need to assign specific JMSCorrelationID values to match those expected by non-JMS clients. JMS providers without native correlation ID values are not required to support this (and the corresponding get) method and the corresponding get method Their implementation might throw java.lang.UnsupportedOperationException.

A client can use this call to set the correlationID either to a messageID from a previous message, or to an application-specific string. Application-specific strings must not start with the characters 'ID:'.

The use of a byte[] value for JMSCorrelationID is not portable.

Parameters

- correlID - the correlation ID value as an array of bytes.

Exceptions

- JMSException - if an internal error occurs
- IndexOutOfBoundsException - if copying would cause access to data outside array bounds.
- ArrayStoreException - if an element in the source array cannot be stored into the destination array because of a type mismatch.
- NullPointerException - if either source or destination is null.

setJMSDeliveryMode

```
public void setJMSDeliveryMode(int deliveryMode) throws JMSException;
```

Sets the delivery mode for this message.

Any value set using this method is ignored when the message is sent, but this method can be used to change the value in a received message.

To alter the delivery mode when a message is sent, use the setDeliveryMode() method on the QueueSender or TopicPublisher (this method is inherited from MessageProducer).

Parameters

- deliveryMode - the delivery mode for this message.

Exceptions

- JMSException - if an internal error occurs.

setJMSDestination

```
public void setJMSDestination(Destination destination)
             throws JMSException;
```

Sets the destination for this message.

Any value set using this method is ignored when the message is sent, but this method can be used to change the value in a received message.

Parameters

- destination - the destination for this message.

Exceptions

- JMSException - if an internal error occurs.

setJMSExpiration

```
public void setJMSExpiration(long expiration) throws JMSException;
```

Sets the message's expiration value.

Any value set using this method is ignored when the message is sent, but this method can be used to change the value in a received message.

Parameters

- expiration - the message's expiration time.

Exceptions

- JMSException - if an internal error occurs.

setJMSMessageID

```
public void setJMSMessageID(String id) throws JMSException;
```

Sets the message ID.

Any value set using this method is ignored when the message is sent, but this method can be used to change the value in a received message.

Because a message ID set by this method is ignored when a message is sent, an application cannot specify the message ID of an outgoing message. As a consequence, an application cannot receive a message and then forward the same message, or send a different message, with the same message ID as that of the message it has received. The behavior of WebSphere MQ classes for Java differs in this respect. An application using WebSphere MQ classes for Java can specify the message ID of an outgoing message.

Parameters

- id - the ID of the message.

Exceptions

- JMSException - if an internal error occurs.

setJMSPriority

```
public void setJMSPriority(int priority) throws JMSException;
```

Sets the priority for this message.

Providers set this field when a message is sent. This operation can be used to change the value of a message that has been received.

JMS defines a ten levels of priority with 0 as the lowest and 9 as the highest. In addition, clients must consider priorities 0-4 as gradations of normal priority, and priorities 5-9 as gradations of expedited priority.

Parameters

- priority - the priority of this message

Exceptions

- JMSException - if an internal error occurs.

setJMSRedelivered

```
public void setJMSRedelivered(boolean redelivered) throws JMSException;
```

Sets a boolean to indicate whether this message is being re-delivered.

This field is set at the time the message is delivered. This operation can be used to change the value of a message that has been received.

Parameters

- re-delivered - an indication of whether this message is being re-delivered.

Exceptions

- JMSException - if an internal error occurs.

setJMSReplyTo

```
public void setJMSReplyTo(Destination replyTo) throws JMSException;
```

Sets the destination to which a reply to this message can be sent.

The replyTo header field contains the destination where a reply to the current message can be sent. If it is null, no reply is expected. The destination can be either a Queue or a Topic.

Messages with a null replyTo value are called JMS datagrams. Datagrams might contain a notification of some change in the sender (i.e. they signal a sender event) or they might just contain some data the sender thinks is of interest.

Messages with a replyTo value are typically expecting a response. A response is optional: it is up to the client to decide. These messages are called JMS requests. A message sent in response to a request is called a reply.

In some cases a client might wish to match a request it sent earlier with a reply it has just received. This can be done using the correlationID.

Parameters

- replyTo - where to send a response to this message

Exceptions

- JMSException - if an internal error occurs.

setJMSTimestamp

```
public void setJMSTimestamp(long timestamp) throws JMSException;
```

Sets the message timestamp.

Providers set this field when a message is sent. This operation can be used to change the value of a message that has been received.

Parameters

- timestamp - the timestamp for this message.

Exceptions

- JMSException - if an internal error occurs.

setJMSType

```
public void setJMSType(String type) throws JMSException;
```

Sets the message type.

Some JMS providers use a message repository that contains the definition of messages sent by applications. The type header field contains the name of a message's definition.

JMS does not define a standard message definition repository nor does it define a naming policy for the definitions it contains. JMS clients should use symbolic values for types that can be configured at installation time which correspond to the values defined in the current provider's message repository.

JMS clients should assign a value whether the application makes use of it or not. This insures that it is properly set for those providers that require it.

Parameters

- type - the class of message

Exceptions

- JMSException - with reason MQJMS_EXCEPTION_BAD_VALUE.

setLongProperty

```
public void setLongProperty(String name, long value) throws JMSException;
```

Sets a long property value with the given name in the message.

Parameters

- name - the name of the long property.
- value - the long property value to set in the Message.

Exceptions

- MessageNotWriteableException - if properties are marked read-only
- JMSException - with reason MQJMS_E_BAD_PROPERTY_NAME

setObjectProperty

```
public void setObjectProperty(String name, Object value)
             throws JMSException;
```

Sets a Java object property value with the given name into the message.

This method only works for the 'objectified' primitive object types (Integer, Double, Long...) and for Strings.

Parameters

- name - the name of the Java object property.
- value - the Java object property value to set in the Message.

Exceptions

- MessageNotWriteableException - if properties are marked read-only
- MessageFormatException - if the class of the object is not Number or String
- JMSException - with reason MQJMS_E_BAD_PROPERTY_NAME

setShortProperty

```
public void setShortProperty(String name, short value)
              throws JMSException;
```

Sets a short property value with the given name in the message.

Parameters

- name - the name of the short property.
- value - the short property value to set in the Message.

Exceptions

- MessageNotWriteableException - if properties are marked read-only
- JMSException - with reason MQJMS_E_BAD_PROPERTY_NAME

setStringProperty

```
public void setStringProperty(String name, String value)
              throws JMSException;
```

Sets a String property value with the given name in the message.

Parameters

- name - the name of the String property.
- value - the String property value to set in the Message.

Exceptions

- MessageNotWriteableException - if properties are marked read-only
- JMSException - with reason MQJMS_E_BAD_PROPERTY_NAME

toString

```
public String toString();
```

Gets a String containing a formatted version of the Message header.

Returns

- the String version of the message header.

JMSObjectMessage

public class **JMSObjectMessage**
extends JMSMessage
implements ObjectMessage

```
java.lang.Object
   |
   +----com.ibm.jms.JMSMessage
           |
           +----com.ibm.jms.JMSObjectMessage
```

An ObjectMessage is used to send a message that contains a serializable Java Object. It inherits from JMSMessage and adds a body containing a single serializable Java Object.

If more than one Java object must be sent, one of the collection classes can be used.

When a client receives an ObjectMessage , it is in read-only mode. If a client attempts to write to the message at this point, a MessageNotWriteableException is thrown. If clearBody() is called, the message can then be both read from and written to.

Methods

clearBody

```
public void clearBody() throws JMSException;
```

Clears the message body. No other part of the message is changed.

Exceptions

- JMSException - if the action fails to due to some internal JMS error.

getObject

```
public Serializable getObject() throws JMSException;
```

Get the Serializable Object containing this message's data. The default value is null.

Returns

- the Serializable Object containing this message's data

Exceptions

- JMSException - with reason MQJMS_E_DESERIALISE_FAILED
- InvalidClassException - if something is wrong with a class used by serialization.

setObject

```
public void setObject(Serializable object) throws JMSException;
```

Sets the Serializable object containing this message's data.

Parameters

- object - the message's data

Exceptions

- JMSException - with reasons
  - MQJMS_EXCEPTION_MESSAGE_NOT_WRITABLE

  - MQJMS_E_SERIALISE_FAILED
- MessageFormatException - if object serialization fails

toString

```
public String toString();
```

This method returns a String containing a formatted version of the Message.

Returns

- the formatted version.

JMSStreamMessage

public class **JMSStreamMessage**
extends JMSMessage
implements StreamMessage

```
java.lang.Object
   |
   +----com.ibm.jms.JMSMessage
           |
           +----com.ibm.jms.JMSStreamMessage
```

A StreamMessage is used to send a stream of Java primitives. It is filled and read sequentially. It inherits from JMSMessage and adds a stream message body. Its methods are based largely on those found in java.io.DataInputStream and java.io.DataOutputStream.

The primitive types can be read or written explicitly using methods for each type. They can also be read or written generically as objects. For example, a call to `StreamMessage.writeInt(6)` is equivalent to `StreamMessage.writeObject(new Integer(6))`. Both forms are provided because the explicit form is convenient for static programming and the object form is needed when types are not known at compile time.

When the message is first created, and when clearBody() is called, the body of the message is in write-only mode. After the first call to the reset() method has been made, the message body is in read-only mode. When a message has been sent the provider always calls the reset() method to read its content. Likewise, when a message has been received, the provider calls reset() so that the message body is in read-only mode for the client.

If clearBody() is called on a message in read-only mode, the message body is cleared and the message body is in write-only mode.

If a client attempts to read a message in write-only mode, a MessageNotReadableException is thrown.

If a client attempts to write a message in read-only mode, a MessageNotWriteableException is thrown.

Stream messages support the following conversion table. The marked cases are supported and the unmarked cases throw a JMSException. The String to primitive conversions throw a runtime exception if the primitives valueOf() method does not accept it as a valid String representation of the primitive.

A value written as the row type can be read as the column type.

	boolean	byte	short	char	int	long	float	double	String	byte[]
boolean	X								X	
byte		X	X		X	X			X	
short			X		X	X			X	
char				X					X	
int					X	X			X	
long						X			X	
float							X	X	X	
double								X	X	
String	X	X	X		X	X	X	X	X	
byte[]										X

Attempting to read a null value as a Java primitive type is treated as calling the primitive's corresponding valueOf(String) conversion method with a null value. Because char does not support a String conversion, attempting to read a null value as a char throws a NullPointerException.

Methods

clearBody

```
public void clearBody() throws JMSException;
```

Clears the message body. All other parts of the message are left untouched.

Exceptions

- JMSException - if the action fails to due to an internal JMS error.

readBoolean

```
public boolean readBoolean() throws JMSException;
```

Reads a `boolean` from the stream message.

Returns

- the `boolean` value read.

Exceptions

- JMSException - if JMS fails to read the message due to an internal JMS error.
- MessageEOFException - if an end of message stream.
- MessageFormatException - if this type conversion is not valid.
- MessageNotReadableException - if the message is in write-only mode.

readByte

```
public byte readByte() throws JMSException;
```

Reads a byte value from the stream message.

Returns

- the next byte from the stream message as a 8-bit `byte`.

Exceptions

- JMSException - if JMS fails to read the message due to an internal JMS error.
- MessageEOFException - if an end of message stream.
- MessageFormatException - if this type conversion is not valid.
- MessageNotReadableException - if the message is in write-only mode.

readBytes

```
public int readBytes(byte[] value) throws JMSException;
```

Reads a byte array field from the stream message into the specified byte[] object (the read buffer).

To read the field value, readBytes() should be successively called until it returns a value less than the length of the read buffer. The values of the bytes in the buffer following the last byte read are undefined.

If readBytes() returns a value equal to the length of the buffer, a subsequent readBytes() call must be made. If there are no more bytes to be read this call will return -1.

If the bytes array field value is null, readBytes() returns -1.

If the bytes array field value is empty, readBytes() returns 0.

After the first readBytes() call on a byte[] field value has been made, the full value of the field must be read before the next field can be read. An attempt to read the next field before that has been done will throw a MessageFormatException.

Parameters

- value - the buffer into which the data is read.

Returns

- the total number of bytes read into the buffer, or -1 if there is no more data because the end of the byte field has been reached.

Exceptions

- JMSException - if JMS fails to read the message due to an internal JMS error.
- MessageEOFException - if an end of message stream.
- MessageFormatException - if this type conversion is not valid.
- MessageNotReadableException - if the message is in write-only mode.

readChar

```
public char readChar() throws JMSException;
```

Reads a Unicode character value from the stream message.

Returns

- a Unicode character from the stream message.

Exceptions

- JMSException - if JMS fails to read the message due to an internal JMS error.
- MessageEOFException - if an end of message stream.
- MessageFormatException - if this type conversion is not valid.
- MessageNotReadableException - if the message is in write-only mode.

readDouble

```
public double readDouble() throws JMSException;
```

Reads a `double` from the stream message.

Returns

- a `double` value from the stream message.

Exceptions

- JMSException - if JMS fails to read the message due to an internal JMS error.
- MessageEOFException - if an end of message stream.
- MessageFormatException - if this type conversion is not valid.
- MessageNotReadableException - if the message is in write-only mode.

readFloat

```
public float readFloat() throws JMSException;
```

Reads a `float` from the stream message.

Returns

- a `float` value from the stream message.

Exceptions

- JMSException - if JMS fails to read the message due to an internal JMS error.
- MessageEOFException - if an end of message stream.
- MessageFormatException - if this type conversion is not valid.
- MessageNotReadableException - if the message is in write-only mode.

readInt

```
public int readInt() throws JMSException;
```

Reads a 32-bit integer from the stream message.

Returns

- a 32-bit integer value from the stream message, interpreted as an `int`.

Exceptions

- JMSException - if JMS fails to read the message due to an internal JMS error.
- MessageEOFException - if an end of message stream.
- MessageFormatException - if this type conversion is not valid.
- MessageNotReadableException - if the message is in write-only mode.

readLong

```
public long readLong() throws JMSException;
```

Reads a 64-bit integer from the stream message.

Returns

- a 64-bit integer value from the stream message, interpreted as a `long`.

Exceptions

- JMSException - if JMS fails to read the message due to an internal JMS error.
- MessageEOFException - if an end of message stream.
- MessageFormatException - if this type conversion is not valid.
- MessageNotReadableException - if the message is in write-only mode.

readObject

```
public Object readObject() throws JMSException;
```

Reads a Java object from the stream message.

This method can be used to return in 'objectified' format, an object that had been written to the stream with the equivalent writeObject() method call, or its equivalent primitive write() method.

Returns

- a Java object from the stream message, in 'objectified' format.

Exceptions

- JMSException - if JMS fails to read the message due to an internal JMS error.
- MessageEOFException - if an end of message stream.
- MessageNotReadableException - if the message is in write-only mode.

readShort

```
public short readShort() throws JMSException;
```

Reads a 16-bit integer from the stream message.

Returns

- a 16-bit integer from the stream message.

Exceptions

- JMSException - if JMS fails to read the message due to an internal JMS error.
- MessageEOFException - if an end of message stream.
- MessageFormatException - if this type conversion is not valid.
- MessageNotReadableException - if the message is in write-only mode.

readString

```
public String readString() throws JMSException;
```

Reads in a String from the stream message.

Returns

- a Unicode string from the stream message.

Exceptions

- JMSException - if JMS fails to read the message due to an internal JMS error.
- MessageEOFException - if an end of message stream.
- MessageFormatException - if this type conversion is not valid.
- MessageNotReadableException - if the message is in write-only mode.

reset

```
public void reset() throws JMSException;
```

Puts the message in read-only mode, and repositions the stream to the beginning.

Exceptions

- JMSException - if JMS fails to reset the message due to an internal JMS error.
- MessageFormatException - if the message's format is not valid

toString

```
public String toString();
```

Gets a String containing a formatted version of the Message.

Returns

- the String version of the message.

writeBoolean

```
public void writeBoolean(boolean value) throws JMSException;
```

Writes a boolean to the stream message. The value true is written as the value (byte)1; the value false is written as the value (byte)0.

Parameters

- value - the boolean value to be written.

Exceptions

- JMSException - if JMS fails to write the boolean to the message due to an internal JMS error.
- MessageNotWriteableException - if the message is in read-only mode.

writeByte

```
public void writeByte(byte value) throws JMSException;
```

Writes a byte to the stream message.

Parameters

- value - the byte value to be written.

Exceptions

- JMSException - if JMS fails to write the byte to the message due to an internal JMS error.
- MessageNotWriteableException - if the message is in read-only mode.

writeBytes

```
public void writeBytes(byte[] value) throws JMSException;
```

Writes a byte array to the stream message.

Parameters

- value - the byte array to be written.

Exceptions

- JMSException - if JMS fails to write the byte array to the message due to an internal JMS error.
- MessageNotWriteableException - if the message is in read-only mode.

writeBytes

```
public void writeBytes(byte[] value, int offset, int length)
             throws JMSException;
```

Writes a portion of a byte array to the stream message.

Parameters

- value - the byte array value to be written.
- offset - the initial offset within the byte array.
- length - the number of bytes to use.

Exceptions

- JMSException - if JMS fails to write the byte array portion to the message due to an internal JMS error.
- MessageNotWriteableException - if the message is in read-only mode.

writeChar

```
public void writeChar(char value) throws JMSException;
```

Writes a `char` to the stream message.

Parameters

- value - the `char` value to be written.

Exceptions

- JMSException - if JMS fails to write the char to the message due to an internal JMS error.
- MessageNotWriteableException - if the message is in read-only mode.

writeDouble

```
public void writeDouble(double value) throws JMSException;
```

Writes a `double` to the stream message.

Parameters

- value - the `double` value to be written.

Exceptions

- JMSException - if JMS fails to write the double to the message due to an internal JMS error.
- MessageNotWriteableException - if the message is in read-only mode.

writeFloat

```
public void writeFloat(float value) throws JMSException;
```

Writes a `float` to the stream message.

Parameters

- value - the `float` value to be written.

Exceptions

- JMSException - if JMS fails to write the float to the message due to an internal JMS error.
- MessageNotWriteableException - if the message is in read-only mode.

writeInt

```
public void writeInt(int value) throws JMSException;
```

Writes an `int` to the stream message.

Parameters

- value - the `int` to be written.

Exceptions

- JMSException - if JMS fails to write the int to the message due to an internal JMS error.
- MessageNotWriteableException - if the message is in read-only mode.

writeLong

```
public void writeLong(long value) throws JMSException;
```

Writes a `long` to the stream message.

Parameters

- value - the `long` to be written.

Exceptions

- JMSException - if JMS fails to write the long to the message due to an internal JMS error.
- MessageNotWriteableException - if the message is in read-only mode.

writeObject

```
public void writeObject(Object value) throws JMSException;
```

Writes a Java object to the stream message.

This method only works for the 'objectified' primitive object types (Integer, Double, Long...), Strings and byte arrays.

Parameters

- value - the Java object to be written.

Exceptions

- JMSException - if JMS fails to write the object to the message due to an internal JMS error.
- MessageNotWriteableException - if the message is in read-only mode.
- MessageFormatException - if the object is not valid

writeShort

```
public void writeShort(short value) throws JMSException;
```

Write a `short` to the stream message.

Parameters

- value - the `short` to be written.

Exceptions

- JMSException - if JMS fails to write the short to the message due to an internal JMS error.
- MessageNotWriteableException - if the message is in read-only mode.

writeString

```
public void writeString(String value) throws JMSException;
```

Writes a String to the stream message.

Parameters

- value - the `String` value to be written.

Exceptions

- JMSException - if JMS fails to write the String to the message due to an internal JMS error.
- MessageNotWriteableException - if the message is in read-only mode.

JMSTextMessage

public class **JMSTextMessage**
extends JMSMessage
implements TextMessage

```
java.lang.Object
   |
   +----com.ibm.jms.JMSMessage
           |
           +----com.ibm.jms.JMSTextMessage
```

A TextMessage is used to send a message containing a java.lang.String. It inherits from JMSMessage and adds a text body.

When a client receives a TextMessage, it is in read-only mode. If a client attempts to write to the message at this point, a MessageNotWriteableException is thrown. If clearBody() is called, the message can then be both read from and written to.

Methods

clearBody

```
public void clearBody() throws JMSException;
```

Clears out the message body. No other part of the message is changed.

Exceptions

- JMSException - if an internal error occurs

getText

```
public String getText() throws JMSException;
```

Gets the String containing this message's data. The default value is null.

Returns

- the message data in String form.

Exceptions

- JMSException - IllegalStateException with reason MQJMS_E_BAD_CCSID.

setText

```
public void setText(String messageText) throws JMSException;
```

Sets the String containing this message's data.

Parameters

- messageText - the String containing the message's data

Exceptions

- JMSException - if an internal error occurs.
- javax.jms.MessageNotWriteableException - if the message is in read-only mode.

toString

```
public String toString();
```

Returns a String containing a formatted version of the Message.

Returns

- the message formatted as a String.

Chapter 16. Package com.ibm.mq.jms

WebSphere MQ classes for Java Message Service consist of a number of Java classes and interfaces that are based on the Sun javax.jms package of interfaces and classes. Write your clients using the Sun interfaces and classes that are described in detail in the following sections. The names of the WebSphere MQ objects that implement the Sun interfaces and classes have a prefix of MQ (unless stated otherwise in the object description). The descriptions include details about any deviations of the WebSphere MQ objects from the standard JMS definitions. This is one of two packages which contain the WebSphere MQ classes for Java Message Service that implement the Sun interfaces. The other package is com.ibm.jms. You do not usually use the implementation classes directly; you program to the JMS interfaces. Many of the interfaces do not apply when running a publish/subscribe application on a direct connection to the IBM WebSphere MQ Event Broker. Where the names of implementation classes are listed, provider-specific methods are documented.

Cleanup

public class **Cleanup**
extends MQConnectionFactory
implements Runnable

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionFactory
           |
           +----com.ibm.mq.jms.Cleanup
```

Cleanup contains utilities for dealing with nondurable subscriptions which are broken, by using the SUBSTATE(BROKER) option. The class is not applicable if you use a direct connection to a broker.

Constructors

Cleanup

```
public Cleanup();
```

Default constructor.

Cleanup

```
public Cleanup(MQConnectionFactory mqcf) throws JMSException;
```

Constructor that imports property values.

Parameters

- mqcf - the topic connection factory that provides the values.

Methods

cleanup

```
public void cleanup() throws JMSException;
```

Runs Cleanup once.

Exceptions

- IllegalStateException - if CleanupLevel is JMSC.MQJMS_CLEANUP_NONE.

getCleanupInterval

```
public long getCleanupInterval();
```

Gets the cleanup interval.

Returns

- the cleanup interval.

getCleanupLevel

```
public int getCleanupLevel();
```

Gets the cleanup level.

Returns

- the cleanup level.

getExceptionListener

```
public ExceptionListener getExceptionListener();
```

Gets the ExceptionListener.

Returns

- the exception listener.

isRunning

```
public boolean isRunning();
```

Indicates whether run() is currently active.

Returns

- `true` if active; `false` otherwise.

main

```
public static void main(String[] args)
             throws UnsupportedEncodingException;
```

Invokes the utility directly from a command line. You can use this if you use the broker-based subscription store. Syntax for bindings attach:

```
Cleanup [-m ] [-r ] [SAFE | STRONG | FORCE | NONDUR]
[-t]                      Syntax for client attach:

Cleanup -client [-m ] -host [-port ] [-channel ] [-r ] [SAFE | STRONG |
FORCE | NONDUR] [-t]
```

qmgr the name of the queue manager.

hostname
the name of the host which is running the queue manager.

port the port on which the queue manager is listening.

channel
the name of the channel.

interval
the interval between executions of cleanup, in minutes. If not set, cleanup is performed once.

-t enables tracing, to the mqjms.trc file.

SAFE | STRONG | FORCE | NONDUR
sets type of clean up. See setCleanupLevel().

run

```
public void run();
```

Runs Cleanup. It runs in the background at intervals, as determined by setCleanupLevel() and setCleanupInterval(). If the field set by setCleanupInterval() is zero, Cleanup runs once and returns. Otherwise Cleanup runs regularly at the time in milliseconds set by setCleanupInterval(). CleanupInterval must be zero with JMSC.MQJMS_CLEANUP_FORCE or JMSC.MQJMS_CLEANUP_NONDUR set, and CleanupLevel cannot be MQJMS_CLEANUP_NONE. In these cases the method fails with an IllegalStateException. Any exceptions generated are routed to the ExceptionListener.

setCleanupInterval

```
public void setCleanupInterval(long interval) throws JMSException;
```

Sets the cleanup interval.

Parameters

- interval - the cleanup interval in milliseconds.

Exceptions

- JMSException - if interval is either null or invalid.

setCleanupLevel

```
public void setCleanupLevel(int level) throws JMSException;
```

Sets the cleanup level.

Parameters

- level - the cleanup level. The following values are accepted:
  - JMSC.MQJMS_CLEANUP_NONE
  - JMSC.MQJMS_CLEANUP_SAFE - default
  - JMSC.MQJMS_CLEANUP_STRONG
  - JMSC.MQJMS_CLEANUP_FORCE
  - JMSC.MQJMS_CLEANUP_NONDUR

Exceptions

- JMSException - if level is not supported or if an illegal state is encountered.

setExceptionListener

```
public void setExceptionListener(ExceptionListener el);
```

Sets the ExceptionListener. If set, the ExceptionListener receives any exceptions caused while run() is running. Cleanup terminates shortly after issuing the exception to the ExceptionListener.

Parameters

- el - the exception listener.

setPassword

```
public void setPassword(String newPassword);
```

Sets the durable connection password

Parameters

- newPassword - the new password.

setUserID

```
public void setUserID(String newuserID);
```

Sets the durable connection user ID.

Parameters

- newuserID - the new user ID.

stop

```
public void stop();
```

Stops any running cleanup thread. Returns when run() has finished. Does nothing if run() is not running.

MQConnection

public class **MQConnection**
extends Object
implements Connection

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnection
```

A JMS MQConnection is a client's active connection to its JMS provider.

Methods

close

```
public void close() throws JMSException;
```

Moves the connection into the closed state.

Exceptions

- JMSException -

createConnectionConsumer

```
public ConnectionConsumer createConnectionConsumer(Destination destination,
                                                   String messageSelector,
                                                   ServerSessionPool sessionPool,
                                                   int maxMessages)
            throws JMSException;
```

Creates a connection consumer for this connection. This facility is only used by advanced JMS clients.

Parameters

- destination - the destination to access
- messageSelector - only messages with properties matching the message selector expression are delivered. A value of null or an empty string indicates that there is no message selector for the message consumer.
- sessionPool - the server session pool to associate with this connection consumer
- maxMessages - the maximum number of messages that can be assigned to a server session at one time

Returns

- the connection consumer.

Exceptions

- JMSException - if the Connection object fails to create a connection consumer due to some internal error or invalid arguments for sessionPool and messageSelector.
- InvalidDestinationException - if an invalid destination is specified.
- InvalidSelectorException - if the message selector is invalid.

createDurableConnectionConsumer

```
public ConnectionConsumer
createDurableConnectionConsumer(Topic topic,
                                String name,
```

```
                                String messageSelector,
                                ServerSessionPool sessionPool,
                                int maxMessageCount) throws JMSException;
```

Creates a durable connection consumer for this connection. This is facility is only used by advanced JMS clients.

Parameters

- topic - the topic to be accessed.
- name - the name of the durable subscription.
- messageSelector - delivers only those messages with properties that match the message selector expression. A value of null or an empty string indicates that there is no message selector for the message consumer.
- sessionPool - the server session pool to associate with this connection consumer.
- maxMessageCount - the maximum number of messages that can be assigned to a server session at one time.

Returns

- the connection consumer.

Exceptions

- JMSException - if JMS Connection fails to create a durable connection consumer due to some internal error or invalid arguments for sessionPool and message selector.
- InvalidSelectorException - if the message selector is invalid.

createSession

```
public Session createSession(boolean transacted, int acknowledgeMode)
              throws JMSException;
```

Creates a Session object.

Parameters

- transacted - **true** indicates that the session is transacted.
- acknowledgeMode - indicates whether the consumer or the client acknowledges any messages it receives. Possible values are:
  - Session.AUTO_ACKNOWLEDGE
  - Session.CLIENT_ACKNOWLEDGE
  - Session.DUPS_OK_ACKNOWLEDGE

  See the JMS specification for details of these values. acknowledgeMode is ignored if the session is transacted.

Returns

- a newly created session.

Exceptions

- JMSException - if the Connection object fails to create a session due to some internal error or lack of support for the specific transaction and acknowledgement mode.

getClientID

```
public String getClientID() throws JMSException;
```

Gets the client ID for this connection.

Returns

- the unique client identifier.

Exceptions

- JMSException - if JMS implementation fails to return the client ID for this Connection due to an internal error.

getExceptionListener

```
public ExceptionListener getExceptionListener() throws JMSException;
```

Gets the exception listener for this connection. A connection's ExceptionListener receives a JMSException if there is an unrecoverable problem with the connection to WebSphere MQ.

Returns

- the exception listener.

Exceptions

- JMSException -

getMetaData

```
public ConnectionMetaData getMetaData() throws JMSException;
```

Gets the meta-data for this connection.

Returns

- the connection meta data.

Exceptions

- JMSException - general exception if JMS implementation fails to get the Connection meta-data for this Connection.

setClientID

```
public void setClientID(String clientID) throws JMSException;
```

Sets the client ID for this connection.

Parameters

- clientID - the unique client identifier.

Exceptions

- JMSException - general exception if JMS implementation fails to set the client ID for this Connection due to an internal error.
- InvalidClientIDException - if JMS client specifies an invalid or duplicate client ID.

setExceptionListener

```
public void setExceptionListener(ExceptionListener listener)
             throws JMSException;
```

Sets an exception listener for this connection. A connection's ExceptionListener receives a JMSException if there is an unrecoverable problem with the connection to WebSphere MQ.

Parameters

- listener - the exception listener.

Exceptions

- JMSException -

start

```
public void start() throws JMSException;
```

Start or restart delivering incoming messages.

Exceptions

- JMSException -

stop

```
public void stop() throws JMSException;
```

Temporarily stops a connection's delivery of incoming messages. It can be restarted with the start() method. When it is stopped, it inhibits delivery to all its message consumers. Synchronous receives are blocked, and messages are not delivered to message listeners. Stopping a session has no affect on its ability to send messages. Stopping a session that is already stopped has no effect.

Exceptions

- JMSException - if the JMS implementation fails to stop the message delivery because of an internal error.

MQConnectionFactory

public class **MQConnectionFactory**
extends Object
implements ConnectionFactoryReferenceableSerializable

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionFactory
```

MQConnectionFactory is the WebSphere MQ implementation of ConnectionFactory. A ConnectionFactory encapsulates a set of connection configuration parameters that has been defined by an administrator. A client uses it to create a Connection with a JMS provider.

Constructors

MQConnectionFactory

```
public MQConnectionFactory();
```

This is the default constructor. It will create an MQConnectionFactory with all properties set to their default values.

Methods

createConnection

```
public Connection createConnection() throws JMSException;
```

Creates a connection with the default user identity. The connection is created in stopped mode. No messages are delivered until Connection.start() is explicitly called.

Returns

- a newly created connection.

Exceptions

- JMSException - if the JMS provider fails to create the connection due to some internal error.
- JMSSecurityException - if client authentication fails due to an invalid user name or password.

createConnection

```
public Connection createConnection(String userName, String password)
             throws JMSException;
```

Creates a connection with the specified user identity. The connection is created in stopped mode. No messages are delivered until Connection.start() is explicitly called.

Parameters

- userName - the caller's user name.
- password - the caller's password.

Returns

- a newly created connection.

Exceptions

- JMSException - if the JMS provider fails to create the connection due to some internal error.
- JMSSecurityException - if client authentication fails due to an invalid user name or password.

getBrokerCCSubQueue

```
public String getBrokerCCSubQueue() throws JMSException;
```

Gets the broker's queue name for nondurable connection consumers.

Returns

- the name of the queue.

Exceptions

- JMSException - if an internal error occurs.

getBrokerControlQueue

```
public String getBrokerControlQueue() throws JMSException;
```

Gets the broker's control queue name.

Returns

- the name of the control queue.

Exceptions

- JMSException - if an internal error occurs.

getBrokerPubQueue

```
public String getBrokerPubQueue() throws JMSException;
```

Gets the broker's publish queue name.

Returns

- the name of the queue.

Exceptions

- JMSException - if an internal error occurs.

getBrokerQueueManager

```
public String getBrokerQueueManager() throws JMSException;
```

Gets the name of the broker's queue manager.

Returns

- the name of the queue manager.

Exceptions

- JMSException - if an internal error occurs.

getBrokerSubQueue

```
public String getBrokerSubQueue() throws JMSException;
```

Gets the broker's queue name for nondurable subscribers.

Returns

- the name of the queue.

Exceptions

- JMSException - if an internal error occurs.

getBrokerVersion

```
public int getBrokerVersion() throws JMSException;
```

Gets the version number of the broker.

Returns

- the version number.

Exceptions

- JMSException - if an internal error occurs.

getCCDTURL

```
public URL getCCDTURL();
```

Gets the URL for the client channel definition table.

Returns

- the address of the client channel definition table.

getCCSID

```
public int getCCSID();
```

Gets the character set of the queue manager.

Returns

- the CCSID.

getChannel

```
public String getChannel();
```

Gets the name of the channel that was used.

Returns

- the name of the channel.

getCleanupInterval

```
public long getCleanupInterval() throws JMSException;
```

Gets the clean up interval.

Returns

- the clean up interval (milliseconds).

Exceptions

- JMSException - if an internal error occurs.

getCleanupLevel

```
public int getCleanupLevel() throws JMSException;
```

Gets the clean up level.

Returns

- the clean up level.

Exceptions

- JMSException - if an internal error occurs.

getClientId

```
public String getClientId();
```

Deprecated

use the getClientID() method instead.

Returns

- the client ID.

getClientID

```
public String getClientID();
```

Gets the client ID.

Returns

- the client ID for all connections made using this factory.

  Note that this method always throws an IllegalStateException when you make a direct connection to a broker.

getCloneSupport

```
public int getCloneSupport() throws JMSException;
```

Indicates whether cloning is supported.

Returns

- supported values are:
  - JMSC.MQJMS_CLONE_ENABLED
  - JMSC.MQJMS_CLONE_DISABLED

Exceptions

- JMSException - if an internal error occurs.

getConnTag

```
public byte[] getConnTag();
```

Gets the value of the queue manager connection tag. It is used by the queue manager to serialize access to the affected resources. This tag is only used when connecting to a z/OS queue manager. On other platforms it will have the value MQCT_NONE - its default value.

Returns

- the String value of the queue manager connection tag.

getDescription

```
public String getDescription();
```

Gets the description.

Returns

- the object description.

getDirectAuth

```
public int getDirectAuth() throws JMSException;
```

Gets the type of direct authentication that is required.

Returns

- one of the following:
  - JMSC.MQJMS_DIRECTAUTH_BASIC
  - JMSC.MQJMS_DIRECTAUTH_CERTIFICATE

Exceptions

- JMSException - if an internal error occurs.

getFailIfQuiesce

```
public int getFailIfQuiesce();
```

Indicates the default behavior of applications accessing a quiescing queue manager.

Returns

- possible values are:
  - JMSC.MQJMS_FIQ_YES (default)
  - JMSC.MQJMS_FIQ_NO

getHdrCompList

```
public Collection getHdrCompList();
```

Gets the list of header compression techniques which has been set.

Returns

- The Collection that holds the header compression techniques. If not set, this collection has a value of null

getHostName

```
public String getHostName();
```

Gets the name of the host. This only applies for client connections or direct TCP/IP connections to WebSphere MQ.

Returns

- the name of the host.

getLocalAddress

```
public String getLocalAddress();
```

Gets the local address.

Returns

- the local address.

getMapNameStyle

```
public boolean getMapNameStyle();
```

Allows compatibility style to be used for MapMessage element names.

Returns

- possible values are:
  - JMSC.MAP_NAME_STYLE_STANDARD
  - JMSC.MAP_NAME_STYLE_COMPATIBLE

getMessageRetention

```
public int getMessageRetention() throws JMSException;
```

Indicates what happens to unwanted messages.

Returns

- possible values are:
  - JMSC.MQJMS_MRET_YES
  - JMSC.MQJMS_MRET_NO

Exceptions

- JMSException - if an internal error occurs.

getMessageSelection

```
public int getMessageSelection() throws JMSException;
```

Indicates whether the client or the broker performs message selection.

Returns

- possible values are:
  - JMSC.MQJMS_MSEL_CLIENT
  - JMSC.MQJMS_MSEL_BROKER

Exceptions

- JMSException - if an internal error occurs.

getMQConnectionOptions

```
public int getMQConnectionOptions();
```

Gets the connection options.

Returns

- the connection options set for the queue manager

getMsgBatchSize

```
public int getMsgBatchSize();
```

Gets the message batch size.

Returns

- the maximum number of messages to be taken at once when using asynchronous delivery.

getMsgCompList

```
public Collection getMsgCompList();
```

Gets the list of message compression techniques that have been set.

Returns

- the Collection holding the message compression techniques. If not set, this collection has a value of null

getMulticast

```
public int getMulticast() throws JMSException;
```

Gets the value of the multicast attribute.

Returns

- the following values are possible:
  - JMSC.MQJMS_MULTICAST_DISABLED
  - JMSC.MQJMS_MULTICAST_NOT_RELIABLE
  - JMSC.MQJMS_MULTICAST_RELIABLE
  - JMSC.MQJMS_MULTICAST_ENABLED

Exceptions

- JMSException - if an internal error occurs.

getOptimisticPublication

```
public boolean getOptimisticPublication() throws JMSException;
```

Indicates whether transactional publish/subscribe MessageProducers should return immediately from a send/publish call rather than wait until the message has completed delivery. If **false**, failure to deliver the message will only be reported when the message is committed.

Returns

- the indicator

Exceptions

- JMSException -

getOutcomeNotification

```
public boolean getOutcomeNotification() throws JMSException;
```

Indicates whether the publish/subscribe MessageConsumers are informed of the outcome of acknowledge or commit calls after receiving messages.

Returns

- **true** if MessageConsumers are informed or **false** if not.

getPollingInterval

```
public int getPollingInterval();
```

Gets the interval between scans of all receivers during asynchronous message delivery.

Returns

- the interval in milliseconds.

getPort

```
public int getPort();
```

Gets the port number. Applies to client connections or direct TCP/IP connection to a broker.

Returns

- the port number.

getProcessDuration

```
public int getProcessDuration() throws JMSException;
```

Indicates how promptly received messages are processed. While this alone does not make any difference, quickly processing messages is a prerequisite for viewing uncommitted messages.

Returns

- the process duration (milliseconds).

getProxyHostName

```
public String getProxyHostName() throws JMSException;
```

Gets the proxy host name.

Returns

- the host name of the proxy server when establishing a direct connection, or null if no proxy server is used.

Exceptions

- JMSException - if an internal error occurs.

getProxyPort

```
public int getProxyPort() throws JMSException;
```

Gets the port number of the proxy server.

Returns

- the port number of the proxy server.

Exceptions

- JMSException - if an internal error occurs.

getPubAckInterval

```
public int getPubAckInterval() throws JMSException;
```

Gets the number of messages that can be published before requiring acknowledgement from the broker.

Returns

- the acknowledgement interval.

Exceptions

- JMSException - if an internal error occurs.

getQueueManager

```
public String getQueueManager();
```

Gets the name of the queue manager.

Returns

- the name.

getReceiveExit

```
public String getReceiveExit();
```

Gets the description of the receive exit.

Returns

- either the name of the class or a String in the form library(entryPoint) where 'library' is the name of the module where the code resides and 'entryPoint' is the entry point.

getReceiveExitInit

```
public String getReceiveExitInit();
```

Gets the initialization string for the receive exit.

Returns

- the initialization string.

getReceiveIsolation

```
public int getReceiveIsolation() throws JMSException;
```

Indicates whether calls by MessageConsumers are isolated from other operations.

Returns

- one of these values:
  - JMSC.MQJMS_RCVISOL_COMMITTED
  - JMSC.MQJMS_RCVISOL_UNCOMMITTED
  - JMSC.MQJMS_RCVISOL_DEFAULT

Exceptions

- JMSException -

getReference

```
public Reference getReference() throws NamingException;
```

Creates a reference for this queue connection factory.

Returns

- the new Reference object.

Exceptions

- NamingException - if there is a naming problem.

getRescanInterval

```
public int getRescanInterval();
```

Gets the interval between browse scans of a queue. The scan looks for messages that have not been returned by the previous browse scan.

Returns

- the interval in milliseconds.

getSecurityExit

```
public String getSecurityExit();
```

Gets the description of the security exit.

Returns

- either the name of the class or a String in the form library(entryPoint) where 'library' is the name of the module where the code resides and 'entryPoint' is the entry point.

getSecurityExitInit

```
public String getSecurityExitInit();
```

Gets the initialization string for the security exit.

Returns

- the initialization string.

getSendExit

```
public String getSendExit();
```

Gets the description of the send exit.

Returns

- either the name of the class or a String in the form library(entryPoint) where 'library' is the name of the module where the code resides and 'entryPoint' is the entry point.

getSendExitInit

```
public String getSendExitInit();
```

Gets the initialization string for the send exit.

Returns

- the initialization string.

getSparseSubscriptions

```
public boolean getSparseSubscriptions() throws JMSException;
```

Gets the sparse subscriptions attribute. A sparse subscription is one that receives infrequent matching messages. If the attribute is **true** the application must be able to open the consumer queue for browsing messages.

Returns

- **true** if sparse subscriptions are selected.

Exceptions

- JMSException - if an internal error occurs.

getSSLCertStores

```
public Collection getSSLCertStores() throws JMSException;
```

Gets the collection of CertStore objects.

Returns

- the list of CertStore objects as a Collection. This works whether setSSLCertStores() was used to set a collection of CertStore objects or a String specifying such a list.

Exceptions

- JMSException - if an unsupported type or bad value is encountered.

getSSLCertStoresAsString

```
public String getSSLCertStoresAsString() throws JMSException;
```

Gets the collection of CertStore objects as a String.

Returns

- the list of CertStore objects representing the LDAP CRLs as a String if setSSLCertStores() set it as a String in the first place.

Exceptions

- JMSException - if the CertStore objects were provided as a Collection.

getSSLCipherSuite

```
public String getSSLCipherSuite();
```

Gets the CipherSuite used for SSL encryption.

Returns

- the CipherSuite String.

getSSLFipsRequired

```
public boolean getSSLFipsRequired();
```

Indicates whether sslFips (FIPS) is required.

Returns

- the value.

getSSLPeerName

```
public String getSSLPeerName();
```

Gets the distinguished name pattern used to validate the queue manager.

Returns

- the distinguished name pattern.

getSSLResetCount

```
public int getSSLResetCount();
```

Gets the SSL reset count.

Returns

- the reset count.

getSSLSocketFactory

```
public Object getSSLSocketFactory();
```

Gets the SSLSocketFactory used with SSL encryption.

Returns

- the SSLSocketFactory.

getStatusRefreshInterval

```
public int getStatusRefreshInterval() throws JMSException;
```

Gets the status refresh interval.

Returns

- the time between transactions to refresh publish/subscribe status (milliseconds).

Exceptions

- JMSException - if an internal error occurs.

getSubscriptionStore

```
public int getSubscriptionStore() throws JMSException;
```

Gets the SUBSTORE property.

Returns

- an integer which represents a valid values of the SUBSTORE property:
  - JMSC.MQJMS_SUBSTORE_QUEUE
  - JMSC.MQJMS_SUBSTORE_BROKER
  - JMSC.MQJMS_SUBSTORE_MIGRATE

Exceptions

- JMSException - if an internal error occurs.

getSyncpointAllGets

```
public boolean getSyncpointAllGets();
```

Indicates how syncpoint is used for GET operations.

Returns

- **true** if all message GETs are done under syncpoint. Otherwise GETs for non-transacted sessions using javax.jms.Session.AUTO_ACKNOWLEDGE or javax.jms.Session.DUPS_OK_ACKNOWLEDGE acknowledge modes can do GETs outside syncpoint.

getTargetClientMatching

```
public boolean getTargetClientMatching();
```

is target client matching enabled.

Returns

- whether target client matching is enabled.

getTemporaryModel

```
public String getTemporaryModel() throws JMSException;
```

Gets the name of a model queue for creating temporary destinations.

Returns

- the name of the model queue. If it refers to a permanent dynamic model queue then you must call the MQTemporaryQueue.delete() method to destroy the queue after use.

Exceptions

- JMSException - is here only to satisfy inheritance.

getTempQPrefix

```
public String getTempQPrefix() throws JMSException;
```

Gets the prefix used to form the name of a WebSphere MQ dynamic queue.

Returns

- the prefix.

Exceptions

- JMSException - is here only to satisfy inheritance.

getTransportType

```
public int getTransportType();
```

Gets the transport type.

Returns

- the transport type. Valid types are:
  - JMSC.MQJMS_TP_BINDINGS_MQ
  - JMSC.MQJMS_TP_CLIENT_MQ_TCPIP
  - JMSC.MQJMS_TP_MQJD
  - JMSC.MQJMS_TP_DIRECT_TCPIP
  - JMSC.MQJMS_TP_DIRECT_HTTP

getUseConnectionPooling

```
public boolean getUseConnectionPooling();
```

Indicates whether connection pooling has been selected.

Returns

- **true** means that JMS has enabled connection pooling for the lifetime of any connections created by this object.

getVersion

```
public int getVersion();
```

Gets the version number.

Returns

- the version number of the class.

setBrokerCCSubQueue

```
public void setBrokerCCSubQueue(String queueName) throws JMSException;
```

Sets the name of the broker nondurable connection consumer subscriber queue.

Parameters

- queueName - the name of queue.

Exceptions

- JMSException - if queueName is either null or invalid.

setBrokerControlQueue

```
public void setBrokerControlQueue(String queueName) throws JMSException;
```

Sets the name of the broker control queue.

Parameters

- queueName - the name of the broker control queue.

Exceptions

- JMSException - if the name is either null or invalid.

setBrokerPubQueue

```
public void setBrokerPubQueue(String queueName) throws JMSException;
```

Sets the name of the broker's publish queue. Note that if this is a user-defined queue, the broker must already be aware of this queue before connecting to the broker.

Parameters

- queueName - the name of the publish queue.

Exceptions

- JMSException - if queueName is either null or invalid.

setBrokerQueueManager

```
public void setBrokerQueueManager(String queueManagerName)
            throws JMSException;
```

Sets the name of the broker's queue manager.

Parameters

- queueManagerName - the name of the queue manager.

Exceptions

- JMSException - if queueManagerName is either null or invalid.

setBrokerSubQueue

```
public void setBrokerSubQueue(String queueName) throws JMSException;
```

Gets the name of the broker nondurable subscriber queue.

Parameters

- queueName - the name of the queue.

Exceptions

- JMSException - if queueName is either null or invalid.

setBrokerVersion

```
public void setBrokerVersion(int version) throws JMSException;
```

Sets the version of the broker.

Parameters

- version - possible values are:
  - JMSC.MQJMS_BROKER_V1
  - JMSC.MQJMS_BROKER_V2

Exceptions

- JMSException - if `version` is invalid.

setCCDTURL

```
public void setCCDTURL(URL url);
```

Sets the URL for the client channel definition table.

Parameters

- url - the address of the client channel definition table.

setCCSID

```
public void setCCSID(int ccsid) throws JMSException;
```

Sets the character set to be used when connecting to the queue manager.

Parameters

- ccsid - the CCSID. The default value (819) is suitable in most situations.

Exceptions

- JMSException - if the value of `ccsid` is not permitted.

setChannel

```
public void setChannel(String channelName) throws JMSException;
```

Sets the name of the channel - applies to clients only.

Parameters

- channelName - the name of the channel.

Exceptions

- JMSException - if channelName is either null or too long.

setCleanupInterval

```
public void setCleanupInterval(long interval) throws JMSException;
```

Sets the clean up interval.

Parameters

- interval - the clean up interval (milliseconds).

Exceptions

- JMSException - if interval is either null or too long.

setCleanupLevel

```
public void setCleanupLevel(int level) throws JMSException;
```

Sets the clean up level.

Parameters

- level - permitted levels are:
  - JMSC.MQJMS_CLEANUP_AS_PROPERTY
  - JMSC.MQJMS_CLEANUP_NONE
  - JMSC.MQJMS_CLEANUP_SAFE
  - JMSC.MQJMS_CLEANUP_STRONG
  - JMSC.MQJMS_CLEANUP_NONDUR
  - JMSC.MQJMS_CLEANUP_FORCE

Exceptions

- JMSException - if level is not as listed above.

setClientId

```
public void setClientId(String id);
```

Deprecated

Use the setClientID() method instead.

Parameters

- id - the client ID.

setClientID

```
public void setClientID(String id);
```

Sets the client ID.

Note that this method always throws an IllegalStateException when you make a direct connection to a broker.

Parameters

- id - the client ID.

setCloneSupport

```
public void setCloneSupport(int type) throws JMSException;
```

Selects whether cloning is supported.

Parameters

- type - supported values are:
  - JMSC.MQJMS_CLONE_ENABLED
  - JMSC.MQJMS_CLONE_DISABLED

Exceptions

- JMSException - if type is not one of the above.

setConnTag

```
public void setConnTag(byte[] cTag);
```

Sets the value of the queue manager connection tag. It is used by the queue manager to serialize access to the affected resources. This tag is only used when connecting to a z/OS queue manager. On other platforms it must have the value MQCT_NONE - its default value.

Parameters

- cTag - the connection tag. The value is truncated if it is longer than 128 bytes.

setDescription

```
public void setDescription(String desc);
```

Sets the description.

Parameters

- desc - the description.

setDirectAuth

```
public void setDirectAuth(int authority) throws JMSException;
```

Sets the type of direct authentication that is required.

Parameters

- authority - one of the following:
  - JMSC.MQJMS_DIRECTAUTH_BASIC
  - JMSC.MQJMS_DIRECTAUTH_CERTIFICATE

Exceptions

- JMSException - if authority is neither of the above.

setFailIfQuiesce

```
public void setFailIfQuiesce(int fiq) throws JMSException;
```

Sets the default behavior of applications accessing a quiescing queue manager.

Parameters

- fiq - acceptable values are:
  - JMSC.MQJMS_FIQ_YES (default)

– JMSC.MQJMS_FIQ_NO

Exceptions

- JMSException - if `fiq` is neither of the above.

setHdrCompList

```
public void setHdrCompList(Collection compList) throws JMSException;
```

Sets the list of header compression techniques.

Parameters

- compList - The Collection of header compression techniques to set. The first item in the list to match the server list becomes the current header compression technique.

setHostName

```
public void setHostName(String hostname);
```

Sets the name of the host. This only applies for client connections or direct TCP/IP connections to WebSphere MQ.

Parameters

- hostname - the name of the host.

setLocalAddress

```
public void setLocalAddress(String address) throws JMSException;
```

Sets the local address.

Parameters

- address - the local address to be used. The format of a local address is `[ip-addr][(low-port[,high-port])]`. Here are some examples:

  9.20.4.98
  The channel binds to address 9.20.4.98 locally.

  9.20.4.98(1000)
  The channel binds to address 9.20.4.98 locally and uses port 1000.

  9.20.4.98(1000,2000)
  The channel binds to address 9.20.4.98 locally and uses a port in the range 1000 to 2000.

  (1000) The channel binds to port 1000 locally.

  (1000,2000)
  The channel binds to a port in the range 1000 to 2000 locally.

  You can specify a host name instead of an IP address.

  Specify a range of ports to allow for connections that are required internally as well as those explicitly used by an application. The number of ports required depends on the application and the facilities it uses. Typically, this is the number of sessions the application uses plus three or four additional ports. If an application is having difficulty making connections, increase the number of ports in the range.

  Note that connection pooling has an effect on how quickly a port can be reused. In JMS, connection pooling is switched on by default and it might be some minutes before a port can be reused Connection errors might occur in the meantime.

For direct connections, the local address determines which of the local network interfaces is used for multicast connections. When specifying a local address for a direct connection, do not include a port number. A port number is not valid for multicast and, if specified, causes a failure at connect time.

Exceptions

- JMSException - if the format of the local address is incorrect.

setMapNameStyle

```
public void setMapNameStyle(boolean style);
```

Allows compatibility style to be used for MapMessage element names.

Parameters

- style - possible values are:
  - JMSC.MAP_NAME_STYLE_STANDARD
  - JMSC.MAP_NAME_STYLE_COMPATIBLE

setMessageRetention

```
public void setMessageRetention(int mRet) throws JMSException;
```

Sets what happens to unwanted messages.

Parameters

- mRet - possible values are:
  - JMSC.MQJMS_MRET_YES
  - JMSC.MQJMS_MRET_NO

Exceptions

- JMSException - if mRet is not one of the above.

setMessageSelection

```
public void setMessageSelection(int selection) throws JMSException;
```

Sets whether the client or the broker performs message selection.

Parameters

- selection - possible values are:
  - JMSC.MQJMS_MSEL_CLIENT
  - JMSC.MQJMS_MSEL_BROKER

Exceptions

- JMSException - if `selection` is not one of the above.

setMQConnectionOptions

```
public void setMQConnectionOptions(int cTagOpt) throws JMSException;
```

Sets the connection options for a queue manager. This method checks that the options are valid for JMS.

Parameters

- cTagOpt - an int representing the connection options. Only one of the four options relating to connection tags and any of the four options relating to binding type can be set. The only default option set is JMSC.MQCNO_STANDARD_BINDING.

Exceptions

- JMSException - if an option or a combination of set options is invalid.

setMsgBatchSize

```
public void setMsgBatchSize(int size) throws JMSException;
```

Sets the message batch size.

Parameters

- size - the maximum number of messages to be taken at once when using asynchronous delivery.

Exceptions

- JMSException -

setMsgCompList

```
public void setMsgCompList(Collection compList) throws JMSException;
```

Sets the list of message compression techniques.

Parameters

- compList - the Collection of message compression techniques to set. The first item in the list to match the server list becomes the current message compression technique.

Exceptions

- JMSException -

setMulticast

```
public void setMulticast(int multicast) throws JMSException;
```

Sets the value of the multicast attribute.

Parameters

- multicast - the following values are possible:
  - JMSC.MQJMS_MULTICAST_DISABLED
  - JMSC.MQJMS_MULTICAST_NOT_RELIABLE
  - JMSC.MQJMS_MULTICAST_RELIABLE
  - JMSC.MQJMS_MULTICAST_ENABLED

Exceptions

- JMSException - if `multicast` does not belong to the above.

setOptimisticPublication

```
public void setOptimisticPublication(boolean newVal) throws JMSException;
```

Determines whether the publish/subscribe MessageProducers can return immediately from a send/publish call rather than wait until the message has completed delivery. If **false**, failure to deliver the message will only be reported when the message is committed.

Parameters

- newVal - **true** requires the MessageProducer to wait until the message has completed delivery, **false** allows the send() or publish() method to return more promptly . This is particularly useful if a MessageConsumer might be receiving uncommitted messages, however it does mean that

the MessageProducer will not be informed of a delivery failure until it attempts to commit its sent messages..

Exceptions

- JMSException -

setOutcomeNotification

```
public void setOutcomeNotification(boolean newVal) throws JMSException;
```

Determines whether publish/subscribe MessageConsumers are informed of the outcome of acknowledge or commit calls after receiving messages.

Parameters

- newVal - **true** if MessageConsumers are to be informed, **false** if not.

setPollingInterval

```
public void setPollingInterval(int interval) throws JMSException;
```

Sets the interval between scans of all receivers during asynchronous message delivery.

Parameters

- interval - the interval in milliseconds.

Exceptions

- JMSException -

setPort

```
public void setPort(int port) throws JMSException;
```

Sets the port for a client connection.

Parameters

- port - the new value to use.

Exceptions

- JMSException - if port is not a permitted value.

setProcessDuration

```
public void setProcessDuration(int newVal) throws JMSException;
```

Sets how promptly received messages are processed. While this alone does not make any difference, quickly processing messages is a prerequisite for viewing uncommitted messages.

Parameters

- newVal - the process duration (milliseconds).

setProxyHostName

```
public void setProxyHostName(String hostName) throws JMSException;
```

Sets the proxy host name.

Parameters

- hostName - the host name of the proxy server when establishing a direct connection, or null if no proxy server is used.

Exceptions

- JMSException - if an internal error occurs.

setProxyPort

```
public void setProxyPort(int proxyPort) throws JMSException;
```

Sets the proxy port attribute.

Parameters

- proxyPort - the port number of the proxy server when establishing a direct connection.

Exceptions

- JMSException - if an internal error occurs.

setPubAckInterval

```
public void setPubAckInterval(int interval) throws JMSException;
```

Sets the number of messages that can be published before requiring acknowledgement from the broker. Applications do not normally alter this value, and must not rely on this acknowledgement.

Parameters

- interval - the number of messages to use as an interval. The default is 25.

Exceptions

- JMSException - if an internal error occurs.

setQueueManager

```
public void setQueueManager(String queueManagerName) throws JMSException;
```

Sets the name of the queue manager.

Parameters

- queueManagerName - the queue manager which is used when selecting a channel definition. This can be in of the following forms:
  - ″qMgrName″, where the actual name of the required queue manager is passed in. The channel must connect to a queue manager of this name.
  - ″*qMgrName″, where ″*″ followed by the actual name of the required queue manager is passed in. The channel definition that is used must specify this queue manager name. This full name is passed onto the queue manager during a connect, but the queue manager that is ultimately connected to may not have the same name as specified here after the ″*″.
  - ″*″ or ″″ or a name which consists entirely of blanks is used. The actual queue manager name is disregarded when a channel definition is being selected.

Exceptions

- JMSException - if queueManagerName is either null or too long.

setReceiveExit

```
public void setReceiveExit(String receiveExit);
```

Sets the receive exit. When writing exits for use with WebSphere MQ Java, each object must also have a constructor that takes a single string argument. When WebSphere MQ creates an instance of the exit, it will pass any initialization data into the exit using this constructor.

Parameters

- receiveExit - either the name of the class or a String in the form library(entryPoint) where 'library' is the name of the module where the code resides and 'entryPoint' is the entry point.

setReceiveExitInit

```
public void setReceiveExitInit(String data);
```

Sets the initialization string for the receive exit.

Parameters

- data - the initialization string.

setReceiveIsolation

```
public void setReceiveIsolation(int newVal) throws JMSException;
```

Sets whether calls by MessageConsumers are isolated from other operations.

Parameters

- newVal - one of these values:
  - JMSC.MQJMS_RCVISOL_COMMITTED
  - JMSC.MQJMS_RCVISOL_UNCOMMITTED
  - JMSC.MQJMS_RCVISOL_DEFAULT

Exceptions

- JMSException -

setRescanInterval

```
public void setRescanInterval(int interval) throws JMSException;
```

Sets the interval between browsing a queue. The scan looks for messages that were not returned by the previous scan.

Parameters

- interval - the interval in milliseconds.

setSecurityExit

```
public void setSecurityExit(String securityExit);
```

Sets the security exit. When writing exits for use with WebSphere MQ Java, each object must also have a constructor that takes a single string argument. When WebSphere MQ creates an instance of the exit, it will pass any initialization data into the exit using this constructor.

Parameters

- securityExit - either the name of the class or a String in the form library(entryPoint) where 'library' is the name of the module where the code resides and 'entryPoint' is the entry point.

setSecurityExitInit

```
public void setSecurityExitInit(String data);
```

Sets the initialization string for the security exit.

Parameters

- data - the initialization string.

setSendExit

```
public void setSendExit(String sendExit);
```

Sets the send exit. When writing exits for use with WebSphere MQ Java, each object must also have a constructor that takes a single string argument. When WebSphere MQ creates an instance of the exit, it will pass any initialization data into the exit using this constructor.

Parameters

- sendExit - either the name of the class or a String in the form library(entryPoint) where 'library' is the name of the module where the code resides and 'entryPoint' is the entry point.

setSendExitInit

```
public void setSendExitInit(String data);
```

Gets the description of the send exit.

Parameters

- data - either the name of the class or a String in the form library(entryPoint) where 'library' is the name of the module where the code resides and 'entryPoint' is the entry point.

setSparseSubscriptions

```
public void setSparseSubscriptions(boolean sparse) throws JMSException;
```

Sets whether sparse subscriptions are selected. A sparse subscription is one that receives infrequent matching messages.

Parameters

- sparse - **true** indicates that sparse subscriptions are selected. This might be required if an application using sparse subscriptions fails to receive messages because of log overflow. If you set the attribute to **true**, the application must be able to open the consumer queue for browsing messages. The default value of this attribute is **false**.

Exceptions

- JMSException - if an internal error occurs.

setSSLCertStores

```
public void setSSLCertStores(Collection stores);
```

Provides a collection of CertStore objects used for certificate revocation list (CRL) checking. The certificate provided by the queue manager is checked against one of the CertStore objects contained within the collection; if the certificate is found, the connection attempt fails. At connect-time, each CertStore in the collection is tried in turn until one is successfully used to verify the queue manager's certificate. This property is ignored if sslCipherSuite is null. Use of this property requires Java 2 v1.4. If CertStore objects are specified using this method the MQConnectionFactory cannot be bound into a JNDI namespace. Attempting to do so will result in an exception being thrown.

You must make sure that your Java Software Development Kit (SDK) is compatible with the CRL to use CertStore successfully with a CRL hosted on an LDAP server. Some SDKs require that the CRL conforms to RFC 2587, which defines a schema for LDAP v2. Most LDAP v3 servers use RFC 2256 instead.

Parameters

- stores - the CRL - a list of CertStore objects which contain certificates that have been revoked. Null (the default) means that no checking of the queue manager's certificate is performed.

setSSLCertStores

```
public void setSSLCertStores(String stores) throws JMSException;
```

Specifies a list of LDAP servers used for certificate revocation list (CRL) checking. It allows the user to specify the URIs of LDAP CertStore objects as a String, which is converted internally to the Collection form as required by the CertStore checking routines. This method is provided to support storing the CertStore list via JMSAdmin. Each LDAP server is tried in turn until one is successfully used to verify the queue manager's certificate.

Parameters

- stores - this String must consist of a sequence of space-delimited LDAP URIs of the form `ldap://host[:port]`. If no port is specified, the LDAP default of 389 is assumed. If set to null (the default), no checking of the queue manager's certificate is performed.

Exceptions

- JMSException - if the ConnectionFactory supplied list of LDAP URIs is not valid.

setSSLCipherSuite

```
public void setSSLCipherSuite(String cipherSuite);
```

Sets the CipherSuite used for SSL encryption. Set this to the CipherSuite matching the CipherSpec set on the SVRCONN channel.

Parameters

- cipherSuite - the CipherSuite used for SSL encryption. If set to null (the default), no SSL encryption is performed.

setSSLFipsRequired

```
public void setSSLFipsRequired(boolean required);
```

Sets whether sslFips (FIPS) is required.

Parameters

- required - **true** indicates FIPS is required.

setSSLPeerName

```
public void setSSLPeerName(String peerName) throws JMSException;
```

Sets a distinguished name (DN) pattern. If sslCipherSuite is set, this pattern can ensure that the correct queue manager is used. The connection attempt fails if the distinguished name provided by the queue manager does not match this pattern.

Parameters

- peerName - the DN pattern. If set to null (the default), no checking of the queue manager's DN pattern is performed. This property is ignored if sslCipherSuite is null.

Exceptions

- JMSException - if the supplied pattern is not valid.

setSSLResetCount

```
public void setSSLResetCount(int bytes) throws JMSException;
```

Sets the SSL reset count.

Parameters

- bytes - the reset count. This must be an integer, with a value between 0 (disabled) and 999,999,999.

Exceptions

- JMSException - if `bytes` is not within the valid range.

setSSLSocketFactory

```
public void setSSLSocketFactory(Object sf);
```

Sets the SSLSocketFactory for use with SSL encryption. Use this to customize all aspects of SSL encryption. Refer to your JSSE provider's documentation for more information on constructing and customizing SSLSocketFactory instances. If a custom SSLSocketFactory is specified, the MQConnectionFactory cannot be bound into a JNDI namespace. Attempting to do so results in an exception.

Parameters

- sf - the SSLSocketFactory object. If set to null (default), the JSSE default SSLSocketFactory is used when SSL encryption is requested. This property is ignored if sslCipherSuite is null.

setStatusRefreshInterval

```
public void setStatusRefreshInterval(int interval) throws JMSException;
```

Sets the status refresh interval.

Parameters

- interval - the time between transactions to refresh publish/subscribe status (milliseconds).

Exceptions

- JMSException - if an internal error occurs.

setSubscriptionStore

```
public void setSubscriptionStore(int flag) throws JMSException;
```

Sets the SUBSTORE property.

Parameters

- flag - an integer which represents a valid values of the SUBSTORE property:
  - JMSC.MQJMS_SUBSTORE_QUEUE
  - JMSC.MQJMS_SUBSTORE_BROKER
  - JMSC.MQJMS_SUBSTORE_MIGRATE

Exceptions

- JMSException - if `flag` is not one of the above.

setSyncpointAllGets

```
public void setSyncpointAllGets(boolean flag);
```

Chooses whether to do all GET operations within a syncpoint. The default setting for this property is **false**.

Parameters

- flag - **true** means that all GETs are to be done under syncpoint. The default is **false** which allows GET operations that are not under transaction management to perform more quickly.

setTargetClientMatching

```
public void setTargetClientMatching(boolean matchClient);
```

Enable or disable target client matching. If this is set to **true**, then only MQMD messages (those from a non-JMS application) containing a replyTo will have a JMS replyTo Destination constructed with targetClient set to JMSC.MQJMS_CLIENT_NONJMS_MQ. This ensures that the reply can be understood by the originator.

If this field is set to **false**, then replies will always contain an RFH2 header, even though the receiver might not understand the reply.

Note that this applies only to point-to-point destinations. This field is set to **true** by default.

setTemporaryModel

```
public void setTemporaryModel(String queueName) throws JMSException;
```

Sets the name of a model queue for creating temporary destinations.

Parameters

- queueName - the name of the model queue. If it refers to a permanent dynamic model queue then you must call the MQTemporaryQueue.delete() method to destroy the queue after use.

Exceptions

- JMSException - if queueName is either null or too long.

setTempQPrefix

```
public void setTempQPrefix(String newTempQPrefix) throws JMSException;
```

Sets the prefix to be used to form the name of a WebSphere MQ dynamic queue.

Parameters

- newTempQPrefix - the prefix to be used to form the name of a WebSphere MQ dynamic queue. This must end with the '*' character.

Exceptions

- JMSException - if the string is null, empty, greater than 33 characters in length, or consists solely of a single asterisk (*).

setTransportType

```
public void setTransportType(int type) throws JMSException;
```

Sets the transport type.

Parameters

- type - the transport type. Valid types are:
  - JMSC.MQJMS_TP_BINDINGS_MQ

- JMSC.MQJMS_TP_CLIENT_MQ_TCPIP
- JMSC.MQJMS_TP_MQJD
- JMSC.MQJMS_TP_DIRECT_TCPIP
- JMSC.MQJMS_TP_DIRECT_HTTP

Exceptions

- JMSException - if the transport is not one of the above.

setUseConnectionPooling

```
public void setUseConnectionPooling(boolean usePooling);
```

Chooses whether to use connection pooling. If you set this to true, JMS enables connection pooling for the lifetime of any connections created through the ConnectionFactory. This also affects connections created with usePooling set to false; to disable connection pooling throughout a JVM, ensure that all ConnectionFactories used within the JVM have usePooling set to false.

Parameters

- usePooling - **true** selects connection pooling. The default, and recommended, value is true. You can disable connection pooling if, for example, your applications run in an environment that performs its own pooling.

MQConnectionMetaData

public class **MQConnectionMetaData**
extends Object
implements ConnectionMetaData

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionMetaData
```

MQConnectionMetaData provides information that describes the connection.

Constructors

MQConnectionMetaData

```
public MQConnectionMetaData();
```

Constructor which uses bindings connection.

MQConnectionMetaData

```
public MQConnectionMetaData(int conntype);
```

Constructor which permits a choice of connection types.

Parameters

- conntype - connection type. Valid types are:
  - JMSC.MQJMS_TP_BINDINGS_MQ
  - JMSC.MQJMS_TP_CLIENT_MQ_TCPIP
  - JMSC.MQJMS_TP_DIRECT_TCPIP
  - JMSC.MQJMS_TP_MQJD
  - JMSC.MQJMS_TP_DIRECT_HTTP

Methods

getJMSMajorVersion

```
public int getJMSMajorVersion();
```

Gets the JMS major version number.

Returns

- the JMS major version number.

getJMSMinorVersion

```
public int getJMSMinorVersion();
```

Gets the JMS minor version number.

Returns

- the JMS minor version number.

getJMSProviderName

```
public String getJMSProviderName();
```

Gets the JMS provider name.

Returns

- the JMS provider name.

getJMSVersion

```
public String getJMSVersion();
```

Gets the JMS API version.

Returns

- JMS_MAJOR.JMS_MINOR the JMS API version.

getJMSXPropertyNames

```
public Enumeration getJMSXPropertyNames() throws JMSException;
```

Gets an enumeration of the JMSX property names.

Returns

- an enumeration of the JMSX property names.

getProviderMajorVersion

```
public int getProviderMajorVersion();
```

Gets the JMS provider major version number.

Returns

- the JMS provider major version number.

getProviderMinorVersion

```
public int getProviderMinorVersion();
```

Gets the JMS provider minor version number.

Returns

- the JMS provider minor version number.

getProviderVersion

```
public String getProviderVersion();
```

Gets the JMS provider version.

Returns

- the JMS provider version.

toString

```
public String toString();
```

Returns a string representation of the object.

Returns

- a string representation of the object.

MQDestination

public abstract class **MQDestination**
extends Object
implements DestinationJMSDestinationSerializable

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQDestination
```

An MQDestination object encapsulates a provider-specific address.

Methods

equals

```
public boolean equals(Object obj);
```

Test for equality.

getCCSID

```
public int getCCSID();
```

Gets the number of the character set that is used by this destination.

Returns

- the CCSID. See MQMD.characterSet.

getDescription

```
public String getDescription();
```

Gets the description of the destination.

Returns

- the description.

getEncoding

```
public int getEncoding();
```

Gets the encoding that is used for this destination.

Returns

- the encoding. See MQMD.encoding.

getExpiry

```
public long getExpiry();
```

Gets the value of the expiry for this destination.

Returns

- the expiry time (milliseconds).

getFailIfQuiesce

```
public int getFailIfQuiesce();
```

Gets the status of the failIfQuiesce property of this destination.

Returns

- possible values are:

- JMSC.MQJMS_FIQ_YES - default
- JMSC.MQJMS_FIQ_NO

getPersistence

```
public int getPersistence();
```

Gets the value of the persistence of all messages sent to this destination.

Returns

- the value of persistence. See MQMD.persistence

getPriority

```
public int getPriority();
```

Gets the override priority value.

Returns

- the new priority. Possible values are:
  - JMSC.MQJMS_PRI_APP
  - JMSC.MQJMS_PRI_QDEF
  - An integer between 0 and 9, inclusive.

    -

getProperty

```
public String getProperty(String name);
```

Gets the named, user-defined property from the MQDestination URI.

Parameters

- name - The name of the property.

Returns

- the value of the named property or **null** if this property has not been defined.

getStringFromDestination

```
public String getStringFromDestination();
```

Takes a JMS Destination object and produces a transport-dependent string that encapsulates the properties of the destination.

Returns

- String - String version of the destination.

getTargetClient

```
public int getTargetClient();
```

Gets the JMS compliance indicator flag.

Returns

- possible values are:
  - JMSC.MQJMS_CLIENT_JMS_COMPLIANT
  - JMSC.MQJMS_CLIENT_NONJMS_MQ

setCCSID

```
public void setCCSID(int ccsid) throws JMSException;
```

Sets the number of the character set that is used by this destination.

Parameters

- ccsid - the CCSID. See MQMD.characterSet .

Exceptions

- JMSException - if ccsid is invalid.

setDescription

```
public void setDescription(String description);
```

Sets a description of the destination.

Parameters

- description - the description for the destination.

setEncoding

```
public void setEncoding(int encoding) throws JMSException;
```

Sets the encoding to be used for numeric fields in messages sent to this destination.

Parameters

- encoding - the encoding. See MQMD.encoding .

Exceptions

- JMSException - if encoding is not valid.

setExpiry

```
public void setExpiry(long expiry) throws JMSException;
```

Sets the expiry of all messages sent to this destination.

Parameters

- expiry - the expiry time (milliseconds).

Exceptions

- JMSException - if expiry is not valid.

setFailIfQuiesce

```
public void setFailIfQuiesce(int fiq) throws JMSException;
```

Sets the behavior of applications accessing a quiescing queue manager with this destination.

Parameters

- fiq - possible values are:
  - JMSC.MQJMS_FIQ_YES - default
  - JMSC.MQJMS_FIQ_NO

Exceptions

- JMSException - if `fiq` is not one of the above.

setPersistence

```
public void setPersistence(int persistence) throws JMSException;
```

Overrides the persistence of all messages sent to this destination.

Parameters

- persistence - the value of persistence. See MQMD.persistence

Exceptions

- JMSException - if persistence is invalid.

setPriority

```
public void setPriority(int priority) throws JMSException;
```

Overrides the priority of all messages sent to this destination.

Parameters

- priority - the new priority. Possible values are:
  - JMSC.MQJMS_PRI_APP
  - JMSC.MQJMS_PRI_QDEF
  - An integer between 0 and 9, inclusive.

Exceptions

- JMSException - if the value is invalid

setProperty

```
public void setProperty(String name, String value);
```

Sets an arbitrary, user-defined property. This property is added to the URI string that is returned by calling getStringFromDestination() Names and values for such properties must conform to the following rules:

Names can contain any character, but '=', '' characters will be escaped using standard URI syntax (that is, %3d, %25 and %26 respectively) when they are added to the Destination URI string.Names beginning with the characters 'ibm' are reserved for IBM internal use only.The names of existing MQDestination properties (for example, priority, CCSID or brokerVersion) are also reserved.Values can contain any character but '' characters will be escaped using standard URI syntax when they are added to the Destination URI string.

Destination URI strings that cannot be decoded due to syntax errors will result in a JMSException with reason MQJMS_EXCEPTION_INVALID_DESTINATION being thrown.

Names and Values added using the setProperty() method must not have any '=', '' characters replaced by escape sequences as this will be done they are added to the Destination URI string.If this method is used on a 1.3.1 JDK or lower, the Names and Values added will not be escaped and will be added to the Destination URI as they are. This can cause unpredictable results if the Names or Values contain unescaped '=', '' characters. For example, they might result in a JMSException being thrown or they might cause additional erroneous properties to be defined in the MQDestination object.

Parameters

- name - The name of the property.
- value - The value of the property.

setTargetClient

```
public void setTargetClient(int targetClient) throws JMSException;
```

Sets a flag indicating whether the remote application supports JMS.

Parameters

- targetClient - the value of the flag. Possible values are:
  - JMSC.MQJMS_CLIENT_JMS_COMPLIANT
  - JMSC.MQJMS_CLIENT_NONJMS_MQ

Exceptions

- JMSException - if the value is invalid

MQJMSLevel

public class **MQJMSLevel**
extends MQJavaLevel

```
java.lang.Object
   |
   +----com.ibm.mq.MQJavaLevel
           |
           +----com.ibm.mq.jms.MQJMSLevel
```

Displays information about the currently installed version of WebSphere MQ Classes for Java Message Service.

Add this class to the CLASSPATH, and run it using the command `java com.ibm.mq.jms.MQJMSLevel`. You can modify the output with the following parameters:

`-b` - basic format (no titles)

`-f n` - fields to display

where `n` is one, or a combination of, the following digits:: 1 - Name 2 - Version 4 - CMVC level 8 - BuildType

You can add these numbers together (for example, '3' displays both the Name and Version fields). If `-f` is not specified, the default is to display all fields.

Constructors

MQJMSLevel

```
public MQJMSLevel();
```

MQMessageConsumer

public class **MQMessageConsumer**
extends Object
implements MessageConsumer

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageConsumer
```

MQMessageConsumer is the parent interface for all message consumers. A client uses a message consumer to receive messages from a Destination .

Methods

close

```
public void close() throws JMSException;
```

Closes the message consumer. Because a provider can allocate some resources outside the Java Virtual Machine on behalf of a MessageConsumer , clients must close them when they are not needed. You cannot rely on garbage collection to reclaim these resources eventually because this might not occur soon enough. This call blocks until a receive() or active message listener has completed.

Exceptions

- JMSException - with one of the following reasons:
  - MQJMS_EXCEPTION_MQ_Q_CLOSE_FAILED or
  - MQJMS_PS_SUB_Q_DELETE_FAILED

getDestination

```
public Destination getDestination() throws JMSException;
```

Gets the message destination.

Returns

- the destination - either queue or topic.

Exceptions

- JMSException - with reason MQJMS_MESSAGECONSUMER_CLOSED

getMessageListener

```
public MessageListener getMessageListener() throws JMSException;
```

Gets the message consumer's MessageListener.

Returns

- the listener for the message consumer, or null if a listener is not set.

Exceptions

- JMSException - with reason MQJMS_MESSAGECONSUMER_CLOSED

getMessageSelector

```
public String getMessageSelector() throws JMSException;
```

Gets this message consumer's message selector expression.

Returns

- this message consumer's message selector.

Exceptions

- JMSException - with reason MQJMS_MESSAGECONSUMER_CLOSED

getNoLocal

```
public boolean getNoLocal() throws JMSException;
```

Indicates whether locally published messages are inhibited.

Returns

- **true** means that locally published messages are inhibited.

Exceptions

- JMSException - with reason MQJMS_MESSAGECONSUMER_CLOSED

receive

```
public Message receive() throws JMSException;
```

Receives the next message produced for this message consumer.

This call blocks indefinitely until a message is produced or until this message consumer is closed.

If this receive() is done within a transaction, the consumer retains the message until the transaction commits.

Returns

- the next message produced for this message consumer, or null if this message consumer is concurrently closed

Exceptions

- JMSException - if the JMS provider fails to receive the next message due to an internal error.

receive

```
public Message receive(long timeout) throws JMSException;
```

Receives the next message that arrives within the specified timeout interval.

This call blocks until a message arrives, the timeout expires, or this message consumer is closed. A `timeout` of zero never expires, and the call blocks indefinitely.

Parameters

- timeout - the timeout value (milliseconds)

Returns

- the next message produced for this message consumer, or null if the timeout expires, or this message consumer is concurrently closed

Exceptions

- JMSException - if the JMS provider fails to receive the next message due to an internal error.

receiveNoWait

```
public Message receiveNoWait() throws JMSException;
```

Receives the next message if one is immediately available.

Returns

- the next message produced for this message consumer, or null if one is not available

Exceptions

- JMSException - if the JMS provider fails to receive the next message due to an internal error.

setMessageListener

```
public void setMessageListener(MessageListener listener)
             throws JMSException;
```

Sets the message consumer's MessageListener.

Parameters

- listener - the messages are delivered to this listener.

Exceptions

- JMSException - with one of the following reasons:
  - MQJMS_QRECEIVER_CLOSED
  - MQJMS_E_SESSION_ASYNC
  - MQJMS_MESSAGECONSUMER_CLOSED
  - MQJMS_SUBSCRIBER_CLOSED
- java.lang.SecurityException -

MQMessageProducer

public class **MQMessageProducer**
extends Object
implements MessageProducer

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageProducer
```

A client uses an MQMessageProducer to send messages to a destination.

Methods

close

```
public void close() throws JMSException;
```

Closes the message producer. Because a provider can allocate some resources outside the JVM on behalf of a MessageProducer, clients must close them when they are not needed. You cannot rely on garbage collection to reclaim these resources because this might not occur soon enough.

Exceptions

- JMSException - with reason MQJMS_EXCEPTION_MQ_Q_CLOSE_FAILED

getDeliveryMode

```
public int getDeliveryMode() throws JMSException;
```

Gets the producer's default delivery mode.

Returns

- the message delivery mode for this message producer.

Exceptions

- JMSException - with reason MQJMS_MESSAGEPRODUCER_CLOSED

getDestination

```
public Destination getDestination() throws JMSException;
```

Gets the destination associated with the message producer.

Returns

- the message destination.

Exceptions

- JMSException - with reasons MQJMS_E_INTERNAL_ERROR

getDisableMessageID

```
public boolean getDisableMessageID() throws JMSException;
```

Indicates whether message IDs are disabled.

Returns

- **true** if message IDs are disabled.

Exceptions

- JMSException - with reason MQJMS_MESSAGEPRODUCER_CLOSED

getDisableMessageTimestamp

```
public boolean getDisableMessageTimestamp() throws JMSException;
```

Indicates whether message timestamps are disabled.

Returns

- **true** indicates that timestamps are disabled.

Exceptions

- JMSException - with reason MQJMS_MESSAGEPRODUCER_CLOSED

getPriority

```
public int getPriority() throws JMSException;
```

Gets the producer's default priority.

Returns

- the message priority for this message producer.

Exceptions

- JMSException - with reason MQJMS_MESSAGEPRODUCER_CLOSED

getTimeToLive

```
public long getTimeToLive() throws JMSException;
```

Gets the default length of time that a produced message will be retained by the message system.

Returns

- the length of time from its dispatch that a message is retained by default (milliseconds).

Exceptions

- JMSException - with reason MQJMS_MESSAGEPRODUCER_CLOSED

send

```
public void send(Destination destination, Message message)
             throws JMSException;
```

Sends a message to a destination if you are using a message producer for which no destination was specified when the message producer was created. The method uses the message producer's default delivery mode, default priority, and default message lifetime. Typically, you specify a destination when you create a message producer but, if you do not, you must specify a destination every time you send a message.

Parameters

- destination - the message destination.
- message - the message to send.

Exceptions

- JMSException - with one of the following reasons:
  - MQJMS_PUBLISHER_CLOSED
  - MQJMS_E_IDENT_PRO_INVALID_OP
  - MQJMS_EXCEPTION_MQ_NULL_Q
  - MQJMS_EXCEPTION_MQ_Q_OPEN_FAILED
  - MQJMS_E_SESSION_ASYNC

- MQJMS_PS_PUBLISH_MSG_FAILED
- MQJMS_EXCEPTION_INVALID_DESTINATION
- MQJMS_EXCEPTION_BAD_VALUE
- MQJMS_E_UNKNOWN_TARGET_CLIENT
- MQJMS_PS_PUBLISH_MSG_BUILD
- MQJMS_EXCEPTION_MSG_CREATE_ERROR
- MQJMS_ERR_QSENDER_CLOSED
- MQJMS_E_SESSION_CLOSED
- MQJMS_E_UNIDENT_PRO_INVALID_OP
- MQJMS_EXCEPTION_MQ_Q_CLOSE_FAILED
- MQRC_Q_TYPE_ERROR
- MQJMS_E_INTERNAL_ERROR
- MQJMS_EXCEPTION_PUT_MSG_FAILED
- MQJMS_MESSAGEPRODUCER_CLOSED

send

```
public void send(Destination destination, Message message,
                 int deliveryMode, int priority, long timeToLive)
              throws JMSException;
```

Sends a message to a destination if you are using a message producer for which no destination was specified when the message producer was created. The method specifies a delivery mode, a priority, and message lifetime. Typically, you specify a destination when you create a message producer but, if do not, you must specify a destination every time you send a message.

Parameters

- destination - the destination to which to send the message.
- message - the message to send.
- deliveryMode - the delivery mode to use
- priority - the priority for the message
- timeToLive - the lifetime of the message in milliseconds

Exceptions

- JMSException - with one of the following reasons:
  - MQJMS_PUBLISHER_CLOSED
  - MQJMS_E_IDENT_PRO_INVALID_OP
  - MQJMS_EXCEPTION_MQ_NULL_Q
  - MQJMS_EXCEPTION_MQ_Q_OPEN_FAILED
  - MQJMS_E_SESSION_ASYNC
  - MQJMS_PS_PUBLISH_MSG_FAILED
  - MQJMS_EXCEPTION_INVALID_DESTINATION
  - MQJMS_EXCEPTION_BAD_VALUE
  - MQJMS_E_UNKNOWN_TARGET_CLIENT
  - MQJMS_PS_PUBLISH_MSG_BUILD
  - MQJMS_EXCEPTION_MSG_CREATE_ERROR
  - MQJMS_ERR_QSENDER_CLOSED
  - MQJMS_E_SESSION_CLOSED
  - MQJMS_E_UNIDENT_PRO_INVALID_OP

- MQJMS_EXCEPTION_MQ_Q_CLOSE_FAILED
- MQRC_Q_TYPE_ERROR
- MQJMS_EXCEPTION_BAD_VALUE
- MQJMS_E_INTERNAL_ERROR
- MQJMS_EXCEPTION_PUT_MSG_FAILED

send

```
public void send(Message message) throws JMSException;
```

Sends a message. Uses the message producer's default delivery mode, default priority, and default time to live.

Parameters

- message - the message to be sent.

Exceptions

- JMSException - with one of the following reasons:
  - MQJMS_PS_TOPIC_NULL
  - MQJMS_E_TMPT_DELETED
  - MQJMS_EXCEPTION_BAD_VALUE
  - MQJMS_PUBLISHER_CLOSED
  - MQJMS_E_UNIDENT_PRO_INVALID_OP
  - MQJMS_EXCEPTION_MQ_NULL_Q
  - MQJMS_E_SESSION_ASYNC
  - MQJMS_PS_PUBLISH_MSG_FAILED
  - MQJMS_ERR_QSENDER_CLOSED
  - MQJMS_E_SESSION_CLOSED
  - MQJMS_E_UNKNOWN_TARGET_CLIENT
  - MQJMS_PS_PUBLISH_MSG_BUILD
  - MQJMS_EXCEPTION_MSG_CREATE_ERROR
  - MQJMS_UTIL_PS_NO_BROKER
  - MQJMS_E_11_SERVICES_NOT_SETUP
  - MQJMS_E_INTERNAL_ERROR
  - MQJMS_EXCEPTION_PUT_MSG_FAILED
  - MQJMS_MESSAGEPRODUCER_CLOSED
- java.lang.UnsupportedOperationException - if a client uses this method with a message producer for which no destination was specified when it was created.

send

```
public void send(Message message, int deliveryMode, int priority,
                 long timeToLive) throws JMSException;
```

Sends a message specifying a delivery mode, a priority, and the lifetime of the message.

Parameters

- message - the message to send.
- deliveryMode - the delivery mode to use.
- priority - the priority for the message
- timeToLive - the lifetime of the message in milliseconds.

Exceptions

- JMSException - with one of the following reasons:
  - MQJMS_PS_TOPIC_NULL
  - MQJMS_E_TMPT_DELETED
  - MQJMS_EXCEPTION_BAD_VALUE
  - MQJMS_PUBLISHER_CLOSED
  - MQJMS_E_UNIDENT_PRO_INVALID_OP
  - MQJMS_EXCEPTION_MQ_NULL_Q
  - MQJMS_E_SESSION_ASYNC
  - MQJMS_PS_PUBLISH_MSG_FAILED
  - MQJMS_ERR_QSENDER_CLOSED
  - MQJMS_E_SESSION_CLOSED
  - MQJMS_E_UNKNOWN_TARGET_CLIENT
  - MQJMS_PS_PUBLISH_MSG_BUILD
  - MQJMS_EXCEPTION_MSG_CREATE_ERROR
  - MQJMS_UTIL_PS_NO_BROKER
  - MQJMS_E_11_SERVICES_NOT_SETUP
  - MQJMS_E_INTERNAL_ERROR
  - MQJMS_EXCEPTION_PUT_MSG_FAILED

setDeliveryMode

```
public void setDeliveryMode(int deliveryMode) throws JMSException;
```

Sets the producer's default delivery mode.

Parameters

- deliveryMode - the message delivery mode for this message producer. Possible values are:
  - DeliveryMode.NON_PERSISTENT
  - DeliveryMode.PERSISTENT, the default

Exceptions

- JMSException - with one of the following reasons:
  - MQJMS_MESSAGEPRODUCER_CLOSED
  - MQJMS_EXCEPTION_BAD_VALUE

setDisableMessageID

```
public void setDisableMessageID(boolean value) throws JMSException;
```

Sets whether message IDs are disabled.

Note: This method is ignored in the WebSphere MQ classes for Java Message Service implementation.

Parameters

- value - **true** if message IDs are disabled. Message IDs are enabled by default.

Exceptions

- JMSException - with reason MQJMS_MESSAGEPRODUCER_CLOSED

setDisableMessageTimestamp

```
public void setDisableMessageTimestamp(boolean value) throws JMSException;
```

Sets whether message timestamps are disabled. They are enabled by default.

Note: This method is ignored in the WebSphere MQ classes for Java Message Service implementation.

Parameters

- value - **true** indicates that timestamps are disabled.

Exceptions

- JMSException - with reason MQJMS_MESSAGEPRODUCER_CLOSED

setPriority

```
public void setPriority(int priority) throws JMSException;
```

Sets the producer's default priority.

Parameters

- priority - the message priority for this message producer. Values can be between 0 and 9, inclusive. The default is 4.

Exceptions

- JMSException - with one of the following reasons:
  - MQJMS_MESSAGEPRODUCER_CLOSED
  - MQJMS_EXCEPTION_BAD_VALUE

setTimeToLive

```
public void setTimeToLive(long timeToLive) throws JMSException;
```

Sets the default length of time that the message system retains a produced message.

Note that this method throws a JMSException if set to other than 0 when you make a direct connection to a broker.

Parameters

- timeToLive - the length of time from its dispatch that a message is retained by default (milliseconds). The default is zero which means unlimited time.

Exceptions

- JMSException - with one of the following reasons:
  - MQJMS_MESSAGEPRODUCER_CLOSED
  - MQJMS_EXCEPTION_BAD_VALUE

MQQueue

public class **MQQueue**
extends MQDestination
implements QueueReferenceableSerializable

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQDestination
           |
           +----com.ibm.mq.jms.MQQueue
```

An MQQueue object encapsulates a provider-specific queue name. It is the way that a client specifies the identity of a queue to JMS methods.

Constructors

MQQueue

```
public MQQueue(String queueName) throws JMSException;
```

Creates a new MQQueue object.

Parameters

- queueName - is one of:
  - the name of a base WebSphere MQ queue - assumes default queue manager
  - a uniform resource identifier (URI). This form allows you to specify remote queues (queues on a queue manager other than the one to which you are connected). It also allows you to set the other properties contained in the object. The URI is in the form:

    `queue://qmgrName/queueName [name-value pairs]`

    Where: `qmgrName` is the name of the queue manager on which the queue resides. `queueName` is the name of the queue `[name-value pairs]` is an optional list of name-value pairs that sets the remaining Queue properties. If the name of the queue manager is omitted the queue manager to which the owning QueueConnection is connected is used.

Exceptions

- JMSException - if the queue or queue manager names are invalid.

MQQueue

```
public MQQueue(String queueManagerName, String queueName)
             throws JMSException;
```

Creates a new MQQueue object.

Parameters

- queueManagerName - the name of the queue manager
- queueName - the name of the queue

Exceptions

- JMSException - if the either name is invalid.

Methods

getQueueName

```
public String getQueueName();
```

Gets the name of this queue. Clients that depend upon the name are not portable.

Returns

- a string in the form of a URI that can be used in the creation methods to reconstruct this object. The URI is in the form:

  `queue://qmgrName/queueName [name-value pairs]`

  Where: `qmgrName` is the name of the queue manager on which the queue resides. `queueName` is the name of the queue `[name-value pairs]` is an optional list of name-value pairs that sets some Queue properties.

Exceptions

- JMSException - if JMS implementation for queue fails to return the queue name because of an internal error.

toString

```
public String toString();
```

Gets a version of the queue name.

Returns

- the provider specific identity values for this queue.

MQQueueBrowser

public class **MQQueueBrowser**
extends Object
implements QueueBrowser

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQQueueBrowser
```

A client uses an MQQueueBrowser to look at messages on a queue without removing them.

Note that the WebSphere MQ class MQQueueEnumeration is used to hold the browse cursor.

Methods

close

```
public void close() throws JMSException;
```

Closes all open queues left in enumerated objects. Because a provider can allocate some resources outside the JVM on behalf of an MQQueueBrowser, clients must close them when they are not needed. You cannot rely on garbage collection to reclaim these resources eventually, because this might not occur soon enough.

Exceptions

- JMSException - if JMS fails to close this browser because of a JMS error.

getEnumeration

```
public Enumeration getEnumeration() throws JMSException;
```

Gets an enumeration for browsing the current queue messages in the order that they are received.

Note that if the browser is created for a nonexistent queue, this is not detected until the first call to getEnumeration().

Returns

- an enumeration for browsing the messages.

Exceptions

- JMSException - if JMS fails to get the enumeration for this browser because of a JMS error.

getMessageSelector

```
public String getMessageSelector() throws JMSException;
```

Gets the queue browser's message selector expression.

Returns

- this queue browser's message selector

Exceptions

- JMSException - if JMS fails to get the message selector for this browser due to some JMS error.

getQueue

```
public Queue getQueue() throws JMSException;
```

Gets the queue associated with this queue browser.

Returns

- the queue

Exceptions

- JMSException - if JMS fails to get the queue associated with this browser due to some JMS error.

MQQueueConnection

public class **MQQueueConnection**
extends MQConnection
implements QueueConnection

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnection
           |
           +----com.ibm.mq.jms.MQQueueConnection
```

An MQueueConnection is an active connection to a JMS point-to-point provider. A client uses an MQQueueConnection to create one or more MQQueueSessions for producing and consuming messages.

Methods

close

```
public void close() throws JMSException;
```

Closes this connection and release its resources.

Exceptions

- JMSException -

createQueueSession

```
public QueueSession createQueueSession(boolean transacted,
                                       int acknowledgeMode)
             throws JMSException;
```

Creates an MQQueueSession object.

Parameters

- transacted - indicates whether the session is transacted
- acknowledgeMode - indicates whether the consumer or the client will acknowledge any messages it receives; ignored if the session is transacted. Legal values are Session.AUTO_ACKNOWLEDGE, Session.CLIENT_ACKNOWLEDGE, and Session.DUPS_OK_ACKNOWLEDGE.

Returns

- a newly created queue session.

Exceptions

- JMSException - if JMS Provider fails to create an MQQueueSession due to an internal error.

MQQueueConnectionFactory

public class **MQQueueConnectionFactory**
extends MQConnectionFactory
implements QueueConnectionFactoryReferenceableSerializable

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionFactory
           |
           +----com.ibm.mq.jms.MQQueueConnectionFactory
```

A client uses an MQQueueConnectionFactory to create QueueConnections with a JMS point-to-point provider.

Constructors

MQQueueConnectionFactory

```
public MQQueueConnectionFactory();
```

This is the default constructor.

Methods

createQueueConnection

```
public QueueConnection createQueueConnection() throws JMSException;
```

Creates a queue connection with default user identity.

Returns

- a newly created queue connection.

Exceptions

- JMSException - if JMS Provider fails to create Queue Connection due to some internal error. required resources for a Queue Connection.
- JMSSecurityException - if client authentication fails due to invalid user name or password.

createQueueConnection

```
public QueueConnection createQueueConnection(String userName,
                                             String password)
            throws JMSException;
```

Creates a queue connection with specified user identity.

Parameters

- userName - the caller's user name
- password - the caller's password

Returns

- a newly created queue connection.

Exceptions

- JMSException - if JMS Provider fails to create an MQQueueConnection due to an internal error.
- JMSSecurityException - if client authentication fails due to an invalid user name or password.

MQQueueEnumeration

public class **MQQueueEnumeration**
extends Object
implements Enumeration

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQQueueEnumeration
```

MQQueueEnumeration enumerates messages on a queue. This class is not defined in the JMS specification; it is created by calling the getEnumeration() method of MQQueueBrowser. The class contains a browse cursor. The queue is closed once the cursor has moved off the end of the queue. There is no way to reset an instance of this class; it acts as a one-shot mechanism.

Methods

hasMoreElements

```
public boolean hasMoreElements();
```

Indicates whether another message can be returned.

Returns

- **true**if another message can be returned.

nextElement

```
public Object nextElement() throws NoSuchElementException;
```

Gets the current message. Always returns a message if hasMoreElements() returns **true**. It is possible for the returned message to pass its expiry date between the hasMoreElements() and the nextElement() calls.

Returns

- the current message.

Exceptions

- NoSuchElementException -

MQQueueReceiver

public class **MQQueueReceiver**
extends MQMessageConsumer
implements QueueReceiver

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageConsumer
           |
           +----com.ibm.mq.jms.MQQueueReceiver
```

A client uses an MQQueueReceiver to receive messages that have been delivered to a queue.

Methods

close

```
public void close() throws JMSException;
```

Close the receiver. Releases underlying resources associated with this receiver.

Exceptions

- JMSException - if underlying WebSphere MQ calls fail.

getQueue

```
public Queue getQueue() throws JMSException;
```

Gets the queue associated with this queue receiver.

MQQueueSender

public class **MQQueueSender**
extends MQMessageProducer
implements QueueSender

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageProducer
           |
           +----com.ibm.mq.jms.MQQueueSender
```

A client uses an MQQueueSender to send messages to a queue. A QueueSender is normally associated with a particular queue. However, it is possible to create an unidentified QueueSender that is not associated with any given queue.

Methods

close

```
public void close() throws JMSException;
```

Close the sender. Releases underlying resources associated with this receiver.

Exceptions

- JMSException - if an underlying WebSphere MQ calls fail.

getQueue

```
public Queue getQueue() throws JMSException;
```

Gets the Queue associated with this sender.

MQQueueSession

public class **MQQueueSession**
extends MQSession
implements QueueSession

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQSession
           |
           +----com.ibm.mq.jms.MQQueueSession
```

An MQQueueSession provides methods to create MQQueueReceivers, MQQueueSenders, MQQueueBrowsers, and MQTemporaryQueues.

Methods

commit

```
public void commit() throws JMSException;
```

Commits all messages done in this transaction and releases any locks currently held.

Exceptions

- JMSException - with one of the following reasons:
  - MQJMS_E_SESSION_CLOSED
  - MQJMS_E_SESSION_NOT_TRANSACTED
  - MQJMS_EXCEPTION_MQ_NULL_QMGR
  - MQJMS_EXCEPTION_TRANSACTION_ROLLED_BACK
  - MQJMS_EXCEPTION_MQ_QM_COMMIT_FAILED

createReceiver

```
public QueueReceiver createReceiver(Queue queue) throws JMSException;
```

Creates a QueueReceiver object to receive messages from the specified queue.

Exceptions

- JMSException -

createReceiver

```
public QueueReceiver createReceiver(Queue queue, String messageSelector)
             throws JMSException;
```

Creates an MQQueueReceiver object to receive messages from the specified queue and message selector.

Exceptions

- JMSException -

createSender

```
public QueueSender createSender(Queue queue) throws JMSException;
```

Creates a QueueSender object to send messages to the specified queue.

Exceptions

- JMSException -

createTemporaryQueue

```
public TemporaryQueue createTemporaryQueue() throws JMSException;
```

Creates a JMS temporary queue. The temporary queue remains until the connection ends or the queue is explicitly deleted, whichever is the sooner.

Returns

- TemporaryQueue

Exceptions

- JMSException - IllegalStateException with one of the following reasons:
  - MQJMS_E_SESSION_CLOSED
  - MQJMS_E_NULL_CONNECTION
  - MQJMS_E_TMPQ_FAILED

recover

```
public void recover() throws JMSException;
```

Restarts message delivery from the oldest unacknowledged message. Analogous to rollback(), but for the non-transacted case.

rollback

```
public void rollback() throws JMSException;
```

Rolls back any messages done in this transaction and releases any locks currently held.

Exceptions

- JMSException - if JMS implementation fails to roll back the transaction due to some internal error.

MQSession

public class **MQSession**
extends Object
implements SessionJMSAcknowledgePointJMSDestinationFactory

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQSession
```

A JMS session is a single-threaded context for producing and consuming messages.

Methods

close

```
public void close() throws JMSException;
```

Closes the session.

Exceptions

- JMSException - with one of the following reasons:
  - MQJMS_EXCEPTION_BAD_STATE_TRANSITION
  - MQJMS_E_CLOSE_FAILED
  - MQJMS_EXCEPTION_QMDISC_FAILED

commit

```
public void commit() throws JMSException;
```

Commits all messages done in this transaction and releases any locks currently held. This always throws a JMSException when you have a direct connection to a broker.

Exceptions

- JMSException - with one of the following reasons:
  - MQJMS_E_SESSION_CLOSED
  - MQJMS_E_SESSION_NOT_TRANSACTED
  - MQJMS_EXCEPTION_MQ_NULL_QMGR
  - MQJMS_EXCEPTION_TRANSACTION_ROLLED_BACK
  - MQJMS_EXCEPTION_MQ_QM_COMMIT_FAILED

createBrowser

```
public QueueBrowser createBrowser(Queue queue) throws JMSException;
```

Creates a QueueBrowser object to peek at the messages on the specified queue.

Parameters

- queue - the queue to access.

Returns

- QueueBrowser - a newly created QueueBrowser.

Exceptions

- JMSException - with one of the following reasons:
  - MQJMS_E_SESSION_CLOSED
  - MQJMS_EXCEPTION_INVALID_DESTINATION

- MQJMS_E_NON_LOCAL_RXQ

createBrowser

```
public QueueBrowser createBrowser(Queue queue, String messageSelector)
             throws JMSException;
```

Creates a QueueBrowser object to peek at the messages on the specified queue using a message selector.

Parameters

- queue - the queue to access
- messageSelector - only messages with properties matching the message selector expression are delivered. A value of null or an empty string indicates that there is no message selector for the message consumer.

Returns

- QueueBrowser - a newly create QueueBrowser

Exceptions

- JMSException - with one of the following reasons:
  - MQJMS_E_SESSION_CLOSED
  - MQJMS_EXCEPTION_INVALID_DESTINATION
  - MQJMS_E_NON_LOCAL_RXQ

createBytesMessage

```
public BytesMessage createBytesMessage() throws JMSException;
```

Creates a BytesMessage object.

Returns

- BytesMessage

Exceptions

- JMSException - if JMS fails due to some internal JMS error.
- JMSException - with reason MQJMS_E_SESSION_CLOSED .

createConsumer

```
public MessageConsumer createConsumer(Destination destination)
             throws JMSException;
```

Creates a MessageConsumer for the specified Destination. Since Queue and Topic both inherit from Destination, they can be used in the destination parameter to create a MessageConsumer.

Parameters

- destination - the Destination to access.

Returns

- MessageConsumer

Exceptions

- JMSException - if the command fails due to some internal JMS error.

createConsumer

```
public MessageConsumer createConsumer(Destination destination,
                                      String messageSelector)
             throws JMSException;
```

Creates a MessageConsumer for the specified destination, using a message selector.

Parameters

- destination - the Destination to access.
- messageSelector - the message selector

Returns

- MessageConsumer

Exceptions

- JMSException - if the command fails due to some internal JMS error.

createConsumer

```
public MessageConsumer createConsumer(Destination destination,
                                      String messageSelector,
                                      boolean noLocal)
             throws JMSException;
```

Creates MessageConsumer for the specified Destination, using a message selector.

Parameters

- destination - the Destination to access.
- messageSelector - the message selector
- noLocal - when the destination is a topic, **true** inhibits the delivery of messages published by its own connection. The behavior for NoLocal is ignored if the destination is a queue.

Returns

- MessageConsumer

Exceptions

- JMSException - if the command fails due to some internal JMS error.

createDurableSubscriber

```
public TopicSubscriber createDurableSubscriber(Topic topic, String name)
             throws JMSException;
```

Creates a durable Subscriber to the specified topic.

Parameters

- topic - the topic to subscribe to
- name - the name used to identify this subscription.

Returns

- TopicSubscriber

Exceptions

- IllegalStateException - if the session has been closed.
- InvalidDestinationException - if the topic specified is not valid.
- JMSException - if the Session fails to create a subscriber due to an internal error.

createDurableSubscriber

```
public TopicSubscriber createDurableSubscriber(Topic topic, String name,
                                               String selector,
                                               boolean noLocal)
             throws JMSException;
```

Creates a durable Subscriber to the specified topic.

Parameters

- topic - the topic to subscribe to
- name - the name used to identify this subscription.
- selector - only messages with properties matching the message selector expression are delivered. This value may be null.
- noLocal - **true**inhibits the delivery of messages published by its own connection.

Exceptions

- IllegalStateException - if the session has been closed.
- InvalidDestinationException - if the topic specified is not valid.
- JMSException - if the Session fails to create a subscriber due to an internal error.

createMapMessage

```
public MapMessage createMapMessage() throws JMSException;
```

Creates a MapMessage. A MapMessage is used to send a self-defining set of name-value pairs where names are Strings and values are Java primitive types.

Returns

- MapMessage

Exceptions

- JMSException - with reason MQJMS_E_SESSION_CLOSED

createMessage

```
public Message createMessage() throws JMSException;
```

Creates a Message. The Message interface is the root interface of all JMS messages. It holds all the standard message header information. It can be sent when a message containing only header information is sufficient.

Returns

- Message

Exceptions

- JMSException - with reason MQJMS_E_SESSION_CLOSED .

createObjectMessage

```
public ObjectMessage createObjectMessage() throws JMSException;
```

Creates an ObjectMessage. An ObjectMessage is used to send a message that contains a serializable Java object.

Returns

- ObjectMessage

Exceptions

- IllegalStateException - with reason MQJMS_E_SESSION_CLOSED .

createObjectMessage

```
public ObjectMessage createObjectMessage(Serializable object)
             throws JMSException;
```

Creates an initialized ObjectMessage. An ObjectMessage is used to send a message that containing a serializable Java object.

Parameters

- object - the object to use to initialize this message.

Returns

- ObjectMessage

Exceptions

- IllegalStateException - with reason MQJMS_E_SESSION_CLOSED .

createProducer

```
public MessageProducer createProducer(Destination destination)
             throws JMSException;
```

Creates a MessageProducer to send messages to the specified destination.

Parameters

- destination - the Destination to send to, or null if this is a producer which does not have a specified destination.

Exceptions

- JMSException - with reason MQJMS_EXCEPTION_MQ_NULL_QMGR
- JMSException - with reason MQJMS_EXCEPTION_MQ_Q_OPEN_FAILED
- InvalidDestinationException - If the topic specified is not valid.
- JMSException - if the Session fails to create a producer because of an internal error.

createQueue

```
public Queue createQueue(String queueName) throws JMSException;
```

Creates a Queue object given a queue name.

Parameters

- queueName - the name of this Queue

Returns

- a Queue with the given name

Exceptions

- JMSException - with one of the following reasons:
  - MQJMS_E_SESSION_CLOSED.
  - MQJMS_EXCEPTION_INVALID_DESTINATION

createStreamMessage

```
public StreamMessage createStreamMessage() throws JMSException;
```

Creates a StreamMessage object. A StreamMessage is used to send a self-defining stream of Java primitives.

Returns

- StreamMessage

Exceptions

- JMSException - IllegalStateException with reason MQJMS_E_SESSION_CLOSED.

createTemporaryQueue

```
public TemporaryQueue createTemporaryQueue() throws JMSException;
```

Creates a JMS temporary queue. The temporary queue remains until the connection ends or the queue is explicitly deleted, whichever is the sooner.

Returns

- TemporaryQueue

Exceptions

- JMSException - IllegalStateException with one of the following reasons:
  - MQJMS_E_SESSION_CLOSED
  - MQJMS_E_NULL_CONNECTION
  - MQJMS_E_TMPQ_FAILED

createTemporaryTopic

```
public TemporaryTopic createTemporaryTopic() throws JMSException;
```

Creates a temporary topic. The temporary topic remains until the connection ends or the topic is explicitly deleted, whichever is the sooner.

Returns

- a temporary topic.

Exceptions

- JMSException - if the Session fails to create a temporary topic due to an internal error.
- IllegalStateException - if the Session object has been closed.

createTextMessage

```
public TextMessage createTextMessage() throws JMSException;
```

Creates a TextMessage. A TextMessage is used to send a message containing a StringBuffer.

Returns

- TextMessage

Exceptions

- JMSException - IllegalStateException with reason MQJMS_E_SESSION_CLOSED.

createTextMessage

```
public TextMessage createTextMessage(String string) throws JMSException;
```

Creates an initialized TextMessage.

Parameters

- string - the string used to initialize this message.

Returns

- TextMessage

Exceptions

- JMSException - IllegalStateException with key MQJMS_E_SESSION_CLOSED.

createTopic

```
public Topic createTopic(String topicName) throws JMSException;
```

Creates a Topic given a Topic name.

Parameters

- topicName - the name of this topic

Returns

- a Topic with the given name.

Exceptions

- JMSException - if a Session fails to create a Topic due to an internal error.
- IllegalStateException - if the Session object has been closed.

getAcknowledgeMode

```
public int getAcknowledgeMode() throws JMSException;
```

Gets the acknowledgement mode of the session. The acknowledgement mode is set at the time that the session is created. If the session is transacted, the acknowledgement mode is ignored.

Returns

- the current acknowledgement mode for the session if the session is not transacted. Otherwise returns SESSION_TRANSACTED.

getMessageListener

```
public MessageListener getMessageListener() throws JMSException;
```

Gets the session's distinguished message listener.

Returns

- MessageListener

Exceptions

- JMSException - with reason MQJMS_E_SESSION_CLOSED .

getTransacted

```
public boolean getTransacted() throws JMSException;
```

Indicates whether the session is in transacted mode. Always returns false when you have a direct connection to a broker.

Returns

- **true** if in transacted mode

Exceptions

- JMSException - with one of the following reasons:
  - MQJMS_E_SESSION_CLOSED
  - MQJMS_E_INTERNAL_ERROR

recover

```
public void recover() throws JMSException;
```

Stops message delivery in this session and restarts message delivery with the oldest unacknowledged message. This always throws a JMSException when you have a direct connection to a broker.

Exceptions

- JMSException - with one of the following reasons:
  - MQJMS_E_SESSION_CLOSED
  - MQJMS_E_SESSION_IS_TRANSACTED
  - MQJMS_EXCEPTION_MQ_NULL_QMGR
  - MQJMS_E_RECOVER_BO_FAILED

rollback

```
public void rollback() throws JMSException;
```

Rolls back any messages processed in this transaction and releases any locks currently held. This always throws a JMSException when you have a direct connection to a broker.

Exceptions

- JMSException - with one of the following reasons:
  - MQJMS_E_SESSION_CLOSED
  - MQJMS_E_SESSION_NOT_TRANSACTED
  - MQJMS_EXCEPTION_MQ_NULL_QMGR
  - MQJMS_E_ROLLBACK_FAILED

setMessageListener

```
public void setMessageListener(MessageListener listener)
             throws JMSException;
```

Sets the session's distinguished message listener.

Parameters

- listener -

Exceptions

- JMSException - with one of the following reasons:
  - MQJMS_E_SESSION_CLOSED
  - MQJMS_E_SESSION_ASYNC

unsubscribe

```
public void unsubscribe(String name) throws JMSException;
```

Unsubscribes a durable subscription that has been created by a client.

For a direct connection to WebSphere MQ Event Broker, WebSphere Business Integration Event Broker, or WebSphere Business Integration Message Broker, this method throws a JMSException.

Exceptions

- JMSException - if the Session fails to unsubscribe to the durable subscription due to an internal error.

MQTemporaryQueue

public class **MQTemporaryQueue**
extends MQQueue
implements TemporaryQueue

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQDestination
           |
           +----com.ibm.mq.jms.MQQueue
                   |
                   +----com.ibm.mq.jms.MQTemporaryQueue
```

An MQTemporaryQueue object is a unique Queue object created for the duration of a Connection.

Methods

delete

```
public void delete() throws JMSException;
```

Deletes this temporary queue.

Exceptions

- JMSException - if the queue is in use, or if the command fails due to some internal error.

MQTemporaryTopic

public class **MQTemporaryTopic**
extends MQTopic
implements TemporaryTopic

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQDestination
           |
           +----com.ibm.mq.jms.MQTopic
                   |
                   +----com.ibm.mq.jms.MQTemporaryTopic
```

An MQTemporaryTopic object is a unique Topic object created for the duration of a Connection.

Methods

delete

```
public void delete() throws JMSException;
```

Deletes this temporary topic. If there are existing subscribers still using it, then a JMSException is thrown.

Exceptions

- JMSException - if JMS implementation fails to delete a Temporary topic due to some internal error.

MQTopic

public class **MQTopic**
extends MQDestination
implements TopicReferenceableSerializableTopic

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQDestination
           |
           +----com.ibm.mq.jms.MQTopic
```

An MQTopic object encapsulates a provider-specific topic name.

Methods

getBrokerCCDurSubQueue

```
public String getBrokerCCDurSubQueue();
```

Gets the brokerCCDurSubQueue attribute

Returns

- the broker's queue name for durable connection consumers

getBrokerDurSubQueue

```
public String getBrokerDurSubQueue();
```

Gets the brokerDurSubQueue attribute

Returns

- the broker's queue name for durable subscribers

getBrokerPubQueue

```
public String getBrokerPubQueue();
```

Gets the broker's publish queue name

Returns

- the broker's publish queue name

getBrokerPubQueueManager

```
public String getBrokerPubQueueManager();
```

Gets the brokerQueueManager attribute

Returns

- the broker's publish queue manager's name

getBrokerVersion

```
public int getBrokerVersion();
```

Gets the broker version.

Returns

- the version number.

getTopicName

```
public String getTopicName();
```

Gets the name of this Topic.

Clients that depend upon the name, are not portable.

Returns

- the Topic name

isTemporary

```
public boolean isTemporary();
```

Returns

- whether the topic is a temporary topic.

setBrokerCCDurSubQueue

```
public void setBrokerCCDurSubQueue(String name) throws JMSException;
```

Sets the name of the subscriber queue for consumers, using a durable connection to the broker.

Parameters

- name - the name of the queue.

Exceptions

- JMSException - if name is either null or not valid.

setBrokerDurSubQueue

```
public void setBrokerDurSubQueue(String x) throws JMSException;
```

Sets the brokerDurSubQueue attribute

Parameters

- x - the name of the broker durable subscriber queue

Exceptions

- JMSException -

setBrokerPubQueue

```
public void setBrokerPubQueue(String brokerPubQueue) throws JMSException;
```

Set method for broker publish queue attribute

Parameters

- brokerPubQueue - the name of the broker publish queue

Exceptions

- JMSException -

setBrokerPubQueueManager

```
public void setBrokerPubQueueManager(String brokerPubQueueManager)
            throws JMSException;
```

Sets the broker's queue manager

Parameters

- brokerPubQueueManager - the name of the broker's queue manager to publish on

Exceptions

- JMSException - if the command failed due to an internal error

setBrokerVersion

```
public void setBrokerVersion(int brkver) throws JMSException;
```

Sets the broker version.

Parameters

- brkver - the version number. Valid numbers are:
  - JMSC.MQJMS_BROKER_V1
  - JMSC.MQJMS_BROKER_V2

Exceptions

- JMSException - if brkver is neither of the above.

toString

```
public String toString();
```

Returns a string representation of the Topic object.

Returns

- the specific identity values for this Topic.

MQTopicConnection

public class **MQTopicConnection**
extends MQConnection
implements TopicConnection

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnection
           |
           +----com.ibm.mq.jms.MQTopicConnection
```

An MQTopicConnection object is an active connection to a publish/subscribe JMS provider.

Methods

createTopicSession

```
public TopicSession createTopicSession(boolean transacted,
                                       int acknowledgeMode)
             throws JMSException;
```

Creates a TopicSession object.

Parameters

- transacted - if true, throws a JMSException on a direct connection to a broker.
- acknowledgeMode - indicates whether the consumer or the client will acknowledge any messages it receives; ignored if the session is transacted. Legal values are Session.AUTO_ACKNOWLEDGE, Session.CLIENT_ACKNOWLEDGE, and Session.DUPS_OK_ACKNOWLEDGE.

Returns

- session - a newly created topic session

Exceptions

- JMSException - if the MQTopicConnection failed to create a session due to an internal error or if a transacted session was requested when using a direct connection to a broker.

MQTopicConnectionFactory

public class **MQTopicConnectionFactory**
extends MQConnectionFactory
implements TopicConnectionFactoryReferenceableSerializable

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionFactory
           |
           +----com.ibm.mq.jms.MQTopicConnectionFactory
```

A client uses an MQTopicConnectionFactory object to create TopicConnection objects with a publish/subscribe JMS provider.

Constructors

MQTopicConnectionFactory

```
public MQTopicConnectionFactory();
```

Creates an instance of a TopicConnectionFactory

Methods

createTopicConnection

```
public TopicConnection createTopicConnection() throws JMSException;
```

Creates a topic connection with default user identity.

Returns

- a newly created queue connection.

Exceptions

- JMSException - if JMS Provider fails to create Topic Connection due to some internal error. required resources for a Topic Connection.
- JMSSecurityException - if client authentication fails due to invalid user name or password.

createTopicConnection

```
public TopicConnection createTopicConnection(String userName,
                                             String password)
             throws JMSException;
```

Creates a topic connection with specified user identity.

Parameters

- userName - the caller's user name
- password - the caller's password

Returns

- a newly created topic connection.

Exceptions

- JMSException - if JMS Provider fails to create Topic Connection due to some internal error.
- JMSSecurityException - if client authentication fails due to invalid user name or password.

MQTopicPublisher

public class **MQTopicPublisher**
extends MQMessageProducer
implements TopicPublisher

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageProducer
           |
           +----com.ibm.mq.jms.MQTopicPublisher
```

A client uses an MQTopicPublisher object to publish messages on a topic. An MQTopicPublisher object is the publish subscribe form of a message producer.

Methods

getTopic

```
public Topic getTopic() throws JMSException;
```

Gets the topic associated with this publisher.

Returns

- this publisher's topic

Exceptions

- JMSException - if JMS fails to get topic for this topic subscriber due to some internal error.

publish

```
public void publish(Message message) throws JMSException;
```

Publishes a message to the topic. Uses the TopicPublisher's default delivery mode, priority, and time to live.

Parameters

- message - the message to publish

Exceptions

- JMSException - if producer fails to publish the message due to an internal error.

publish

```
public void publish(Message message, int deliveryMode, int priority,
                    long timeToLive) throws JMSException;
```

Publishes a message to the topic, specifying delivery mode, priority, and time to live.

Parameters

- message - the message to publish
- deliveryMode - the delivery mode to use
- priority - the priority for this message
- timeToLive - the message's lifetime (in milliseconds)

Exceptions

- JMSException - if producer fails to publish the message due to an internal error.

publish

```
public void publish(Topic topic, Message message) throws JMSException;
```

Publishes a message to a topic for an unidentified message producer. Uses the TopicPublisher's default delivery mode, priority, and time to live.

Parameters

- topic - the topic to publish this message to
- message - the message to publish

Exceptions

- JMSException - if producer fails to publish the message due to an internal error.

publish

```
public void publish(Topic topic, Message message, int deliveryMode,
                    int priority, long timeToLive)
             throws JMSException;
```

Publishes a message to a topic for an unidentified message producer, specifying delivery mode, priority and time to live.

Parameters

- topic - the topic to publish this message to
- message - the message to publish
- deliveryMode - the delivery mode to use
- priority - the priority for this message
- timeToLive - the message's lifetime (in milliseconds)

Exceptions

- JMSException - if producer fails to publish the message due to an internal error.

MQTopicSession

public class **MQTopicSession**
extends MQSession
implements TopicSession

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQSession
           |
           +----com.ibm.mq.jms.MQTopicSession
```

An MQTopicSession object provides methods for creating MQTopicPublisher, MQTopicSubscriber, and MQTemporaryTopic objects.

Methods

createPublisher

```
public TopicPublisher createPublisher(Topic topic) throws JMSException;
```

Creates a publisher for the specified topic.

Parameters

- topic - the Topic on which messages are to be published.

Exceptions

- JMSException - if a Session fails to create a publisher due to an internal error.

createSubscriber

```
public TopicSubscriber createSubscriber(Topic topic) throws JMSException;
```

Creates a nondurable Subscriber to the specified topic.

Parameters

- topic - the topic to subscribe to

Exceptions

- JMSException - if a session fails to create a subscriber due to some JMS error.
- InvalidDestinationException - if invalid Topic specified.

createSubscriber

```
public TopicSubscriber createSubscriber(Topic topic, String selector,
                                        boolean noLocal)
             throws JMSException;
```

Creates a nondurable Subscriber to the specified topic.

Parameters

- topic - the topic to subscribe to
- selector - only messages with properties matching the message selector expression are delivered. This value may be null.
- noLocal - if set, inhibits the delivery of messages published by its own connection.

Exceptions

- JMSException - if a session fails to create a subscriber due to some JMS error or invalid selector.
- InvalidDestinationException - if invalid Topic specified.
- InvalidSelectorException - if the message selector is invalid.

createTemporaryTopic

```
public TemporaryTopic createTemporaryTopic() throws JMSException;
```

Creates a TemporaryTopic object. Its lifetime will be that of the MQTopicConnection unless it is deleted earlier.

Returns

- a temporary topic

Exceptions

- JMSException - if the session fails to create a temporary topic due to some internal error.

createTopic

```
public Topic createTopic(String topicName) throws JMSException;
```

Creates a topic identity given a Topic name.

Parameters

- topicName - the name of this Topic

Returns

- a Topic with the given name

Exceptions

- JMSException - if the session fails to create a topic due to some internal error.

MQTopicSubscriber

public class **MQTopicSubscriber**
extends MQMessageConsumer
implements TopicSubscriber

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQMessageConsumer
           |
           +----com.ibm.mq.jms.MQTopicSubscriber
```

A client uses an MQTopicSubscriber object to receive messages that have been published to a topic.

Methods

close

```
public void close() throws JMSException;
```

Closes the subscriber and releases underlying resources associated with this subscriber.

Exceptions

- JMSException - if the underlying MQ calls fail.

getTopic

```
public Topic getTopic() throws JMSException;
```

Gets the topic associated with this subscriber.

Returns

- this subscriber's topic

Exceptions

- JMSException - if JMS fails to get topic for this topic subscriber due to some internal error.

MQXAConnection

public class **MQXAConnection**
extends MQConnection
implements XAConnection

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnection
           |
           +----com.ibm.mq.jms.MQXAConnection
```

The XAConnection interface extends the capability of Connection by providing an XASession.

Methods

createXASession

```
public XASession createXASession() throws JMSException;
```

Creates an XASession.

Returns

- the XASession

Exceptions

- JMSException - if JMS Connection fails to create an XA topic session due to some internal error.

MQXAConnectionFactory

public class **MQXAConnectionFactory**
extends MQConnectionFactory
implements XAConnectionFactory

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionFactory
           |
           +----com.ibm.mq.jms.MQXAConnectionFactory
```

The MQXAConnectionFactory interface is an implementation of XAConnectionFactory interface.

WebSphere MQ JMS exposes its JTS support in the XAConnectionFactory , XAConnection, and XASession classes. These classes are provided for use in a J2EE application server environment.

WebSphere Application Server Version 5 uses these classes to create and manage a pool of XAConnection and XASession objects. A JMS application does not need to use these classes directly if it is running in this environment.

A JMS application might need to use the XAConnectionFactory class if it is running in a WebSphere Application Server environment with a version of WebSphere Application Server before Version 5.

Constructors

MQXAConnectionFactory

```
public MQXAConnectionFactory();
```

Default constructor.

Methods

createXAConnection

```
public XAConnection createXAConnection() throws JMSException;
```

Creates an XA connection with the default user identity. The connection is created in stopped mode. No messages are delivered until the XAConnection.start method is called explicitly.

Returns

- a newly created XA connection.

Exceptions

- JMSException - if JMS Provider fails to create XAConnection due to some internal error.
- JMSSecurityException - if client authentication fails due to invalid user name or password.

createXAConnection

```
public XAConnection createXAConnection(String userName, String password)
             throws JMSException;
```

Creates an XA connection with the specified user identity. The connection is created in stopped mode. No messages are delivered until the XAConnection.start() method is called explicitly.

Parameters

- userName - the user name of the caller.
- password - the password of the caller.

Returns

- a newly created XA connection.

Exceptions

- JMSException - if JMS fails to create an XA connection because of an internal JMS error.
- JMSSecurityException - if client authentication fails because the user name or password is not valid.

MQXAQueueConnection

public class **MQXAQueueConnection**
extends MQQueueConnection
implements XAQueueConnection

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnection
           |
           +----com.ibm.mq.jms.MQQueueConnection
                   |
                   +----com.ibm.mq.jms.MQXAQueueConnection
```

XAQueueConnection provides the same create options as MQQueueConnection. The only difference is that, by definition, an XAConnection is transacted.

Methods

createXAQueueSession

```
public XAQueueSession createXAQueueSession() throws JMSException;
```

Creates an XAQueueSession.

Returns

- the XAQueueSession.

Exceptions

- JMSException - if JMS Connection fails to create a XA Queue session due to some internal error.

MQXAQueueConnectionFactory

public class **MQXAQueueConnectionFactory**
extends MQQueueConnectionFactory
implements XAQueueConnectionFactory

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionFactory
           |
           +----com.ibm.mq.jms.MQQueueConnectionFactory
                   |
                   +----com.ibm.mq.jms.MQXAQueueConnectionFactory
```

An XAQueueConnectionFactory provides the same create options as an MQQueueConnectionFactory.

Constructors

MQXAQueueConnectionFactory

```
public MQXAQueueConnectionFactory();
```

Default constructor.

Methods

createXAQueueConnection

```
public XAQueueConnection createXAQueueConnection() throws JMSException;
```

Creates an XA Queue connection with default user identity.

Returns

- a newly created XA Queue connection.

Exceptions

- JMSException - if JMS Provider fails to create an XA queue connection due to an internal error.
- JMSSecurityException - if client authentication fails due to invalid user name or password.

createXAQueueConnection

```
public XAQueueConnection createXAQueueConnection(String userName,
                                                 String password)
            throws JMSException;
```

Creates an XA Queue connection with specified user identity.

Parameters

- userName - the caller's user name
- password - the caller's password

Returns

- a newly created XA Queue connection.

Exceptions

- JMSException - if JMS Provider fails to create an XA queue connection due to an internal error.

- JMSSecurityException - if client authentication fails due to invalid user name or password.

MQXAQueueSession

public class **MQXAQueueSession**
extends MQXASession
implements XAQueueSession

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQSession
           |
           +----com.ibm.mq.jms.MQXASession
                   |
                   +----com.ibm.mq.jms.MQXAQueueSession
```

An XAQueueSession provides an MQQueueSession which can be used to create MQQueueReceivers, MQQueueSenders and MQQueueBrowsers.

Constructors

MQXAQueueSession

```
public MQXAQueueSession(MQConnection connection, MQQueueSession session,
                        MQXAResource resource)
             throws JMSException;
```

Constructor which extends an MQXASession object.

Parameters

- connection - the connection.
- session - the input session.
- resource - the XA resources.

Exceptions

- JMSException - if the constructor fails.

Methods

getQueueSession

```
public QueueSession getQueueSession() throws JMSException;
```

Gets the Queue session associated with this XAQueueSession.

Returns

- the Queue session object.

Exceptions

- JMSException - never.

MQXASession

public class **MQXASession**
extends MQSession
implements XASession

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQSession
           |
           +----com.ibm.mq.jms.MQXASession
```

WebSphere MQ JMS exposes its JTS support in the XAConnectionFactory , XAConnection, and XASession classes. These classes are provided for use in a J2EE application server environment.

WebSphere Application Server Version 5 uses these classes to create and manage a pool of XAConnection and XASession objects. A JMS application does not need to use these classes directly if it is running in this environment.

A JMS application might need to use the XASession class if it is running in a WebSphere Application Server environment with a version of WebSphere Application Server before Version 5.

Methods

close

```
public void close() throws JMSException;
```

Closes the session.

Exceptions

- JMSException - if the command failed due to an internal error.

commit

```
public void commit() throws JMSException;
```

Not to be called in this context.

Exceptions

- TransactionInProgressException - always.

getSession

```
public Session getSession() throws JMSException;
```

Gets the session associated with this XASession.

Returns

- the session object.

Exceptions

- JMSException - never

getTransacted

```
public boolean getTransacted() throws JMSException;
```

Indicates that XA sessions are always transacted.

Returns

- **true**; always in transacted mode.

Exceptions

- JMSException, - for reasons of inheritance.

getXAResource

```
public XAResource getXAResource();
```

Gets the XA resource.

Returns

- the XAResource

recover

```
public void recover() throws JMSException;
```

Not to be called in this context.

Exceptions

- IllegalStateException - always.

rollback

```
public void rollback() throws JMSException;
```

Not to be called in this context.

Exceptions

- TransactionInProgressException - always.

MQXATopicConnection

public class **MQXATopicConnection**
extends MQTopicConnection
implements XATopicConnection

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnection
           |
           +----com.ibm.mq.jms.MQTopicConnection
                   |
                   +----com.ibm.mq.jms.MQXATopicConnection
```

An XATopicConnection provides the same creation options as MQTopicConnection. The only difference is that an XAConnection is transacted.

Methods

createXATopicSession

```
public XATopicSession createXATopicSession() throws JMSException;
```

Creates an XATopicSession.

Returns

- the XATopicSession.

Exceptions

- JMSException - if JMS Connection fails to create a XA topic session due to some internal error.

MQXATopicConnectionFactory

public class **MQXATopicConnectionFactory**
extends MQTopicConnectionFactory
implements XATopicConnectionFactory

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQConnectionFactory
           |
           +----com.ibm.mq.jms.MQTopicConnectionFactory
                   |
                   +----com.ibm.mq.jms.MQXATopicConnectionFactory
```

An MQXATopicConnectionFactory provides the same creation options as MQTopicConnectionFactory.

Constructors

MQXATopicConnectionFactory

```
public MQXATopicConnectionFactory();
```

Default constructor.

Methods

createXATopicConnection

```
public XATopicConnection createXATopicConnection() throws JMSException;
```

Creates an XA topic connection with the default user identity.

Returns

- a newly created XA topic connection.

Exceptions

- JMSException - if JMS Provider fails to create XA topic connection due to some internal error.
- JMSSecurityException - if client authentication fails due to invalid user name or password.

createXATopicConnection

```
public XATopicConnection createXATopicConnection(String userName,
                                                 String password)
            throws JMSException;
```

Creates an XA topic connection using the specified user identity. The connection is created in stopped mode. No messages are delivered until the Connection.start() method is called explicitly.

Parameters

- userName - the user name of the caller
- password - the password of the caller

Returns

- A newly-created XA topic connection.

Exceptions

- JMSException - if the JMS provider fails to create an XA topic connection because of an internal error.
- JMSSecurityException - if client authentication fails because a user name or password is not valid.

MQXATopicSession

public class **MQXATopicSession**
extends MQXASession
implements XATopicSession

```
java.lang.Object
   |
   +----com.ibm.mq.jms.MQSession
           |
           +----com.ibm.mq.jms.MQXASession
                   |
                   +----com.ibm.mq.jms.MQXATopicSession
```

An MQXATopicSession provides an MQTopicSession, which you can use to create MQTopicSubscribers and MQTopicPublishers.

The XAResource that corresponds to the TopicSession can be obtained by calling the getXAResource() method, which is inherited from XASession.

Methods

getTopicSession

```
public TopicSession getTopicSession() throws JMSException;
```

Gets the topic session associated with this XATopicSession.

Returns

- the topic session object.

Exceptions

- JMSException - never

JMSC

public interface **JMSC**

`com.ibm.mq.jms.JMSC`

Contains all constants used in the WebSphere MQ Java Message Service product.

Fields

CC_DEF_D_SHARED_QUEUE

`public final static java.lang.String`

Default name for the JMS ConnectionConsumer durable subscriber queue. This can be altered using MQTopic.setBrokerCCDurSubQueue(String) .

CC_DEF_ND_SHARED_QUEUE

`public final static java.lang.String`

Default name for the JMS ConnectionConsumer nondurable subscriber queue. This can be altered using MQTopic.setBrokerCCSubQueue(String) .

MAP_NAME_STYLE_COMPATIBLE

`public final static boolean`

This parameter can be passed to MQConnectionFactory.setMapNameStyle(boolean) to indicate that the legacy com.ibm.jms.JMSMapMessage element naming format will be used. This is only needed if Map messages are being sent to WebSphere MQ JMS Clients older than version 5.3.

MAP_NAME_STYLE_STANDARD

`public final static boolean`

This parameter may be passed to MQConnectionFactory.setMapNameStyle(boolean) to indicate that the standard com.ibm.jms.JMSMapMessage element naming format will be used. This is the default.

MQCNO_FASTPATH_BINDING

`public final static int`

This parameter can be passed to MQConnectionFactory.setMQConnectionOptions(int) to indicate that the application and the local queue manager agent must be part of the same unit of execution.

MQCNO_ISOLATED_BINDING

`public final static int`

This parameter can be passed to MQConnectionFactory.setMQConnectionOptions(int) to indicate that the application and the local queue manager agent must run in separate units of execution and no resources will be shared.

MQCNO_RESTRICT_CONN_TAG_Q_MGR

`public final static int`

This parameter can be passed to MQConnectionFactory.setMQConnectionOptions(int) to indicate that connection tag use is restricted within the queue manager. This connection option is supported on z/OS only.

MQCNO_RESTRICT_CONN_TAG_QSG

```
public final static int
```

This parameter can be passed to MQConnectionFactory.setMQConnectionOptions(int) to indicate that connection tag use is restricted within the queue-sharing group. This connection option is supported on z/OS only.

MQCNO_SERIALIZE_CONN_TAG_Q_MGR

```
public final static int
```

This parameter can be passed to MQConnectionFactory.setMQConnectionOptions(int) to indicate that connection tag use is serialized within the queue manager. This connection option is supported on z/OS only.

MQCNO_SERIALIZE_CONN_TAG_QSG

```
public final static int
```

This parameter can be passed to MQConnectionFactory.setMQConnectionOptions(int) to indicate that connection tag use is serialized within the queue-sharing group. This connection option is supported on z/OS only.

MQCNO_SHARED_BINDING

```
public final static int
```

This parameter can be passed to MQConnectionFactory.setMQConnectionOptions(int) to indicate that the application and the local queue manager agent must run in separate units of execution, with some resources shared between the application and the local queue manager agent.

MQCNO_STANDARD_BINDING

```
public final static int
```

This parameter can be passed to MQConnectionFactory.setMQConnectionOptions(int) to indicate that the application and the local queue manager agent must run in separate units of execution.

MQJMS_BROKER_V1

```
public final static int
```

This parameter can be passed to MQConnectionFactory.setBrokerVersion(int) to indicate that the broker will use RFH1 headers. This is required when using the WebSphere MQ Publish/Subscribe broker or when using a broker of WebSphere MQ Integrator, WebSphere MQ Event Broker, WebSphere Business Integration Event Broker or WebSphere Business Integration Message Broker in compatibility mode.

MQJMS_BROKER_V2

```
public final static int
```

This parameter can be passed to MQConnection.setBrokerVersion(int) to indicate that the broker will use RFH2 headers. This is available when using a broker of WebSphere MQ Integrator, WebSphere MQ Event Broker, WebSphere Business Integration Event Broker or WebSphere Business Integration Message Broker in native mode.

MQJMS_CLEANUP_AS_PROPERTY

```
public final static int
```

This parameter can be passed to MQConnectionFactory.setCleanupLevel(int) to indicate that the style of cleanup to use is determined by the system property `com.ibm.mq.jms.cleanup`, which is queried at JVM startup. This property can be set on the Java command line using the -D option, to **NONE**, **SAFE**, or **STRONG**. Any other value will cause an exception to be thrown. If not set, the property defaults to **SAFE**. This allows easy JVM-wide changes to the cleanup level without updating every TopicConnectionFactory used by the system.

MQJMS_CLEANUP_FORCE

```
public final static int
```

This parameter can be passed to MQConnectionFactory.setCleanupLevel(int) to use forced cleanup. This option behaves like JMSC.MQJMS_CLEANUP_STRONG except that, instead of leaving messages that cannot be processed on SYSTEM.JMS.REPORT.QUEUE , all messages are removed even if an error is encountered during processing.

MQJMS_CLEANUP_NONDUR

```
public final static int
```

This parameter can be passed to MQConnectionFactory.setCleanupLevel(int) to use nondurable cleanup. This option behaves like JMSC.MQJMS_CLEANUP_FORCE. After clearing the SYSTEM.JMS.REPORT.QUEUE, it attempts to remove any remaining unconsumed messages sent to nondurable subscribers. If the queue manager's command server is running on any queue beginning SYSTEM.JMS.ND.* , messages are cleared and the queue itself might be deleted. Otherwise, only SYSTEM.JMS.ND.SUBSCRIBER.QUEUE and SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE are cleared of messages.

MQJMS_CLEANUP_NONE

```
public final static int
```

This parameter can be passed to MQConnectionFactory.setCleanupLevel(int) to disable cleanup. In this mode, no cleanup is performed, and no cleanup Publish/subscribe thread exists. Additionally, if the application is using the single queue approach, unconsumed messages can be left on the queue. This option can be useful if the application is distant from the queue manager, and especially if it only publishes rather than subscribes.

MQJMS_CLEANUP_SAFE

```
public final static int
```

This parameter can be passed to MQConnectionFactory.setCleanupLevel(int) to use safe cleanup. The cleanup thread tries to remove unconsumed subscription

messages or temporary queues for failed subscriptions. This mode of cleanup does not interfere with the operation of other JMS applications.

MQJMS_CLEANUP_STRONG

```
public final static int
```

This parameter can be passed to MQConnectionFactory.setCleanupLevel(int) to use strong cleanup. The cleanup thread performs as JMSC.MQJMS_CLEANUP_SAFE, but also clears the SYSTEM.JMS.REPORT.QUEUE of any unrecognized messages. This mode of cleanup can interfere with the operation of JMS applications running with later versions of WebSphere MQ JMS. If multiple JMS applications are using the same queue manager, but using different versions of WebSphere MQ JMS, only clients using the most recent version of WebSphere MQ JMS should use this option.

MQJMS_CLIENT_JMS_COMPLIANT

```
public final static int
```

This parameter can be passed to MQDestination.setTargetClient(int) to indicate that messages will be sent to a client running the WebSphere MQ JMS client. This is the default value.

MQJMS_CLIENT_NONJMS_MQ

```
public final static int
```

This parameter can be passed to MQDestination.setTargetClient(int) to indicate that messages will be sent to a non-JMS WebSphere MQ client.

MQJMS_COMPHDR_NONE

```
public final static int
```

This parameter can be passed in a Vector to MQConnectionFactory.setHdrCompList(Collection) to indicate that no message header compression will be used. This is the default value.

MQJMS_COMPHDR_SYSTEM

```
public final static int
```

This parameter can be passed in a Vector to MQConnectionFactory.setHdrCompList(Collection) to indicate that RLE message header compression will be used.

MQJMS_COMPMSG_NONE

```
public final static int
```

This parameter can be passed in a Vector to MQConnectionFactory.setMsgCompList(Collection) to indicate that no message data compression will be used. This is the default value.

MQJMS_COMPMSG_RLE

```
public final static int
```

This parameter can be passed in a Vector to MQConnectionFactory.setMsgCompList(Collection) to indicate that message data compression is to be performed using run-length encoding (RLE) compression.

MQJMS_COMPMSG_ZLIBFAST

```
public final static int
```

This parameter can be passed in a Vector to MQConnectionFactory.setMsgCompList(Collection) to indicate that message data compression is to be performed using ZLIB encoding and with speed of compression taking priority over degree of compression.

MQJMS_COMPMSG_ZLIBHIGH

```
public final static int
```

This parameter can be passed in a Vector to MQConnectionFactory.setMsgCompList(Collection) to indicate that message data compression is to be performed using ZLIB encoding and with degree of compression taking priority over speed of compression.

MQJMS_DIRECTAUTH_BASIC

```
public final static int
```

This parameter can be passed to MQConnectionFactory.setDirectAuth(int) to indicate either no authentication or basic user name and password authentication will be used.

MQJMS_DIRECTAUTH_CERTIFICATE

```
public final static int
```

This parameter can be passed to MQConnectionFactory.setDirectAuth(int) to indicate public key certificate authentication will be used.

MQJMS_EXP_APP

```
public final static int
```

This parameter can be passed to MQDestination.setExpiry(long) to indicate that the message expiry time is set to the value supplied by the application. This is the default behavior.

MQJMS_EXP_UNLIMITED

```
public final static int
```

This parameter can be passed to MQDestination.setExpiry(long) to indicate that the message expiry time is set to unlimited.

MQJMS_FIQ_NO

```
public final static int
```

This parameter can be passed to MQConnectionFactory.setFailIfQuiesce(int) to indicate applications accessing a quiescing queue manager will not fail.

MQJMS_FIQ_YES

```
public final static int
```

This parameter can be passed to MQConnectionFactory.setFailIfQuiesce(int) to indicate applications accessing a quiescing queue manager will fail. This is the default value.

MQJMS_MRET_NO

```
public final static int
```

This parameter can be passed to MQConnectionFactory.setMessageRetention(int) to indicate that unwanted messages will be dealt with according to their disposition options.

MQJMS_MRET_YES

```
public final static int
```

This parameter can be passed to MQConnectionFactory.setMessageRetention(int) to indicate that unwanted messages will remain on the input queue. This is the default value for message retention.

MQJMS_MSEL_BROKER

```
public final static int
```

This parameter can be passed to MQConnectionFactory.setMessageSelection(int) to indicate that the broker will perform message selection.

MQJMS_MSEL_CLIENT

```
public final static int
```

This parameter can be passed to MQConnectionFactory.setMessageSelection(int) to indicate that the client will perform message selection. This is the default value for message selection.

MQJMS_MULTICAST_AS_CF

```
public final static int
```

This parameter can be passed to MQConnectionFactory.setMulticast(int) to indicate that multicast usage on the Topic will be determined by the setting on MQConnectionFactory.

MQJMS_MULTICAST_DISABLED

```
public final static int
```

This parameter can be passed to MQConnectionFactory.setMulticast(int) to disable multicast.

MQJMS_MULTICAST_ENABLED

```
public final static int
```

This parameter can be passed to MQConnectionFactory.setMulticast(int) to enable multicast, if available.

MQJMS_MULTICAST_NOT_RELIABLE

```
public final static int
```

This parameter can be passed to MQConnectionFactory.setMulticast(int) to enable multicast but not to use reliable delivery. This value is used to enable multicast for legacy applications.

MQJMS_MULTICAST_RELIABLE

```
public final static int
```

This parameter can be passed to MQConnectionFactory.setMulticast(int) to enable multicast with reliable delivery only.

MQJMS_PER_APP

```
public final static int
```

This parameter can be passed to MQDestination.setPersistence(int) to indicate that the persistence is set to the value supplied by the application. This is the default behavior.

MQJMS_PER_NON

```
public final static int
```

This parameter can be passed to MQDestination.setPersistence(int) to indicate that the persistence for all messages is set to javax.jms.DeliveryMode.NON_PERSISTENT.

MQJMS_PER_NPHIGH

```
public final static int
```

This parameter can be passed to MQDestination.setPersistence(int) to indicate that messages will use NPMCLASS(HIGH) persistence if it is available on the queue. If it is not available, messages will be sent using javax.jms.DeliveryMode.PERSISTENT persistence. See *WebSphere MQ System Administration Guide* for more information about NPMCLASS(HIGH).

MQJMS_PER_PER

```
public final static int
```

This parameter can be passed to MQDestination.setPersistence(int) to indicate that the persistence for all messages is set to javax.jms.DeliveryMode.PERSISTENT.

MQJMS_PER_QDEF

```
public final static int
```

This parameter can be passed to MQDestination.setPersistence(int) to indicate that the persistence is taken from the queue definition.

MQJMS_PRI_APP

```
public final static int
```

This parameter can be passed to MQDestination.setPriority(int) to indicate that the priority is set to the value supplied by the application. This is the default behavior.

MQJMS_PRI_QDEF

```
public final static int
```

This parameter can be passed to MQDestination.setPriority(int) to indicate that the priority is taken from the queue definition.

MQJMS_PROCESSING_SHORT

```
public final static int
```

This parameter can be passed to MQConnectionFactory.setProcessDuration(int) to indicate that the processing of a MessageConsumers messages is guaranteed to be completed promptly. This value must be used if using the

MQJMS_RCVISOL_UNCOMMITTED paramter with the MQConnectionFactory.setReceiveIsolation(int) method.

MQJMS_PROCESSING_UNKNOWN

```
public final static int
```

This parameter can be passed to MQConnectionFactory.setProcessDuration(int) to indicate that the processing of a MessageConsumers messages will take an unknown amount of time.

MQJMS_RCVISOL_COMMITTED

```
public final static int
```

This parameter can be passed to MQConnectionFactory.setReceiveIsolation(int) to indicate that publish/subscribe MessageConsumers can only attempt to receive messages that have been committed by their publisher.

MQJMS_RCVISOL_DEFAULT

```
public final static int
```

The default value for the receive isolation property - currently maps to MQJMS_RCVISOL_COMMITTED

MQJMS_RCVISOL_UNCOMMITTED

```
public final static int
```

This parameter can be passed to MQConnectionFactory.setReceiveIsolation(int) to indicate that publish/subscribe MessageConsumers are willing to see messages that have not yet been committed by their publisher. Using this value might cause a subscribing application's commit() method or acknowledge() call to fail if a message's original publish is backed out.

MQJMS_SUBSTORE_BROKER

```
public final static int
```

This parameter can be passed to MQConnectionFactory.setSubscriptionStore(int) to use the Broker subscription store to hold details of subscriptions.

MQJMS_SUBSTORE_MIGRATE

```
public final static int
```

This parameter can be passed to MQConnectionFactory.setSubscriptionStore(int) to use the Migrate subscription to hold details of subscriptions.

MQJMS_SUBSTORE_QUEUE

```
public final static int
```

This parameter can be passed to MQConnectionFactory.setSubscriptionStore(int) to use the Queue subscription store to hold details of subscriptions.

MQJMS_TP_BINDINGS_MQ

```
public final static int
```

This parameter can be passed to MQConnectionFactory.setTransportType(int) to indicate that the application will connect to the queue manager in bindings mode.

MQJMS_TP_CLIENT_MQ_TCPIP

```
public final static int
```

This parameter can be passed to MQConnectionFactory.setTransportType(int) to indicate that the application will connect to the queue manager in (client TCP/IP) mode.

MQJMS_TP_DIRECT_HTTP

```
public final static int
```

This parameter can be passed to MQConnectionFactory.setTransportType(int) to indicate that the application will connect to a broker in DirectHTTP mode. See the broker documentation for more details of this connection mode.

MQJMS_TP_DIRECT_TCPIP

```
public final static int
```

This parameter can be passed to MQConnectionFactory.setTransportType(int) to indicate that the application will connect to a broker in DirectIP mode. See the broker documentation for more details of this connection mode.

MQRCCF_ALREADY_JOINED

```
public final static int
```

This reason code is returned by a broker to indicate that the identity already has an entry for this subscription. A Join registration option was specified but the subscriber identity was already a member of the subscription's identity set. See *WebSphere MQ Publish/Subscribe User's Guide* for more information on this error code.

MQRCCF_DUPLICATE_IDENTITY

```
public final static int
```

This reason code is returned by a broker to indicate that the publisher or subscriber identity is already assigned to another user ID. A given identity can only be assigned to one user ID at a time. See *WebSphere MQ Publish/Subscribe User's Guide* for more information on this error code.

MQRCCF_DUPLICATE_SUBSCRIPTION

```
public final static int
```

This reason code is returned by a broker to indicate that a matching subscription already exists. See *WebSphere MQ Publish/Subscribe User's Guide* for more information on this error code.

MQRCCF_NOT_AUTHORIZED

```
public final static int
```

This reason code is returned by a broker to indicate that the subscriber has insufficient authority. To receive publications a subscriber application needs both browse authority for the stream queue that it is subscribing to, and put authority for the queue that publications are to be sent to. Subscriptions are rejected if the subscriber does not have both authorities. See *WebSphere MQ Publish/Subscribe User's Guide* for more information on this error code.

MQRCCF_NOT_REGISTERED

```
public final static int
```

This reason code is returned by a broker to indicate that the subscriber or publisher is not registered. See *WebSphere MQ Publish/Subscribe User's Guide* for more information on this error code.

MQRCCF_REG_OPTIONS_ERROR

```
public final static int
```

This reason code is returned by a broker to indicate that incorrect registration options have been supplied. See *WebSphere MQ Publish/Subscribe User's Guide* for more information on this error code.

MQRCCF_SUB_IDENTITY_ERROR

```
public final static int
```

This reason code is returned by a broker to indicate that the subscription identity parameter is in error. Either the supplied value exceeds the maximum length allowed or the subscription identity is not currently a member of the subscription's identity set and a Join registration option was not specified. See *WebSphere MQ Publish/Subscribe User's Guide* for more information on this error code.

MQRCCF_SUB_NAME_ERROR

```
public final static int
```

This reason code is returned by a broker to indicate that the subscription name parameter is in the wrong. Either the subscription name is of an invalid format or a matching subscription already exists with no subscription name. See *WebSphere MQ Publish/Subscribe User's Guide* for more information on this error code.

MQRCCF_SUBSCRIPTION_IN_USE

```
public final static int
```

This reason code is returned by a broker to indicate that the subscription is in use. An attempt to modify or deregister a subscription was attempted by a member of the identity set when they were not the only member of this set. See *WebSphere MQ Publish/Subscribe User's Guide* for more information on this error code.

MQRCCF_SUBSCRIPTION_LOCKED

```
public final static int
```

This reason code is returned by a broker to indicate that the subscription is currently exclusively locked by another identity. See *WebSphere MQ Publish/Subscribe User's Guide* for more information on this error code.

MQRCCF_TOPIC_ERROR

```
public final static int
```

This reason code is returned by a broker to indicate that the topic name has an invalid length or contains invalid characters. Note that wildcard topic names are not allowed for Register Publisher and Publish commands. See *WebSphere MQ Publish/Subscribe User's Guide* for more information on this error code.

PS_CONTROL_QUEUE

```
public final static java.lang.String
```

Default name for the broker control queue. This can be changed using MQConnectionFactory.setBrokerControlQueue(String).

PS_DEF_D_SHARED_QUEUE

```
public final static java.lang.String
```

Default name for the JMS durable subscriber queue. This can be altered using MTopic.setBrokerDurSubQueue(String).

PS_DEF_ND_SHARED_QUEUE

```
public final static java.lang.String
```

Default name for the JMS nondurable subscriber queue. This can be altered using MQConnectionFactory.steBrokerSubQueue(String).

PS_DEFAULT_STREAM_QUEUE

```
public final static java.lang.String
```

Default name for the broker publication queue. This can be changed using MQConnectionFactory.setBrokerPubQueue(String).

BrokerCommandFailedException

public class **BrokerCommandFailedException**
extends JMSException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----javax.jms.JMSException
                           |
                           +----com.ibm.mq.jms.BrokerCommandFailedException
```

The broker has sent a response message which indicates that the command failed.

Methods

getReason

```
public int getReason();
```

Gets the exception reason code.

Returns

- the reason code

FieldNameException

public class **FieldNameException**
extends Exception

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----com.ibm.mq.jms.FieldNameException
```

This exception is thrown when there is a problem with a field name encountered within an SQL expression.

FieldTypeException

public class **FieldTypeException**
extends Exception

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----com.ibm.mq.jms.FieldTypeException
```

This exception is thrown when there is a problem with the type of a field encountered within an SQL expression.

IntErrorException

public class **IntErrorException**
extends JMSException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----javax.jms.JMSException
                           |
                           +----com.ibm.mq.jms.IntErrorException
```

This exception is thrown when there is an internal problem with one of the JMS classes.

ISSLException

public class **ISSLException**
extends Exception

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----com.ibm.mq.jms.ISSLException
```

Indicates an SSL exception.

JMSInvalidParameterException

public class **JMSInvalidParameterException**
extends JMSException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----javax.jms.JMSException
                           |
                           +----com.ibm.mq.jms.JMSInvalidParameterException
```

Indicates that an invalid parameter was passed on a method call.

JMSListenerSetException

public class **JMSListenerSetException**
extends JMSException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----javax.jms.JMSException
                           |
                           +----com.ibm.mq.jms.JMSListenerSetException
```

JMSListenerSetException indicates that a MessageListener is set. MessageConsumer.receive() cannot be called if a MessageListener is set. A MessageListener cannot be set on both MessageConsumer and Session.

JMSMessageQueueOverflowException

public class **JMSMessageQueueOverflowException**
extends JMSException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----javax.jms.JMSException
                           |
                           +----com.ibm.mq.jms.JMSMessageQueueOverflowException
```

Indicates that a MessageQueue overflow has occurred.

JMSNotActiveException

public class **JMSNotActiveException**
extends JMSException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----javax.jms.JMSException
                           |
                           +----com.ibm.mq.jms.JMSNotActiveException
```

JMSNotActiveException is thrown if a resource has been closed.

JMSNotSupportedException

public class **JMSNotSupportedException**
extends JMSException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----javax.jms.JMSException
                           |
                           +----com.ibm.mq.jms.JMSNotSupportedException
```

JMSNotSupportedException is thrown by methods not currently supported, for example, the application server facilities when using a direct connection to a broker..

JMSParameterIsNullException

public class **JMSParameterIsNullException**
extends JMSException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----javax.jms.JMSException
                           |
                           +----com.ibm.mq.jms.JMSParameterIsNullException
```

Indicates that the value of a method parameter is null. In the case of String parameters, the value might have been the empty String.

MulticastHeartbeatTimeoutException

public class **MulticastHeartbeatTimeoutException**
extends JMSException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----javax.jms.JMSException
                           |
                           +----com.ibm.mq.jms.MulticastHeartbeatTimeoutException
```

A MulticastHeartbeatTimeoutException indicates that the multicast connection timed out, which has resulted in the client being disconnected. This might indicate network problems or that the broker is no longer running. If the network load is high, then it might be necessary to increase the broker heartbeat timeout value to avoid heartbeat timeouts. See your Broker documentation for details of how to do this.

MulticastPacketLossException

public class **MulticastPacketLossException**
extends JMSException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----javax.jms.JMSException
                           |
                           +----com.ibm.mq.jms.MulticastPacketLossException
```

A MulticastPacketLossException is thrown if a multicast receiver detects lost messages, and the messages cannot be retransmitted because they have been purged from the brokers's buffer. This exception is only thrown if the multicast connection is of type JMSC.MQJMS_MULTICAST_RELIABLE .

If the message is no longer in the brokers transmission buffer, then it might be necessary to increase the Broker Minimal History Size, or the Broker History Cleaning Time, or both, to give the client more time to request retransmission of lost messages. See the Broker documentation for details of how to do this.

NoBrokerResponseException

public class **NoBrokerResponseException**
extends JMSException

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----javax.jms.JMSException
                           |
                           +----com.ibm.mq.jms.NoBrokerResponseException
```

Indicates that a response was requested for a broker command, but none was received within the timeout interval.

SyntaxException

public class **SyntaxException**
extends Exception

```
java.lang.Object
   |
   +----java.lang.Throwable
           |
           +----java.lang.Exception
                   |
                   +----com.ibm.mq.jms.SyntaxException
```

Indicates that there is a syntax error in an SQL expression.

Chapter 17. Package com.ibm.mq.jms.services

This package provides services which are used by the com.ibm.mq.jms package.

MQJMS_Messages

public final class **MQJMS_Messages**
extends Object

```
java.lang.Object
   |
   +----com.ibm.mq.jms.services.MQJMS_Messages
```

Defines the most common exceptions that can be generated by WebSphere MQ JMS. It does not include all messages that can be written to a trace file. If you receive an exception message not in this list (except in a trace file), or if the cause seems to be an error in WebSphere MQ JMS, contact your IBM service representative.

A JMSException might have an embedded exception that contains a WebSphere MQ reason code. For an explanation of each WebSphere MQ reason code, see the *WebSphere MQ Messages* and *WebSphere MQ z/OS Messages and Codes* books.

Reading variables in a message: Some messages display text or numbers that vary according to the circumstances giving rise to the message; these are known as message variables. Message variables are indicated by the use of numbers in braces; for example, {0}, {1}, and so on.

Fields

MQJMS_ADMIN_BIND_FAIL

```
public final static java.lang.String
```

Unable to bind object.
Explanation: Administration service bind or copy or move operation failed.
User Response: Check that you have correctly set up your JNDI provider.

MQJMS_ADMIN_BND_NONADMIN

```
public final static java.lang.String
```

Binding non-administerable or not found.
Explanation: From JMSAdmin, an object was specified on the command line that either does not exist, or is not an object that JMSAdmin can administer.
User Response: Specify a valid object on the JMSAdmin command line.

MQJMS_ADMIN_CLASH_CLIENT

```
public final static java.lang.String
```

Clash of the client properties.
Explanation: Client properties specified for a bindings connection are not compatible.
User Response: Ensure the ConnectionFactory properties are correct.

MQJMS_ADMIN_CLASH_EXITINIT

```
public final static java.lang.String
```

ExitInit string supplied without Exit string.
Explanation: ExitInit string supplied but Exit is not set.
User Response: Set appropriate exit, or unset ExitInit string.

MQJMS_ADMIN_CNT_OPEN_CFG

```
public final static java.lang.String
```

Cannot open configuration file.
Explanation: Configuration file might not exist.
User Response: Check that the MQ_JAVA_INSTALL_PATH environment variable exists and that it specifies the installation directory of the base Java classes.

MQJMS_ADMIN_CONF_MISSING

```
public final static java.lang.String
```

Missing configuration properties.
Explanation: The Provider URL or InitialContextFactory are undefined
User Response: Ensure that these parameters are defined. See "Using the WebSphere MQ JMS administration tool" for JMSAdmin and JNDI information.

MQJMS_ADMIN_CONVERT_CIPHER

```
public final static java.lang.String
```

Matching CipherSpec {0} to CipherSuite {1} not possible.
Explanation: The supplied CipherSpec did not match to a known CipherSuite
User Response: Check the supplied CipherSpec is supported and retry.

MQJMS_ADMIN_CTX_NOT_EMPTY

```
public final static java.lang.String
```

Context is not empty.
Explanation: Error deleting Context due to context not being empty.
User Response: Remove context contents before trying to delete.

MQJMS_ADMIN_CTX_NOTFND

```
public final static java.lang.String
```

Context not found.
Explanation: Could not find a context to match the name given.
User Response: Ensure the correct context name is specified.

MQJMS_ADMIN_CTX_NOTFNDU

```
public final static java.lang.String
```

Context not found or cannot be deleted.
Explanation: The specified child context could not be deleted.
User Response: Ensure the correct context name was specified.

MQJMS_ADMIN_ERROR

```
public final static java.lang.String
```

Error.
Explanation: An internal error has occurred.
User Response: Contact your system administrator.

MQJMS_ADMIN_ICF_NOT_FOUND

```
public final static java.lang.String
```

Class specified by INITIAL_CONTEXT_FACTORY not found in CLASSPATH.
User Response: Check the class name or add the class to the CLASSPATH.

MQJMS_ADMIN_INV_PROP

```
public final static java.lang.String
```

Parameter {1} for method {0} was not valid.
Explanation: The parameter {1} supplied to method {0} was incorrectly specified.
User Response: Correct the parameter supplied to {0} and retry.

MQJMS_ADMIN_INV_PROP_CTX

```
public final static java.lang.String
```

The property was not valid in this context.
Explanation: JMSAdmin object value is not valid in the current context.
User Response: Check the property being specified.

MQJMS_ADMIN_INV_PROP_VAL

```
public final static java.lang.String
```

Value {1} for property {0} was not valid.
Explanation: The value supplied for parameter {0} is not valid.
User Response: Correct the value supplied and retry.

MQJMS_ADMIN_INVALID_AUTH_TYPE

```
public final static java.lang.String
```

The authentication type supplied was not valid - used **none**.
Explanation: The Admin Service JNDI initialization parameters contained an authorization scheme which was not valid, so **none**is used as the value instead.

MQJMS_ADMIN_INVALID_NAME

```
public final static java.lang.String
```

A name was supplied that was not valid.
Explanation: JMSAdmin error. The name supplied when trying to delete a context was invalid.
User Response: Check and correct the name supplied.

MQJMS_ADMIN_JNDI_INITFAIL

```
public final static java.lang.String
```

JNDI initialization failed.
User Response: Check your JNDI settings and service for additional information on the cause of this problem run with the -v argument.

MQJMS_ADMIN_LEXERR

```
public final static java.lang.String
```

Lexical error.
User Response: Contact your IBM representitive.

MQJMS_ADMIN_MV_SEMIFAIL

```
public final static java.lang.String
```

Unable to complete MOVE; a COPY has been done instead.
Explanation: The object was copied, but the original object could not be deleted.
User Response: Delete the original object manually.

MQJMS_ADMIN_NEW_CTX_FAIL

```
public final static java.lang.String
```

Unable to create context.
Explanation: Administration service failed.
User Response: Check LDAP and JNDI settings.

MQJMS_ADMIN_NON_MQJMS

```
public final static java.lang.String
```

Object is not a WebSphere MQ JMS administered object.
User Response: Ensure you only attempt to administer WebSphere MQ JMS objects through the JMSAdmin tool.

MQJMS_ADMIN_OBJ_INACTIVE

```
public final static java.lang.String
```

Object is inactive, so cannot perform directory operations.
Explanation: The JNDI service is inactive.
User Response: Contact your system administrator.

MQJMS_ADMIN_OBJTYPE_MISMATCH

```
public final static java.lang.String
```

Expected and actual object types do not match.
Explanation: Requested and retrieved objects are of different types.
User Response: Check that you have specified the correct object type.

MQJMS_ADMIN_PROP_UNK

```
public final static java.lang.String
```

Unknown parameter: {0}.
Explanation: The parameter {0} is unknown.
User Response: Check the parameter to be set and retry.

MQJMS_ADMIN_PROPVAL_NULL

```
public final static java.lang.String
```

Parameter {0} is null.
Explanation: A null value was supplied when a non-null Object was expected.
User Response: Check that the supplied parameter for {0} has been correctly instantiated and retry.

MQJMS_ADMIN_SYN_ERR

```
public final static java.lang.String
```

Syntax error.
Explanation: The command syntax was not valid.
User Response: Check the command given and retry.

MQJMS_ADMIN_VAL_OBJ_FAIL

```
public final static java.lang.String
```

Unable to create a valid object.
Explanation: Consistency check failed.
User Response: Check the parameters supplied or contact your IBM representative.

MQJMS_ADMIN_WS_INST

```
public final static java.lang.String
```

Unable to create a WebSphere MQ specific class. The WebSphere MQ classes might not have been installed or added to the CLASSPATH.
User Response: Check your WebSphere Application Server installation and CLASSPATH variable.

MQJMS_CLIENTID_FIXED

```
public final static java.lang.String
```

Client ID cannot be set after connection has been used.
Explanation: The Client ID of a connection can be set only once, and only before the connection is used.
User Response: Set the clientID before using the connection.

MQJMS_CLIENTID_NO_RESET

```
public final static java.lang.String
```

Resetting the Client ID is not allowed.
Explanation: The Client ID of a connection can be set only once, and only before the connection is used.
User Response: Set the Client ID before using the connection.

MQJMS_DIR_IMB_BADSOCKNAME

```
public final static java.lang.String
```

The socket family name: {0} is not valid.
Explanation: A socket family name given to an instance of IMBSocketFactory was not valid. {0} shows the bad name.
User Response: Contact your IBM representative.

MQJMS_DIR_IMB_NOCLASS

```
public final static java.lang.String
```

An exception occurred while attempting to load socket factory class {0}, exception:.{1}
Explanation: Either a ClassNotFoundException, an InstantiationException or an IllegalAccessException occurred while trying to load a particular IMBSocketFactory. {1} gives the name of the exception.
User Response: Contact your IBM representative.

MQJMS_DIR_JMS_BADID

```
public final static java.lang.String
```

The specified JMSMessageID, {0}, is not valid.
Explanation: Incorrect syntax was used to specify a message ID in Message.setJMSMessageID(). The correct syntax is ID:[0-9]+.
User Response: Check parameters.

MQJMS_DIR_JMS_BADNUM

```
public final static java.lang.String
```

Exception {1} occurred when initializing parameter {0}.
Explanation: {0} identifies the parameter that failed to initialize and {1}, the caught

exception.
User Response: Contact your IBM representative.

MQJMS_DIR_JMS_BDTOPIMPL

```
public final static java.lang.String
```

The {0} implementation of Topic is not supported.
Explanation: The Topic instance passed to a TopicPublisher or TopicSession method has an unsupported run-time implementation. {0} gives the class name of the unsupported implementation.
User Response: Check the Topic used and retry.

MQJMS_DIR_JMS_CLOSED

```
public final static java.lang.String
```

Operation not permitted on an entity that is closed.
Explanation: An operation was requested on a closed publisher, session, or connection.
User Response: Ensure that the publisher, session, or connection is open before trying this operation.

MQJMS_DIR_JMS_FMTINT

```
public final static java.lang.String
```

An attempt was made to read from a Stream message before a previous read has completed.
Explanation: Internal error.
User Response: Contact your IBM representative.

MQJMS_DIR_JMS_INVPRI

```
public final static java.lang.String
```

A JMSPriority level of {0} is outside the range specified in JMS.
Explanation: {0} gives the value that is in error.
User Response: Check the value specified and retry.

MQJMS_DIR_JMS_KILLMON

```
public final static java.lang.String
```

The client-side connection monitor is terminating.
User Response: Restart the connection.

MQJMS_DIR_JMS_LSTACT

```
public final static java.lang.String
```

Attempted to synchronously receive on a MessageConsumer for which a listener is active.
Explanation: MessageConsumer.receive() was called but a message listener is already active on the connection. This is not allowed under the JMS specification.

MQJMS_DIR_JMS_NEXCLIS

```
public final static java.lang.String
```

No ExceptionListener was set.
User Response: Create an ExceptionListener.

MQJMS_DIR_JMS_NOMORE

```
public final static java.lang.String
```

No more client parameter changes allowed.
Explanation: Internal error. An attempt was made to set a SessionConfig parameter when no more changes are allowed.
User Response: Contact your IBM representative.

MQJMS_DIR_JMS_NOTHDPOOL

```
public final static java.lang.String
```

An exception occurred while attempting to load thread pooling support; {0}.
Explanation: An exception occurred while attempting to load thread pooling support in the JMS client. {0} gives details of the exception.
User Response: Contact your IBM representative.

MQJMS_DIR_JMS_PBIOERR

```
public final static java.lang.String
```

An IOException occurred while publishing; {0}.
Explanation: An IOException occurred while publishing a message. {0} gives details of the exception.
User Response: See "Writing WebSphere MQ JMS publish/subscribe applications" for more information.

MQJMS_DIR_JMS_PBNOWLD

```
public final static java.lang.String
```

Topic {0} contains a wildcard, which is invalid for publishing.
Explanation: The Topic specified to a TopicPublisher method contained a wildcard. Wildcards are not allowed in Topics when publishing messages. The failing Topic is given by {0}.
User Response: Check the Topic specified and retry.

MQJMS_DIR_JMS_RUNKEXC

```
public final static java.lang.String
```

An exception occurred during synchronous receive; {0}.
Explanation: Internal error. {0} gives details of the exception.
User Response: Restart connection.

MQJMS_DIR_JMS_TCFLERR

```
public final static java.lang.String
```

An exception occurred while creating the TopicConnection; {0}.
Explanation: {0} gives details of the exception.
User Response: Contact your IBM representative.

MQJMS_DIR_JMS_TCSTSTP

```
public final static java.lang.String
```

An IOException occurred when starting or stopping delivery on the connection; {0}.
Explanation: {0} gives details of the exception.
User Response: Restart the connection.

MQJMS_DIR_JMS_THDEXC

```
public final static java.lang.String
```

An exception occurred while initializing a thread pool instance; {0}.
Explanation: A SocketThreadPoolException occurred while initializing a thread pool instance in the JMS client. {0} gives details of the exception.
User Response: Contact your IBM representative.

MQJMS_DIR_JMS_TMPVIO

```
public final static java.lang.String
```

Wrong use of temporary topic with the current connection.
Explanation: Attempted to use a temporary topic not created on the current connection.
User Response: Check the TemporaryTopic and retry.

MQJMS_DIR_JMS_TSBADMTC

```
public final static java.lang.String
```

While creating a TopicSubscriber attempting to add the subscription to the matching engine resulted in exception: {0}.
Explanation: {0} gives details of the exception.
User Response: Check the Exception given and retry.

MQJMS_DIR_JMS_TSIOERR

```
public final static java.lang.String
```

An IOException occurred while subscribing; {0}.
Explanation: An IOException occurred while subscribing. {0} gives details of the exception.
User Response: See "Writing WebSphere MQ JMS publish/subscribe applications" for more information.

MQJMS_DIR_MIN_ACK_NOT_SUPPORTED

```
public final static java.lang.String
```

Client Acknowledge is not supported for transport type DIRECT.
Explanation: Client Acknowledge mode was requested when creating a session. This is not supported with a DirectIP transport type.
User Response: Change either the transport type or acknowledge mode requested.

MQJMS_DIR_MIN_AUTHREJ

```
public final static java.lang.String
```

Minimal client connection rejected because of authentication failure.
User Response: Check authentication details.

MQJMS_DIR_MIN_BADBRKMSG

```
public final static java.lang.String
```

The broker sent a message during authentication which was not valid.
User Response: Check authentication details and retry.

MQJMS_DIR_MIN_BADFIELD

```
public final static java.lang.String
```

The value of field {0} is not valid.
Explanation: {0} shows the value used.
User Response: Check the value and retry.

MQJMS_DIR_MIN_BADMSG

```
public final static java.lang.String
```

Connector.send() was called with a message value that is not valid.
User Response: See "Writing WebSphere MQ base Java programs" for more information.

MQJMS_DIR_MIN_BADTOP

```
public final static java.lang.String
```

A specified topic was malformed, topic; {0}.
Explanation: {0} gives the name of the malformed topic.
User Response: Check the topic and retry.

MQJMS_DIR_MIN_BRKERR

```
public final static java.lang.String
```

The broker indicated an error on the minimal client connection.
User Response: Refer to JMS or broker documentation. Contact your IBM representative.

MQJMS_DIR_MIN_EOF

```
public final static java.lang.String
```

End of file (EOF) was encountered while receiving data in the minimal client.
User Response: Contact your IBM representative.

MQJMS_DIR_MIN_EXP_NOT_SUPPORTED

```
public final static java.lang.String
```

A Topic Expiry greater than zero not supported for transport type DIRECT.
Explanation: An Expiry time that is not valid was specified when using DirectIP transport.
User Response: Check the topic expiry and retry.

MQJMS_DIR_MIN_INTERR

```
public final static java.lang.String
```

An unexpected internal error occurred in the minimal client.
Explanation: Internal problem.
User Response: Contact your IBM representative.

MQJMS_DIR_MIN_NOQOP

```
public final static java.lang.String
```

No QOP available in the minimal client.
Explanation: Indicates that QOP is not implemented in the current version of the minimal client.
User Response: Contact your IBM representative.

MQJMS_DIR_MIN_NOSUB

```
public final static java.lang.String
```

Unauthorized subscription to topic {0}.
Explanation: Attempted to create a subscription to a Topic that is not authorized for the client. {0} shows the Topic.
User Response: Check the topic used and retry.

MQJMS_DIR_MIN_NOTBYTES

```
public final static java.lang.String
```

A Byte message operation was requested on something that is not a Byte message.
Explanation: The wrong message type was found.
User Response: Check message type before performing type specific operations.

MQJMS_DIR_MIN_NOTMAP

```
public final static java.lang.String
```

A Map message operation was requested on something that is not a Map message.
Explanation: The wrong message type was found.
User Response: Check message type before performing type specific operations.

MQJMS_DIR_MIN_NOTSTREAM

```
public final static java.lang.String
```

A Stream message operation was requested on something that is not a Stream message.
Explanation: The wrong message type was found.
User Response: Check message type before performing type specific operations.

MQJMS_DIR_MIN_NOTTEXT

```
public final static java.lang.String
```

A Text message operation was requested on something that is not a Text message.
Explanation: The wrong message type was found.
User Response: Check message type before performing type specific operations.

MQJMS_DIR_MIN_NOXASUP

```
public final static java.lang.String
```

Transport type DIRECT within a transaction is not supported.
Explanation: The application attempted to use a transactional method. This is not supported with a DirectIP transport type.

MQJMS_DIR_MIN_PER_NOT_SUPPORTED

```
public final static java.lang.String
```

Persistent messages not supported for transport type DIRECT.
Explanation: A DeliveryMode that is not valid was specified when using DirectIP transport.
User Response: Check DeliveryMode and retry.

MQJMS_DIR_MIN_SECLDERR

```
public final static java.lang.String
```

An exception occurred while loading the minimal client security implementation.
User Response: Contact your IBM representative.

MQJMS_DIR_MIN_TTL_NOT_SUPPORTED

```
public final static java.lang.String
```

Time to Live greater than zero not supported for transport type DIRECT.
Explanation: A time to live that is not valid was specified when using DirectIP transport.
User Response: Check Time to Live and retry.

MQJMS_DIR_MIN_UNVPRO

```
public final static java.lang.String
```

The broker requested an unavailable protocol during authentication.
User Response:

MQJMS_DIR_MIN_UNXEXC

```
public final static java.lang.String
```

An unexpected exception in minimal client; {0}.
Explanation: An unusual or unexpected exception occurred at the minimal client. {0} gives more details.
User Response: Contact your IBM representative.

MQJMS_DIR_MTCH_BDESC

```
public final static java.lang.String
```

The escape sequence was used to terminate the pattern; {0}.
Explanation: This might indicate a syntax error in your Selector.
User Response: Check your selector and retry.

MQJMS_DIR_MTCH_BDESCL

```
public final static java.lang.String
```

The escape sequence {0} passed to the pattern tool is longer than one character.
Explanation: This might indicate a syntax error in your Selector.
User Response: Check your selector and retry.

MQJMS_DIR_MTCH_BDMSG

```
public final static java.lang.String
```

An exception occurred while attempting to access a field of a message; {0}.
Explanation: A corrupt message format was discovered. Internal error.
User Response: Contact your IBM representative.

MQJMS_DIR_MTCH_BDSEP

```
public final static java.lang.String
```

The Topic segment separator {0} appears in an incorrect position.
Explanation: A subscription Topic separator was used incorrectly. {0} shows the bad separator.
User Response: Check your Topic definitions and retry.

MQJMS_DIR_MTCH_BDWLD

```
public final static java.lang.String
```

An incorrect use of the Topic wildcard character {0} was detected.
Explanation: The failed Topic is given by parameter {0}.
User Response: Check your Topic definitions and retry.

MQJMS_DIR_MTCH_ECNMIN

```
public final static java.lang.String
```

An EvalCache get or put operation specified an invalid ID.
Explanation: Internal Error. The MinValue of an EvalCache could not be increased, although the operation expected it to be.
User Response: Contact your IBM representative.

MQJMS_DIR_MTCH_NULCH

```
public final static java.lang.String
```

An attempt was made to remove an object with a Topic {0} from the matching engine, but it did not have a cache entry: {1}.
Explanation: Internal error. An attempt was made to remove an object from a null tree in the matching engine. {0} gives the Topic and {1} gives the cache entry.
User Response: Contact your IBM representative.

MQJMS_DIR_MTCH_NULRM

```
public final static java.lang.String
```

An attempt was made to remove an object with Topic {0} from an empty matching engine: {1}.
Explanation: Internal error. An attempt was made to remove an object from a null tree in the matching engine. {0} gives the Topic and {1} gives the MatchTarget.
User Response: Contact your IBM representative.

MQJMS_DIR_MTCH_PSTPER

```
public final static java.lang.String
```

An exception occurred while parsing a subscription selector; {0}.
Explanation: A TypeCheckException occurred while loading or invoking the match parser. This might indicate a syntax error in your Selector.
User Response: Check your selector and retry.

MQJMS_DIR_MTCH_TOMNY

```
public final static java.lang.String
```

Too many content attributes were specified.
Explanation: Internal Error. Too many non-topic attributes were specified.
User Response: Contact your IBM representative.

MQJMS_DIR_MTCH_UNKEXC

```
public final static java.lang.String
```

An unexpected exception occurred in the matching engine: {0}.
User Response: Contact your IBM representative.

MQJMS_DIR_MTCH_UNXTYP

```
public final static java.lang.String
```

A message field was expected to contain a value of type {0} but contained one of type {1}.
Explanation: This might indicate a syntax error in your Selector.
User Response: Check your selector and retry.

MQJMS_E_11_INVALID_CROSS_DOMAIN

```
public final static java.lang.String
```

Attribute for a domain specific object was not valid.
Explanation: A JMS application attempted to set an attribute of a domain specific

object, but the attribute is valid only for the other messaging domain.
User Response: Make sure that the JMS object types used by your application are relevant for the required messaging domain. If your application uses both messaging domains, consider using domain independent objects throughout the application.

MQJMS_E_11_INVALID_DOMAIN_SPECIFIC

```
public final static java.lang.String
```

Operation for a domain specific object was not valid.
Explanation: A JMS application attempted to perform an operation on domain specific object, but the operation is valid only for the other messaging domain.
User Response: Make sure that the JMS objects used by your application are relevant for the required messaging domain. If your application uses both messaging domains, consider using domain independent objects throughout the application.

MQJMS_E_11_SERVICES_NOT_SETUP

```
public final static java.lang.String
```

The required queues or publish/subscribe services are not set up: {0}.
Explanation: The required WebSphere MQ setup for the messaging domain is not complete.
User Response: For the point-to-point messaging, make sure that you have started the queue manager and, if your JMS application is connecting as a client application, make sure that you have started a listener for the correct port. For publish/subscribe messaging, make sure that you have done the post installation setup.

MQJMS_E_BAD_CCSID

```
public final static java.lang.String
```

The character set {0} is not supported.
Explanation: An attempt was made to send or receive a map message, stream message or text message whose body is encoded using a character set not supported by the JVM. In the case of text messages, this exception might be thrown when the body of the message is first queried, rather than at receive time.
User Response: Only encode a message using a character set known to be available to the receiving application.

MQJMS_E_BAD_DEST_STR

```
public final static java.lang.String
```

Failed to reconstitute destination from {0}.
Explanation: A message has been received which contains destination information in the RFH2 header which is not valid.
User Response: Ensure that any messages being sent by non-JMS applications have correctly formatted destination information. In the case of RFH2 headers, pay special attention to the **Rto**(reply to) and **Dst**(destination) elements of the XML portion of the header. Valid destination strings must start either **queue**or **topic**.

MQJMS_E_BAD_ELEMENT_NAME

```
public final static java.lang.String
```

Message element name: {0} is not valid.
Explanation: Attempted to set a message property using either a property name

which is not valid, or the name of a property which cannot have its value set.
User Response: Ensure that the property name specified conforms to the JMS specification. If the property name supplied is that of a JMS property, or a vendor specific extension, ensure that this property name can be set.

MQJMS_E_BAD_EXIT_CLASS

```
public final static java.lang.String
```

Failed to create instance of exit class {0}.
User Response: Check that the supplied String matches the fully-qualified name of the exit class, and that the exit implements the correct interface.

MQJMS_E_BAD_PROPERTY_NAME

```
public final static java.lang.String
```

Invalid message property name: {0}.
Explanation: Attempted to set a property that either does not have a valid property name, or cannot be set.
User Response: Ensure that the property name used is a valid property name in accordance with the JMS specification. If the property name refers to a JMS or provider-specific extension property, ensure that this property can be set.

MQJMS_E_BAD_RFH2

```
public final static java.lang.String
```

The MQRFH2 header has an incorrect format.
Explanation: Received a message with a badly formed RFH2 header.
User Response: Ensure that any non-JMS applications building messages with RFH2 headers create well-formed RFH2 headers.

MQJMS_E_BAD_TIMEOUT

```
public final static java.lang.String
```

Timeout not valid for WebSphere MQ.
Explanation: An attempt was made to invoke the receive method on either a QueueReceiver or TopicSubscriber, specifying a long timeout value that is not valid.
User Response: Ensure the timeout value specified is not negative and not greater than the value of Integer.MAX_VALUE.

MQJMS_E_BAD_TYPE

```
public final static java.lang.String
```

The property or element in the message has an incompatible datatype {0}.
Explanation: Attempted to retrieve a property from a JMS message using a accessor method which specifies an incompatible type. For example, attempting to retrieve an integer property using the getBooleanProperty() method.
User Response: Use an accessor method defined by the JMS specification as being able to retrieve property values of the required type.

MQJMS_E_BATCH_SIZE

```
public final static java.lang.String
```

Incorrect message batch size (must be greater than zero).
Explanation: An incorrect batch size parameter was passed to createConnectionConsumer() or createDurableConnectionConsumer().

User Response: Set a batch size greater than zero. In a J2EE application server, this might represent an error in the application server. Refer to your application server's documentation.

MQJMS_E_BROKER_MESSAGE_CONTENT

```
public final static java.lang.String
```

The broker control message content: {0} is not valid.
Explanation: {0} explains further.
User Response: Check the broker documentation for message content information.

MQJMS_E_BROWSE_MSG_FAILED

```
public final static java.lang.String
```

Failed to browse message.
Explanation: No message was available for browsing, possibly because there was no message on the Queue.
User Response: Check the linked WebSphere MQ Exception reason and completion codes. Check that a message is available for browsing.

MQJMS_E_BYTE_TO_STRING

```
public final static java.lang.String
```

The JMS client attempted to convert a byte array to a String.
Explanation: Attempted to receive a byte array from a stream message using the readString() method.
User Response: Either use the appropriate method to receive the data, or format the data placed into the stream message correctly.

MQJMS_E_CC_MIXED_DOMAIN

```
public final static java.lang.String
```

Mixed-domain consumers acting on the same input is forbidden.
Explanation: A point-to-point ConnectionConsumer is using the subscriber queue of a publish/subscribe ConnectionConsumer.
User Response: Do not attempt to access subscriber queues using the point-to-point ConnectionConsumer facilities of JMS. Check your TopicConnectionFactory and Topic objects to make sure they are not using a QLOCAL intended for use by point-to-point applications as a subscriber queue.

MQJMS_E_CLEANUP_NONE_REQUESTED

```
public final static java.lang.String
```

Cleanup level of NONE requested.
Explanation: Cleanup requested while cleanupLevel set to NONE.
User Response: Set cleanupLevel property to an appropriate value.

MQJMS_E_CLEANUP_REP_BAD_LEVEL

```
public final static java.lang.String
```

Level for repeating Cleanup is not valid.
User Response: Set cleanupLevel property to an appropriate value.

MQJMS_E_CLOSE_FAILED

```
public final static java.lang.String
```

Close failed because of {0}.
Explanation: Internal Error. {0} indicates the reason for the error.
User Response: Contact your IBM representative.

MQJMS_E_CONN_DEST_MISMATCH

```
public final static java.lang.String
```

Connection and Destination mismatch.
Explanation: An operation was requested, but the Destination class is incompatible with the Connection class. Topics cannot be used with QueueConnections and Queues cannot be used with TopicConnections.
User Response: Supply a suitable Destination. If this error represents an internal error condition in JMS contact your IBM representative.

MQJMS_E_DELIVERY_MODE_INVALID

```
public final static java.lang.String
```

The delivery mode was not valid.
Explanation: Either the value for the delivery mode of a message producer was incorrectly specified, or the mode value was specified incorrectly when publishing a message.
User Response: Check to ensure that the value specified is a valid enumeration for delivery mode.

MQJMS_E_DESERIALISE_FAILED

```
public final static java.lang.String
```

Unable to deserialize an object.
Explanation: Deserialization of an ObjectMessage failed.
User Response: Ensure that the ObjectMessage being received contains valid data. Ensure that the class files representing object data contained within the ObjectMessage are present on the machine deserializing the ObjectMessage. If the object contained within the ObjectMessage references other objects, ensure that these class files are also present.

MQJMS_E_DISCARD_FAILED

```
public final static java.lang.String
```

Error while discarding message.
Explanation: JMS encountered an error while discarding a message, or while generating an exception report for a message to be discarded.
User Response: Use the linked exception to determine the cause of this error.

MQJMS_E_DLH_READ_FAILED

```
public final static java.lang.String
```

Error reading dead letter header.
Explanation: JMS attempted to interpret a message with a dead letter header, but encountered a problem.
User Response: Use the linked exception to determine the cause of this error.

MQJMS_E_DLH_WRITE_FAILED

```
public final static java.lang.String
```

Error writing dead letter header.
Explanation: JMS attempted to requeue a message to the dead letter queue, but

failed to construct a dead letter header.
User Response: Use the linked exception to determine the cause of this error.

MQJMS_E_DLQ_FAILED

```
public final static java.lang.String
```

Unable to write a message to dead letter queue.
Explanation: JMS attempted to requeue a message to the dead letter queue, but failed.
User Response: Use the linked exception to determine the cause of this error. If there is no linked exception, check that the queue manager has a defined a dead letter queue. Once JMS has sent a message to the dead letter queue, the reason code stored in the message's dead letter header can be used to determine why the message was sent there.

MQJMS_E_EXCP_LSTNR_FAILED

```
public final static java.lang.String
```

ExceptionListener threw exception: {0}.
User Response: Check linked exceptions for further information.

MQJMS_E_IDENT_PRO_INVALID_OP

```
public final static java.lang.String
```

The operation is not valid for the identified producer.
Explanation: The QueueSender.send() method has been used on an identified QueueSender, which contradicts the JMS specification.
User Response: For further information see the MQQueueSender class.

MQJMS_E_INTERNAL_ERROR

```
public final static java.lang.String
```

An internal error has occurred. Contact your system administrator. Detail: {0}.
User Response: Check the linked WebSphere MQ exception reason and completion code for more information.

MQJMS_E_INVALID_ESCAPE

```
public final static java.lang.String
```

An incorrect XML escape sequence detected {0}.
Explanation: An XML escape sequence was encountered which is not valid in the RFH2 header of a received message.
User Response: Ensure that only valid XML escape sequences are placed into any RFH2 headers built by non-JMS applications.

MQJMS_E_INVALID_HEX_STRING

```
public final static java.lang.String
```

String is not a valid hexadecimal number - {0}.
Explanation: Either an attempt was made to specify a group ID or correlation ID which starts with the prefix ID but is not followed by a well-formed hexadecimal value, or an attempt was made to receive a message which contains an RFH2 property of type bin.hex that does not have a well-formed hexadecimal value.
User Response: Ensure that a valid hexadecimal value always follows the ID prefix when setting group ID or correlation ID values. Ensure that any RFH2 headers generated by non-JMS applications are well-formed.

MQJMS_E_INVALID_MAP_MESSAGE

```
public final static java.lang.String
```

The map message has an incorrect format.
Explanation: A map message was received, but its RFH2 header information is badly formatted.
User Response: Ensure any non-JMS applications are building well-formed RFH2 header information for inclusion in map messages.

MQJMS_E_INVALID_MESSAGE

```
public final static java.lang.String
```

Cannot transmit JMS messages which are not WebSphere MQ messages.
Explanation: Wrong message type used. This might be an internal problem.
User Response: Check the message type. Contact your IBM representative if there appears to be an internal error.

MQJMS_E_INVALID_SESSION

```
public final static java.lang.String
```

Session object not valid.
Explanation: The JMS ConnectionConsumer feature attempted to deliver a batch of messages to a Session. However, the Session contained in the ServerSession object returned by the ServerSessionPool was not a WebSphere MQ JMS Session.
User Response: This is an error in the ServerSessionPool. If you have supplied a ServerSessionPool, check its behavior. In a J2EE application server, this might represent an error in the application server; in which case, refer to your application server's documentation.

MQJMS_E_INVALID_STREAM_MESSAGE

```
public final static java.lang.String
```

The stream message has an incorrect format.
Explanation: A stream message was received, but its RFH2 header information is badly formatted.
User Response: Ensure that any non-JMS applications are building well-formed RFH2 header information for inclusion in stream messages.

MQJMS_E_INVALID_SURROGATE

```
public final static java.lang.String
```

An incorrect UTF-16 surrogate character detected {0}.
Explanation: A UTF-16 surrogate character was encountered which is not valid as part of a topic name or RFH2 property.
User Response: Ensure that, when specifying UTF-16, topic names or RFH2 properties are well-formed.

MQJMS_E_JNDI_GENERAL_ERROR

```
public final static java.lang.String
```

JNDI failed due to {0}.
Explanation: {0} gives further information.
User Response: Check settings for LDAP, JNDI, and in the JMSAdmin.config file.

MQJMS_E_LOG_ERROR

```
public final static java.lang.String
```

Failed to log error.
Explanation: Log settings might be incorrect. See the linked LogException.
User Response: Check that the log settings are correct.

MQJMS_E_MSG_LSTNR_FAILED

```
public final static java.lang.String
```

MessageListener threw: {0}.
Explanation: When performing asynchronous delivery, the onMessage() method of the application's MessageListener failed with a Throwable. WebSphere MQ JMS tries to redeliver or requeue the message.
User Response: Do not throw Throwable objects from the onMessage() method of a MessageListener.

MQJMS_E_MULTICAST_HEARTBEAT_TIMEOUT

```
public final static java.lang.String
```

Multicast connection disconnected due to timeout.
Explanation: No heartbeat packet was received when expected. There has probably been a transmitter failure or a network failure.
User Response: Check for network problems and check the broker is still running. If the network load is high, consider increasing the heartbeat interval.

MQJMS_E_MULTICAST_LOST_MESSAGES

```
public final static java.lang.String
```

Lost {0} messages in reliable multicast mode.
Explanation: A MulticastPacketLossException is thrown if a multicast receiver detects lost messages, and the messages cannot be retransmitted because they have been purged from the brokers's buffer.
User Response: Increase the Broker Minimal History Size and/or Broker History Cleaning Time to give the client more time to request retransmission of lost messages. See the Broker documentation for details of how to do this.

MQJMS_E_MULTICAST_NOT_AVAILABLE

```
public final static java.lang.String
```

Multicast connection cannot be established.
Explanation: Multicast is enabled, but a multicast connection could not be established and the application specified that fallback to unicast is disallowed.
User Response: Check the connection parameters and check that the broker is still running.

MQJMS_E_MULTICAST_PORT_INVALID

```
public final static java.lang.String
```

Cannot connect with a specific local port for multicast.
Explanation: A port number was specified. This is not valid for multicast.
User Response: Check the String supplied to com.ibm.mq.jms.MQConnectionFactory.setLocalAddress(String) is valid for multicast.

MQJMS_E_NO_BORQ

```
public final static java.lang.String
```

No Backout-Requeue queue defined.
Explanation: JMS encountered a message which has been backed out more than the queue's Backout Threshold, but the queue does not have a Backout-Requeue queue defined.
User Response: Define a Backout-Requeue queue for the queue, or set the Backout Threshold to zero to disable poison message handling. Investigate the repeated backouts.

MQJMS_E_NO_MSG_LISTENER

```
public final static java.lang.String
```

No message listener.
Explanation: The message listener has stopped or was never started.
User Response: Restart the message listener and retry.

MQJMS_E_NO_SESSION

```
public final static java.lang.String
```

Message has no session associated with it.
Explanation: An attempt was made to acknowledge a message on a session which is not in an open state.
User Response: Ensure that the session associated with the message has been correctly opened. Check that the session has not been closed.

MQJMS_E_NO_STR_CONSTRUCTOR

```
public final static java.lang.String
```

There is no constructor with a String argument.
Explanation: User exits must provide a constructor that takes a single String. No such constructor was found in the specified exit.
User Response: Ensure that the user exit provides a suitable constructor.

MQJMS_E_NO_UTF8

```
public final static java.lang.String
```

Fatal error - UTF-8 not supported.
Explanation: The Java runtime environment you are using does not support the UTF-8 character encoding. JMS requires support for this encoding to perform some operations.
User Response: Consult the documentation and or provider of your Java runtime environment to determine how to obtain support for the UTF-8 character encoding.

MQJMS_E_NO_XARESOURCE

```
public final static java.lang.String
```

Failed to obtain XAResource.
Explanation: JMS failed to create an XA Queue resource due to an error.
User Response: See the linked XAException for more information.

MQJMS_E_NON_LOCAL_RXQ

```
public final static java.lang.String
```

Non-local WebSphere MQ queue not valid for receiving or browsing.
Explanation: An attempt was made to perform an inappropriate operation on a non-local queue.
User Response: Check the queue properties.

MQJMS_E_NOT_ALLOWED_WITH_XA

```
public final static java.lang.String
```

Operation for an XA transacted session was not valid.
Explanation: The acknowledgement mode for a transacted session was not valid. Acknowledge and Recover are not valid operations in transacted sessions.

MQJMS_E_NOT_IMPLEMENTED

```
public final static java.lang.String
```

Not implemented.
Explanation: The function requested is not implemented. This can be thrown by message acknowledgement, if the session or acknowledgement parameters are not valid or are incorrect.

MQJMS_E_NULL_CONNECTION

```
public final static java.lang.String
```

No valid connection available.
Explanation: The queue is busy, there are network problems, or a connection has not been defined for the object.
User Response: Create a valid connection for this operation.

MQJMS_E_NULL_MESSAGE

```
public final static java.lang.String
```

Unable to process a null message.
Explanation: Internal error in WebSphere MQ JMS.
User Response: Contact your IBM representative.

MQJMS_E_NULL_POOL

```
public final static java.lang.String
```

Null ServerSessionPool has been provided.
Explanation: The ServerSessionPool specified on createConnectionConsumer() or createDurableConnectionConsumer() was null.
User Response: Set an appropriate ServerSessionPool. In a J2EE application server, this might represent an error in the application server. Refer to your application server's documentation.

MQJMS_E_QMGR_NAME_INQUIRE_FAILED

```
public final static java.lang.String
```

The queue manager name could not be queried.
Explanation: In createConnectionConsumer() or createDurableConnectionConsumer(), JMS was unable to determine the name of the queue manager.
User Response: Check your queue manager error logs for problems which might cause this. If there are no other error conditions, contact your IBM representative.

MQJMS_E_QUEUE_NOT_LOCAL_OR_ALIAS

```
public final static java.lang.String
```

Specified WebSphere MQ Queue is neither a QLOCAL nor a QALIAS queue.
Explanation: createConnectionConsumer() was called, but a queue of the wrong type was specified. Only QALIAS and QLOCAL queues can be used with the

ConnectionConsumer feature.
User Response: Specify a queue of the correct type.

MQJMS_E_READING_MSG

```
public final static java.lang.String
```

Exception occurred reading message body: {0}.
Explanation: JMS encountered an exception while reading data from a message. The message being read is likely to be a response message from the publish/subscribe broker.
User Response: Use the linked exception to determine the cause of this error.

MQJMS_E_RECOVER_BO_FAILED

```
public final static java.lang.String
```

Failed to recover: unacknowledged messages might not get redelivered.
Explanation: The system was unable to recover from a failure.
User Response: Check the linked exception to determine why the call to recover failed.

MQJMS_E_REDIRECT_FAILED

```
public final static java.lang.String
```

Failed to redirect message.
Explanation: When performing asynchronous delivery, WebSphere MQ JMS attempted to redirect the message to the backout queue. No backout queue was defined.
User Response: Ensure that the backout queue is defined. Also, investigate why WebSphere MQ JMS was attempting to redirect the message. It might do so in response to a failing MessageListener implementation.

MQJMS_E_REQUEUE_FAILED

```
public final static java.lang.String
```

Message requeue failed.
Explanation: JMS found an error when requeuing a message which has been backed out more than the queue's Backout Threshold.
User Response: Use the linked exception to determine the cause of this error. Investigate the repeated backouts.

MQJMS_E_RESOURCE_BUNDLE_NOT_FOUND

```
public final static java.lang.String
```

Failed to locate the resource bundle.
Explanation: The resource bundle is either not present or not in the application's CLASSPATH.
User Response: Check that the CLASSPATH includes the location of property files.

MQJMS_E_RFH_CONTENTS_ERROR

```
public final static java.lang.String
```

RFH content was unrecognized or not valid.
Explanation: JMS expected to find an RFH message header, but found it to be missing, malformed or lacking required data.
User Response: Investigate the source of the message. If this represents an internal error condition in JMS, contact your IBM representative.

MQJMS_E_RFH_READ_FAILED

```
public final static java.lang.String
```

Error reading RFH.
Explanation: JMS encountered an error while parsing an RFH message header.
User Response: Use the linked exception to determine the cause of this error.

MQJMS_E_RFH_WRITE_FAILED

```
public final static java.lang.String
```

Error writing RFH.
Explanation: JMS attempted to construct an RFH message header, but encountered an error.
User Response: Use the linked exception to determine the cause of this error.

MQJMS_E_ROLLBACK_FAILED

```
public final static java.lang.String
```

Failed to roll back.
Explanation: The system was unable to roll back to a safe state.
User Response: Check the linked WebSphere MQ Exception reason and completion codes for further information.

MQJMS_E_S390_DOUBLE_TOO_BIG

```
public final static java.lang.String
```

Number outside of range for double precision z/OS Float {0}.
Explanation: This is a z/OS specific error.

MQJMS_E_SECURITY_CREDS_INVALID

```
public final static java.lang.String
```

Security credentials cannot be specified when using WebSphere MQ bindings.
Explanation: The RRS queue does not support a client connection, and bindings connections do not support the specification of security credentials.
User Response: Ensure that you do not try to specify security credentials when using a bindings connection.

MQJMS_E_SERIALISE_FAILED

```
public final static java.lang.String
```

Unable to serialize an object.
Explanation: An attempt has been made to serialize an ObjectMessage which contains a non-serializable object.
User Response: Ensure that ObjectMessages only contain serializable objects. If the object placed inside an ObjectMessage references other objects, these must also be serializable.

MQJMS_E_SESSION_ASYNC

```
public final static java.lang.String
```

Operation cannot be performed while the session is using asynchronous delivery.
Explanation: You cannot perform the requested operation while the session is actively using asynchronous delivery mode.

MQJMS_E_SESSION_CLOSED

```
public final static java.lang.String
```

Session closed.
Explanation: The session was closed. It either timed out, or it was closed explicitly, or it was closed because the connection or the queue manager was closed.
User Response: Restart the session, and check all required resources are available.

MQJMS_E_SESSION_IS_TRANSACTED

```
public final static java.lang.String
```

The operation is not valid for a transacted session.
Explanation: The acknowledgement mode is not valid for a transacted session. Acknowledge and Recover are not valid operations in transacted sessions.

MQJMS_E_SESSION_NOT_TRANSACTED

```
public final static java.lang.String
```

The operation is not valid for a non-transacted session.
Explanation: Commit is not allowed on a session that is not transacted.
User Response: Check the linked IllegalStateException for more information.

MQJMS_E_START_FAILED

```
public final static java.lang.String
```

Start failed because of {0}.
Explanation: {0} indicates why the session failed to start.
User Response: Contact your IBM representative.

MQJMS_E_SYSTEM_PROPERTY_NOT_FOUND

```
public final static java.lang.String
```

Failed to find the system property {0}.
Explanation: The system property specified in {0} does not exist or was not found in the application's CLASSPATH.
User Response: Check the CLASSPATH settings and the product installation.

MQJMS_E_TMPQ_CLOSED

```
public final static java.lang.String
```

Temporary queue already closed or deleted.
Explanation: Temporary queue no longer exists or is equal to null.
User Response: Check to see that the queue has been created, and that the session is still available.

MQJMS_E_TMPQ_DEL_FAILED

```
public final static java.lang.String
```

Failed to delete temporary queue.
Explanation: The temporary queue might be persistent or busy.
User Response: See the linked exception for more details. Wait if the queue is busy, or delete the queue manually if it is persistent.

MQJMS_E_TMPQ_DEL_STATIC

```
public final static java.lang.String
```

Cannot delete a static queue.
Explanation: Attempted to delete a queue of type static, where a temporary queue was expected.
User Response: Check the expected queue type for deletion.

MQJMS_E_TMPQ_FAILED

```
public final static java.lang.String
```

Failed to create a temporary queue from {0}.
Explanation: Creation of temporary queue failed.
User Response: See linked exception for more information. Check that the TemporaryModel parameter against the QueueConnectionFactory is set to a valid model queue.

MQJMS_E_TMPQ_INUSE

```
public final static java.lang.String
```

Temporary queue in use.
Explanation: Another program is using the queue.
User Response: Wait for the temporary queue to become free or create another.

MQJMS_E_TMPT_DELETED

```
public final static java.lang.String
```

TemporaryTopic already deleted.
Explanation: TemporaryTopic no longer exists or is equal to null.
User Response: Check that the queue has been created, and that the session is still available.

MQJMS_E_TMPT_IN_USE

```
public final static java.lang.String
```

TemporaryTopic in use.
Explanation: Something else is currently using the topic.
User Response: Wait until the topic is free or create another topic. Ensure subscribers deregister when finished.

MQJMS_E_TMPT_OUTOFSCOPE

```
public final static java.lang.String
```

TemporaryTopic out of scope.
Explanation: The current connection ID does not match the connection that created the temporary topic.
User Response: Check that the topic has been created under this connection.

MQJMS_E_TRACE_FILE_NOT_FOUND

```
public final static java.lang.String
```

The trace file does not exist.
Explanation: Trace settings might be incorrect.
User Response: Check trace settings and trace file existence.

MQJMS_E_TRACE_STREAM_ERROR

```
public final static java.lang.String
```

Failed to connect to Trace stream.
Explanation: Trace settings might be incorrect.
User Response: Check Trace settings and retry.

MQJMS_E_UNIDENT_PRO_INVALID_OP

```
public final static java.lang.String
```

Operation not valid for unidentified producer.
Explanation: An attempt was made to send a message from an unidentified MessageProducer without specifying a Destination to send the message to.
User Response: Ensure that either an identified producer is used (that is, a Destination was supplied when the producer was created), or specify a Destination when the send method is invoked.

MQJMS_E_UNKNOWN_TARGET_CLIENT

```
public final static java.lang.String
```

Unknown value of target client: {0}.
Explanation: The value for the targetClient property set by the application for this destination is not recognized by WebSphere MQ JMS.
User Response: Check targetClient property and retry.

MQJMS_E_UNKNOWN_TRANSPORT

```
public final static java.lang.String
```

Unknown value of transportType: {0}.
Explanation: The value given for transportType was not valid. {0} shows the incorrect value.
User Response: Check transport type and retry.

MQJMS_E_UNREC_BROKER_MESSAGE

```
public final static java.lang.String
```

Unrecognized message from publish / subscribe broker.
Explanation: The message received from the broker was not of a recognized or supported format.
User Response: Check that the broker you are using is supported and refer to broker documentation for settings.

MQJMS_E_UNSUPPORTED_TYPE

```
public final static java.lang.String
```

Unsupported property or element datatype {0}.
Explanation: This error is caused by one of the following:

1. Attempting to set a property of a JMS message using an object which is not one of the supported types.
2. Attempting to set or receive a message whose RFH2 contains an element representing a property which does not have a valid type associated with it.

User Response: Ensure that when setting message properties, you use a valid JMS object type. If this exception occurs when receiving a message containing an RFH2 header sent by a non-JMS application, ensure that the RFH2 header is well-formed.

MQJMS_ERR_QSENDER_CLOSED

```
public final static java.lang.String
```

QueueSender is closed.
User Response: Open or reopen the queue sender if required.

MQJMS_EXC_ENLIST_FAILED

```
public final static java.lang.String
```

Enlist failed.
Explanation: JTSXA.enlist() threw an exception that was caught by JMS.
User Response: Check the linked exception, reason and completion codes for more information. Contact your IBM representative.

MQJMS_EXCEPTION_AUTHENTICATION_FAILED

```
public final static java.lang.String
```

The security authentication supplied for MQQueueManager was not valid.
Explanation: Bad user name, or password, or both. In bindings mode, a supplied user ID does not match the logged in user ID.
User Response: Check that the user IDs used by WebSphere MQ are all assigned to the relevant groups and given appropriate user permissions.

MQJMS_EXCEPTION_BAD_STATE_TRANSITION

```
public final static java.lang.String
```

Unhandled state transition from {0} to {1}.
Explanation: The state transition is not valid, see log for more information.
User Response: Check the linked WebSphere MQ exception reason and completion code.

MQJMS_EXCEPTION_BAD_VALUE

```
public final static java.lang.String
```

The value for {0}:{1} is not valid.
Explanation: The value {1} for property {0} is not correct.
User Response: Check the linked WebSphere MQ exception reason and completion code.

MQJMS_EXCEPTION_CONNECTION_CLOSED

```
public final static java.lang.String
```

Connection closed.
Explanation: An operation such as start() or stop() has been called on a connection that is already closed.
User Response: Ensure that the connection is open before performing any operation.

MQJMS_EXCEPTION_GET_MSG_FAILED

```
public final static java.lang.String
```

Failed to get message from an MQQueue object.
Explanation: JMS attempted to perform an MQGET; however WebSphere MQ reported an error.
User Response: Use the linked exception to determine the cause of this error.

MQJMS_EXCEPTION_ILLEGAL_STATE

```
public final static java.lang.String
```

Method {0} has been invoked at an illegal or inappropriate time, or the provider is not in an appropriate state for the requested operation.
Explanation: The normal reason for this exception is that the SSL certificate stores have not been defined. {0} identifies the method that has caused the problem.

MQJMS_EXCEPTION_INVALID_CLIENTID

```
public final static java.lang.String
```

JMS client attempted to set a client ID on a connection
Explanation: An application attempted to set the client ID property of a valid connection to null, or attempted to set the clientID property of an invalid connection.
User Response: The clientID property on a connection can only be set once, only to a non-null value, and only before the connection is used. Ensure that the connection is valid and that the clientID value is not null.

MQJMS_EXCEPTION_INVALID_DESTINATION

```
public final static java.lang.String
```

Destination not understood or no longer valid.
Explanation: The queue or topic might have become unavailable, the application might be using an incorrect connection for the queue or topic, or the supplied destination is not of the correct type for this method.
User Response: Check that WebSphere MQ is still running and the queue manager is available. Check that the right connection is being used for your queue or topic.

MQJMS_EXCEPTION_INVALID_SELECTOR

```
public final static java.lang.String
```

JMS Client has given JMS provider a message selector with incorrect syntax.
Explanation: The message selector string is empty or contains a value which is incorrect or has the wrong syntax.
User Response: Check the linked WebSphere MQ exception reason and completion codes for more information.

MQJMS_EXCEPTION_MESSAGE_EOF

```
public final static java.lang.String
```

Unexpected end of stream has been reached when a StreamMessage or BytesMessage is being read.
Explanation: The byte stream being read is shorter than the buffer supplied. This can also be caused by receiving a corrupt StreamMessage or BytesMessage.
User Response: Check the length of buffer supplied. Check system event logs for more information.

MQJMS_EXCEPTION_MESSAGE_FORMAT

```
public final static java.lang.String
```

JMS Client attempts to use a data type not supported by a message or attempts to read data in the wrong type.
Explanation: Wrong data types used to read message property types.
User Response: Check that the message received and the properties to be read are of the type expected.

MQJMS_EXCEPTION_MQ_NULL_Q

```
public final static java.lang.String
```

MQQueue reference is null.
Explanation: JMS attempted to perform some operation on a null MQQueue object.
User Response: Check your system setup, and that all required queue names have been specified. If this represents an internal error condition in JMS, contact your IBM representative.

MQJMS_EXCEPTION_MQ_NULL_QMGR

```
public final static java.lang.String
```

MQQueueManager reference is null.
Explanation: JMS attempted to perform an operation on a null MQQueueManager object.
User Response: Check that the relevant object has not been closed. If this represents an internal error condition in JMS, contact your IBM representative.

MQJMS_EXCEPTION_MQ_Q_CLOSE_FAILED

```
public final static java.lang.String
```

Failed to close WebSphere MQ queue.
Explanation: JMS attempted to close a WebSphere MQ queue, but encountered an error. The queue might already be closed, or another thread might be performing an MQGET while close() is called.
User Response: Use the linked exception to determine the cause of this error. You might be able to perform the close() later.

MQJMS_EXCEPTION_MQ_Q_INQUIRE_FAILED

```
public final static java.lang.String
```

Failed to inquire an MQQueue object depth.
Explanation: WebSphere MQ JMS is unable to determine how many messages are on the queue.
User Response: Check that the queue and queue manager are available.

MQJMS_EXCEPTION_MQ_Q_OPEN_FAILED

```
public final static java.lang.String
```

Failed to open an MQQueue object.
Explanation: JMS attempted to perform an MQOPEN, but WebSphere MQ reported an error.
User Response: Use the linked exception to determine the cause of this error. Check that the specified queue and queue manager are defined correctly.

MQJMS_EXCEPTION_MQ_QM_COMMIT_FAILED

```
public final static java.lang.String
```

MQQueueManager.commit() failed.
Explanation: JMS attempted to perform an MQCMIT, but WebSphere MQ reported an error.
User Response: Use the linked exception to determine the cause of this error.

MQJMS_EXCEPTION_MQ_UNKNOWN_DEFTYPE

```
public final static java.lang.String
```

Unknown value for an MQQueue object definitionType: {0}.
Explanation: Unable to delete the temporary queue because the definitionType is

not valid.
User Response: Check the setting of the definitionType.

MQJMS_EXCEPTION_MSG_CREATE_ERROR

```
public final static java.lang.String
```

Failed to create JMS message.
Explanation: The wrong message type or properties were specified when creating a base message.
User Response: Check the linked WebSphere MQ exception Reason and Completion code for more information.

MQJMS_EXCEPTION_NULL_ELEMENT_NAME

```
public final static java.lang.String
```

Element name is null.
Explanation: A null name string was passed to one of the get value by name methods of MapMessage.
User Response: Ensure that all name strings being used to retrieve values are not null.

MQJMS_EXCEPTION_NULL_PROPERTY_NAME

```
public final static java.lang.String
```

Property name is null.
Explanation: The itemExists() method of MapMessage was invoked with a null item name, or a null name string was used as an argument to a method which retrieves property values by name from a JMS message.
User Response: Ensure that the name strings indicated do not have null values.

MQJMS_EXCEPTION_PUT_MSG_FAILED

```
public final static java.lang.String
```

Failed to send message to an MQQueue object.
Explanation: JMS attempted to perform an MQPUT, but WebSphere MQ reported an error.
User Response: Use the linked exception to determine the cause of this error.

MQJMS_EXCEPTION_QMDISC_FAILED

```
public final static java.lang.String
```

Failed to disconnect queue manager.
Explanation: JMS encountered an error while attempting to disconnect.
User Response: Use the linked exception to determine the cause of this error.

MQJMS_EXCEPTION_QMGR_FAILED

```
public final static java.lang.String
```

Failed to create an MQQueueManager for {0}.
Explanation: JMS was unable to connect to a queue manager. {0} gives the name of the queue manager.
User Response: Use the linked exception to determine the cause of this error. Check the queue manager is running and. If using client attach, check that the listener is running and the channel, port and hostname are set correctly. If no queue manager name has been specified, check that the default queue manager has been defined.

MQJMS_EXCEPTION_RESOURCE_ALLOCATION

```
public final static java.lang.String
```

JMS Provider is unable to allocate the resources required for a method.
Explanation: Machine resources might be overloaded, the linked exception might give further information.
User Response: Check system resources and load.

MQJMS_EXCEPTION_SOME_PROBLEM

```
public final static java.lang.String
```

WebSphere MQ problem: {0}.
Explanation: JMS encountered a problem with WebSphere MQ. {0} describes the problem.
User Response: Use the included text and linked exception to determine the cause of this error.

MQJMS_EXCEPTION_TRANSACTION_IN_PROGRESS

```
public final static java.lang.String
```

Operation cannot be performed while a transaction is in progress.
User Response: Wait for the current transaction to complete. See the linked WebSphere MQ exception for further information.

MQJMS_EXCEPTION_TRANSACTION_ROLLED_BACK

```
public final static java.lang.String
```

Call to Session.commit() resulted in a rollback of the current transaction.
Explanation: The transaction failed and was rolled back to a safe state. See the linked exception for more information.

MQJMS_EXCEPTION_UNEXPECTED_ERROR

```
public final static java.lang.String
```

An internal error has occurred. Contact your system administrator.
Explanation: Internal Error.
User Response: Contact your IBM representative.

MQJMS_EXCEPTION_UNKNOWN_ACK_MODE

```
public final static java.lang.String
```

Unknown acknowledgement mode {0}.
Explanation: Incorrect or no parameter {0} set for acknowledgement mode on the session.
User Response: See the JMS specification for the possible values for the acknowledgement mode.

MQJMS_EXCEPTION_XACLOSE_FAILED

```
public final static java.lang.String
```

XACLOSE failed.
Explanation: See linked XAException for more details.

MQJMS_LOCAL_XA_CLASH

```
public final static java.lang.String
```

Local transactions not allowed with XA sessions.
Explanation: A call pertaining to a local transaction was made on a session involved with XA-coordinated transactions
User Response: This typically represents an error in an application server. Consult your application server's documentation and any error logs.

MQJMS_MESSAGECONSUMER_CLOSED

```
public final static java.lang.String
```

Message Consumer is closed.
Explanation: Either or both of the session and connection are closed.
User Response: Check to ensure that the session and connection are both available.

MQJMS_MESSAGEPRODUCER_CLOSED

```
public final static java.lang.String
```

Message Producer is closed.
Explanation: Either or both of the session and connection are closed.
User Response: Check to ensure that the session and connection are both available.

MQJMS_MSEL_AND_BVER_INCOMPATIBLE

```
public final static java.lang.String
```

Broker message selection is only valid when using WebSphere MQ Integrator broker.
Explanation: Broker version and message selection are not consistent.
User Response: Ensure the broker version has been set in the ConnectionFactory. Use the method ConnectionFactory.setBrokerVersion(JMSC.MQJMS_BROKER_V2) for a broker of WebSphere MQ Integrator, WebSphere MQ Event Broker, WebSphere Business Integration Event Broker, or WebSphere Business Integration Message Broker.

MQJMS_PS_COMMAND_MSG_BUILD

```
public final static java.lang.String
```

Failed to build command {0}.
Explanation: Broker message command parameters incorrect.
User Response: Check linked exception for cause.

MQJMS_PS_COMMAND_MSG_FAILED

```
public final static java.lang.String
```

Failed to publish command to WebSphere MQ queue.
Explanation: Invalid command, queue unavailable or broker errors.
User Response: Check linked exception reason and completion codes for more information.

MQJMS_PS_GENERAL_ERROR

```
public final static java.lang.String
```

Publish/Subscribe failed due to {0}.
Explanation: General error: {0} shows the reason.
User Response: Check the linked exception reason and completion codes for more information. It is possible that the broker and queue manager versions are incompatible.

MQJMS_PS_INCORRECT_SUBSTORE

```
public final static java.lang.String
```

Incorrect subscription store type.
Explanation: Subscription store changed within TopicConnection.
User Response: Contact your IBM representative.

MQJMS_PS_INVALID_SUBQ_PREFIX

```
public final static java.lang.String
```

Subscriber queue prefix was not valid: {0}.
Explanation: The name specified is not valid. It must begin with SYSTEM.JMS.D for durable subscriptions or SYSTEM.JMS.ND for nondurable subscriptions. The name specified must end with an asterisk ('*').

MQJMS_PS_NULL_CLIENTID

```
public final static java.lang.String
```

The use of an uninitialized client ID is not possible.
Explanation: The client ID in the connection has not been set.
User Response: Set the clientID before attempting to perform any operation.

MQJMS_PS_NULL_NAME

```
public final static java.lang.String
```

The use of a null name is not possible.
Explanation: Durable connection consumers must be named.
User Response: Check for null values.

MQJMS_PS_PUBLISH_MSG_BUILD

```
public final static java.lang.String
```

Failed to build publish message.
Explanation: Unable to build the base message for the broker.
User Response: See the linked exception for further details. Check settings and parameters are all correct.

MQJMS_PS_PUBLISH_MSG_FAILED

```
public final static java.lang.String
```

Failed to publish message to WebSphere MQ queue.
Explanation: See linked exception for more information.
User Response: Check settings and parameters are all correct.

MQJMS_PS_STORE_ADMIN_ENTRY

```
public final static java.lang.String
```

Failed to store administration entry.
Explanation: An add to the admin or status queue failed due to duplication or some other error. See linked exception for more information.
User Response: Check for duplicates and retry.

MQJMS_PS_SUB_ACTIVE

```
public final static java.lang.String
```

Subscription has an active TopicSubscriber.
Explanation: Can be caused by a problem opening a queue or if a subscription already exists on the JVM. If running in WebSphere Application Server there can be other causes. See linked exception, if set, for more information.
User Response: Check settings.

MQJMS_PS_SUB_Q_DELETE_FAILED

```
public final static java.lang.String
```

Failed to delete subscriber queue {0}.
Explanation: {0} gives the queue name. See linked exception for more information.
User Response: See "Writing WebSphere MQ JMS publish/subscribe applications" for more information.

MQJMS_PS_SUB_Q_OPEN_FAILED

```
public final static java.lang.String
```

Failed to open subscriber queue {0}.
User Response: See linked exception for more information.

MQJMS_PS_SUBQ_REQUEUE

```
public final static java.lang.String
```

Durable re-subscribe must use same subscriber queue; specified: {0}, original: {1}.
Explanation: {0} and {1} show the differing queue names. Unable to get a subscription due to wrong queue manager or queue.
User Response: Check settings.

MQJMS_PS_SUBSTORE_NOT_SUPPORTED

```
public final static java.lang.String
```

Subscription store type not supported by queue manager.
Explanation: Deferred messages not supported by queue manager or broker is to low a version.
User Response: Possible incompatibility between queue manager version and broker. Specify a different type of subscription store or upgrade the queue manager or broker.

MQJMS_PS_TOPIC_NULL

```
public final static java.lang.String
```

Topic reference is null.
Explanation: Topic supplied to a publisher is null.
User Response: Use non-null values.

MQJMS_PS_UNKNOWN_DS

```
public final static java.lang.String
```

Unknown durable subscription {0}.
Explanation: Unable to locate the given subscription. For example, during an unsubscribe request.
User Response: Check the subscriber name specified and retry.

MQJMS_PS_WRONG_SUBSCRIPTION_TYPE

```
public final static java.lang.String
```

Incorrect subscription type for this subscription store.
Explanation: TopicSubscriber was created with a different SUBSTORE setting than current TopicConnection.
User Response: Ensure TopicSubscribers are only used during the lifetime of their parent TopicConnection.

MQJMS_PUBLISHER_CLOSED

```
public final static java.lang.String
```

TopicPublisher is closed.
User Response: Open or reopen the topic publisher if required.

MQJMS_QRECEIVER_CLOSED

```
public final static java.lang.String
```

QueueReceiver is closed.
User Response: Open or reopen the receiver.

MQJMS_SUBSCRIBER_CLOSED

```
public final static java.lang.String
```

TopicSubscriber is closed.
User Response: Open or reopen the TopicSubscriber.

MQJMS_UTIL_PS_NO_BRK_Q

```
public final static java.lang.String
```

Unable to access the broker control queue on the queue manager.
User Response: Check that the control queue exists. The default name is SYSTEM.BROKER.CONTROL.QUEUE.

MQJMS_UTIL_PS_NO_BROKER

```
public final static java.lang.String
```

No broker response.
Explanation: Possible causes are: Broker is not running. You are using a BrokerVersion of v2 in your TopicConnectionFactory with the WebSphere MQ Publish/Subscribe broker, which does not support this. The Broker has rejected the publication or subscription and placed it on the SYSTEM.DEAD.LETTER.QUEUE
User Response: Ensure that your broker is running. Check the system event log for broker error messages. Check that the broker supports the BrokerVersion you are using. Check the SYSTEM.DEAD.LETTER.QUEUE for rejected messages.

MQJMS_UTIL_PS_NO_MSG

```
public final static java.lang.String
```

The broker appears to be running, but the message did not arrive.
Explanation: Generated by the installation verification test when the subscriber fails to receive the published message.
User Response: Check that you have set up the broker correctly. Check system event logs for broker error messages. Check the SYSTEM.DEAD.LETTER.QUEUE for messages rejected by the broker.

MQJMS_UTIL_PS_NO_QM

```
public final static java.lang.String
```

Unable to connect to queue manager.
Explanation: Generated by the installation verification test.
User Response: Check that the queue manager is running and that its name is specified correctly in the IVTTest parameters.

Part 6. Appendixes

Appendix A. Mapping between administration tool properties and programmable properties

WebSphere MQ classes for Java Message Service provides facilities to set and query the properties of administered objects either using the WebSphere MQ JMS administration tool or in an application. Table 33 lists the name of each property, as used in the administration tool, and the set method that is used to set the value of the property in an application. The table also shows the mapping between symbolic property values used in the tool and their programmable equivalents.

Table 33. Comparison of representations of property values within the administration tool and within applications

Property	Set method	Tool property values	Program property values
BROKERCCDSUBQ	setBrokerCCDurSubQueue		
BROKERCCSUBQ	setBrokerCCSubQueue		
BROKERCONQ	setBrokerControlQueue		
BROKERDURSUBQ	setBrokerDurSubQueue		
BROKERPUBQ	setBrokerPubQueue		
BROKERPUBQMGR	setBrokerPubQueueManager		
BROKERQMGR	setBrokerQueueManager		
BROKERSUBQ	setBrokerSubQueue		
BROKERVER	setBrokerVersion	V1 V2	JMSC.MQJMS_BROKER_V1 JMSC.MQJMS_BROKER_V2
CCDTURL	setCCDTURL		
CCSID	setCCSID		
CHANNEL	setChannel		
CLEANUP	setCleanupLevel	NONE SAFE STRONG ASPROP	JMSC.MQJMS_CLEANUP_NONE JMSC.MQJMS_CLEANUP_SAFE JMSC.MQJMS_CLEANUP_STRONG JMSC.MQJMS_CLEANUP_AS_PROPERTY
CLEANUPINT	setCleanupInterval		
CLIENTID	setClientId		
CLONESUPP	setCloneSupport	DISABLED ENABLED	JMSC.MQJMS_CLONE_DISABLED JMSC.MQJMS_CLONE_ENABLED
COMPHDR	setHdrCompList	NONE SYSTEM	JMSC.MQJMS_COMPHDR_NONE JMSC.MQJMS_COMPHDR_SYSTEM

Table 33. Comparison of representations of property values within the administration tool and within applications (continued)

Property	Set method	Tool property values	Program property values
COMPMSG	setMsgCompList	NONE RLE ZLIBFAST ZLIBHIGH	JMSC.MQJMS_COMPMSG_NONE JMSC.MQJMS_COMPMSG_RLE JMSC.MQJMS_COMPMSG_ZLIBFAST JMSC.MQJMS_COMPMSG_ZLIBHIGH
CONNOPT	setMQConnectionOptions	STANDARD SHARED ISOLATED FASTPATH SERIALQM SERIALQSG RESTRICTQM RESTRICTQSG	JMSC.MQCNO_STANDARD_BINDING JMSC.MQCNO_SHARED_BINDING JMSC.MQCNO_ISOLATED_BINDING JMSC.MQCNO_FASTPATH_BINDING JMSC.MQCNO_SERIALIZE_CONN_TAG_Q_MGR JMSC.MQCNO_SERIALIZE_CONN_TAG_QSG JMSC.MQCNO_RESTRICT_CONN_TAG_Q_MGR JMSC.MQCNO_RESTRICT_CONN_TAG_QSG
CONNTAG	setConnTag		
DESCRIPTION	setDescription		
DIRECTAUTH	setDirectAuth	BASIC CERTIFICATE	AJMSC.MQJMS_DIRECTAUTH_BASIC JMSC.MQJMS_DIRECTAUTH_CERTIFICATE
ENCODING	setEncoding		
EXPIRY	setExpiry	APP UNLIM	JMSC.MQJMS_EXP_APP JMSC.MQJMS_EXP_UNLIMITED
FAILIFQUIESCE	setFailIfQuiesce	YES NO	JMSC.MQJMS_FIQ_YES JMSC.MQJMS_FIQ_NO
HOSTNAME	setHostName		
LOCALADDRESS	setLocalAddress		
MAPNAMESTYLE	setMapNameStyle	STANDARD COMPATIBLE	JMSC.MAP_NAME_STYLE_STANDARD JMSC.MAP_NAME_STYLE_COMPATIBLE
MAXBUFFSIZE	setMaxBufferSize		
MSGBATCHSZ	setMsgBatchSize		
MSGRETENTION	setMessageRetention	YES NO	JMSC.MQJMS_MRET_YES JMSC.MQJMS_MRET_NO
MSGSELECTION	setMessageSelection	CLIENT BROKER	JMSC.MQJMS_MSEL_CLIENT JMSC.MQJMS_MSEL_BROKER

Table 33. Comparison of representations of property values within the administration tool and within applications (continued)

Property	Set method	Tool property values	Program property values
MULTICAST	setMulticast	DISABLED ASCF ENABLED RELIABLE NOTR	JMSC.MQJMS_MULTICAST_DISABLED JMSC.MQJMS_MULTICAST_AS_CF JMSC.MQJMS_MULTICAST_ENABLED JMSC.MQJMS_MULTICAST_RELIABLE JMSC.MQJMS_MULTICAST_NOT_RELIABLE
OPTIMISTICPUBLICATION	setOptimisticPublication	NO YES	false true
OUTCOMENOTIFICATION	setOutcomeNotification	YES NO	true false
PERSISTENCE	setPersistence	APP QDEF PERS NON HIGH	JMSC.MQJMS_PER_APP JMSC.MQJMS_PER_QDEF JMSC.MQJMS_PER_PER JMSC.MQJMS_PER_NON JMSC.MQJMS_PER_NPHIGH
POLLINGINT	setPollingInterval		
PORT	setPort		
PRIORITY	setPriority	APP QDEF	JMSC.MQJMS_PRI_APP JMSC.MQJMS_PRI_QDEF
PROCESSDURATION	setProcessDuration	UNKNOWN SHORT	JMSC.MQJMS_PROCESSING_UNKNOWN JMSC.MQJMS_PROCESSING_SHORT
PROXYHOSTNAME	setProxyHostName		
PROXYPORT	setProxyPort		
PUBACKINT	setPubAckInterval		
QMANAGER	setQueueManager		
RECEIVEISOLATION	setReceiveIsolation	COMMITED UNCOMMITTED	JMSC.MQJMS_RCVISOL_COMMITTED JMSC.MQJMS_RCVISOL_UNCOMMITTED
RECEXIT	setReceiveExit		
RECEXITINIT	setReceiveExitInit		
RESCANINT	setRescanInterval		
SECEXIT	setSecurityExit		
SECEXITINIT	setSecurityExitInit		
SENDEXIT	setSendExit		
SENDEXITINIT	setSendExitInit		
SPARSESUBS	setSparseSubscriptions	YES NO	true false
SSLCIPHERSUITE	setSSLCipherSuite		

Table 33. Comparison of representations of property values within the administration tool and within applications (continued)

Property	Set method	Tool property values	Program property values
SSLCRL	setSSLCertStores		
SSLFIPSREQUIRED	setSSLFipsRequired	NO YES	false true
SSLPEERNAME	setSSLPeerName		
SSLRESETCOUNT	setSSLResetCount		
STATREFRESHINT	setStatusRefreshInterval		
SUBSTORE	setSubscriptionStore	MIGRATE QUEUE BROKER	JMSC.MQJMS_SUBSTORE_MIGRATE JMSC.MQJMS_SUBSTORE_QUEUE JMSC.MQJMS_SUBSTORE_BROKER
SYNCPOINTALLGETS	setSyncpointAllGets		
TARGCLIENT	setTargetClient	JMS MQ	JMSC.MQJMS_CLIENT_JMS_COMPLIANT JMSC.MQJMS_CLIENT_NONJMS_MQ
TARGCLIENTMATCHING	setTargClientMatching	YES NO	true false
TEMPMODEL	setTemporaryModel		
TEMPQPREFIX	setTempQPrefix		
TRANSPORT	setTransportType	BIND CLIENT DIRECT DIRECTHTTP	JMSC.MQJMS_TP_BINDINGS_MQ JMSC.MQJMS_TP_CLIENT_MQ_TCPIP JMSC.MQJMS_TP_DIRECT_TCPIP JMSC.MQJMS_TP_DIRECT_HTTP
USECONNPOOLING	setUseConnectionPooling		

Appendix B. Scripts provided with WebSphere MQ classes for Java Message Service

The following files are provided in the `bin` directory of your WebSphere MQ JMS installation. These scripts are provided to assist with common tasks that need to be performed while installing or using WebSphere MQ JMS. Table 34 lists the scripts and their uses.

Table 34. Utilities supplied with WebSphere MQ classes for Java Message Service

Utility	Use
Cleanup[1]	Runs the subscription cleanup utility as described in "Manual cleanup" on page 346, or the consumer cleanup utility as described in "Manual cleanup" on page 369..
DefaultConfiguration	Runs the default configuration application on non-Windows systems as described in "JMS Postcard configuration" on page 20.
formatLog[1]	Converts binary log files to plain text as described in "Logging" on page 34.
IVTRun[1] IVTTidy[1] IVTSetup[1]	Runs the point-to-point installation verification test program as described in "Running the point-to-point IVT" on page 26.
JMSAdmin[1]	Runs the administration tool as described in Chapter 5, "Using the WebSphere MQ JMS administration tool," on page 35.
JMSAdmin.config	Configuration file for the administration tool as described in "Configuration" on page 36.
postcard[1]	Starts the JMS Postcard application as described in "JMS Postcard" on page 17.
PSIVTRun[1]	Runs the publish/subscribe installation verification test program as described in "The publish/subscribe installation verification test" on page 30.
PSReportDump.class	Views broker report messages as described in "Handling broker reports" on page 347. For information specific to JMS 1.1, see "Handling broker reports" on page 371.
setjmsenv	Sets the environment variables on a UNIX system as described in "Environment variables" on page 8.
Note: 1. On Windows, the file name has the extension .bat .	

Scripts provided with WebSphere MQ JMS

Appendix C. Connecting to other products

This section covers:

- How to configure a publish/subscribe broker for a connection from WebSphere MQ JMS in "Setting up a publish/subscribe broker"
- How to use WebSphere MQ Integrator V2 to route or transform messages sent to or from a JMS client in "Transformation and routing with WebSphere MQ Integrator V2" on page 641
- How to configure WebSphere MQ JMS for a direct connection to WebSphere Business Integration Event Broker, Version 5.0 or WebSphere Business Integration Message Broker, Version 5.0 in "Configuring WebSphere MQ JMS for a direct connection to WebSphere Business Integration Event Broker, Version 5.0 or later and WebSphere Business Integration Message Broker, Version 5.0 or later" on page 642.

Setting up a publish/subscribe broker

You can use WebSphere MQ Integrator Version 2, WebSphere MQ Event Broker Version 2.1, WebSphere Business Integration Event Broker, Version 5.0, or WebSphere Business Integration Message Broker, Version 5.0 as the publish/subscribe broker for WebSphere MQ JMS. You can link to each of these brokers across a connection to base WebSphere MQ, or you can connect directly to WebSphere MQ Event Broker, WebSphere Business Integration Event Broker, or WebSphere Business Integration Message Broker over TCP/IP. Each method requires some setup activities:

Linking across WebSphere MQ

- Base WebSphere MQ

  First, create a broker publication queue. This is a WebSphere MQ queue on the broker queue manager; it is used to submit publications to the broker. You can choose your own name for this queue, but it must match the queue name in your TopicConnectionFactory's BROKERPUBQ property. By default, a TopicConnectionFactory's BROKERPUBQ property is set to the value SYSTEM.BROKER.DEFAULT.STREAM so, unless you want to configure a different name in the TopicConnectionFactory, name the queueSYSTEM.BROKER.DEFAULT.STREAM.

- WebSphere MQ Integrator V2

  The next step is to set up a *message flow* within an execution group for the broker. The purpose of this message flow is to read messages from the broker publication queue. (If you want, you can set up multiple publication queues; each needs its own TopicConnectionFactory and message flow.)

  The basic message flow consists of an MQInput node (configured to read from the SYSTEM.BROKER.DEFAULT.STREAM queue) whose output is connected to the input of a Publication (or MQOutput) node.

  The message flow diagram therefore looks similar to the following:

Figure 5. WebSphere MQ Integrator message flow

When this message flow is deployed and the broker is started, from the JMS application's perspective the WebSphere MQ Integrator V2 broker behaves like an WebSphere MQ Publish/Subscribe broker. The current subscription state can be viewed using the WebSphere MQ Integrator Control Center.

Note:

1. No modifications are required to WebSphere MQ classes for Java Message Service.
2. WebSphere MQ Publish/Subscribe and WebSphere MQ Integrator V2 brokers cannot coexist on the same queue manager.
3. Details of the WebSphere MQ Integrator V2 installation and setup procedure are described in the *WebSphere MQ Integrator for Windows NT Version 2.0 Installation Guide*.

Direct connection to WebSphere MQ Event Broker, Version 2.1 over TCP/IP

For this, set up a message flow within an execution group on WebSphere MQ Event Broker. This message flow is to read messages from the TCP/IP socket on which the broker is listening.

The basic message flow consists of a JMSIPOptimised flow set to listen on the port configured for direct connections. By default, this port is 1506.

Note: WebSphere MQ Event Broker can be configured to listen for both direct connections across TCP/IP from WebSphere MQ JMS and connections made across TCP/IP through WebSphere MQ. In this case, the two listeners must be configured on different ports. The default port for a WebSphere MQ connection is 1414.

Direct connection to WebSphere Business Integration Event Broker, Version 5.0 or WebSphere Business Integration Message Broker, Version 5.0

To configure a WebSphere Business Integration Event Broker or WebSphere Business Integration Message Broker broker for a direct connection from WebSphere MQ JMS, create a message flow to read messages from the TCP/IP port on which the broker is listening and publish the messages. You can do this in either of the following ways:

- You can create a message flow that contains a Real-timeOptimizedFlow message processing node.
- You can create a message flow that contains a Real-timeInput message processing node and a Publication message processing node.

You must configure the Real-timeOptimizedFlow or Real-timeInput node to listen on the port used for direct connections. By default, the port number for direct connections is 1506.

Transformation and routing with WebSphere MQ Integrator V2

You can use WebSphere MQ Integrator V2 to route or transform messages that are created by a JMS client application, and to send or publish messages to a JMS client.

The WebSphere MQ JMS implementation uses the mcd folder of the MQRFH2 to carry information about the message, as described in "The MQRFH2 header" on page 384. By default, the Message Domain (Msd) property is used to identify whether the message is a text, bytes, stream, map, or object message. This value is set depending on the type of the JMS message.

If the application calls setJMSType, it can set the mcd type field to a value of its choosing. This type field can be read by the WebSphere MQ Integrator message flow, and a receiving JMS application can use the getJMSType method to retrieve its value. This applies to all kinds of JMS message.

When a JMS application creates a text or bytes message, the application can set mcd folder fields explicitly by calling the setJMSType method and passing in a string argument in a special URI format as follows:

`mcd://domain/[set]/[type][?format=fmt]`

This URI form allows an application to set the mcd to a domain that is not one of the standard jms_xxxx values; for example, to domain mrm. It also allows the application to set any or all of the mcd set, type, and format fields.

The string argument to setJMSType is interpreted as follows:

1. If the string does not appear to be in the special URI format (it does not start with mcd://), the string is added to the mcd folder as the `type` field.
2. If the string starts with mcd://, conforms to the URI format, *and* the message is a Text or Bytes message, the URI string is split into its constituent parts. The domain part overrides the jms_text or jms_bytes value that would otherwise have been generated, and the remaining parts (if present) are used to set the set, type, and format fields in the mcd. Note that set, type, and format are all optional.
3. If the string starts with mcd:// and the message is a Map, Stream, or Object message, the setJMSType call throws an exception. So you cannot override the domain, or provide a set or format for these classes of message, but you can provide a type.

When a WebSphere MQ message is received with an Msd domain other than one of the standard jms_xxxx values, it is instantiated as a JMS text or bytes message and a URI-style JMSType is assigned to it. If the format field of the RFH2 is MQFMT_STRING, it becomes a TextMessage; otherwise it becomes a BytesMessage. The receiving application can read this using the getJMSType method.

Configuring WebSphere MQ JMS for a direct connection to WebSphere Business Integration Event Broker, Version 5.0 or later and WebSphere Business Integration Message Broker, Version 5.0 or later

A WebSphere MQ JMS client can connect directly to a WebSphere Business Integration Event Broker or WebSphere Business Integration Message Broker broker over TCP/IP. The available function is comparable to that provided for a direct connection to a WebSphere MQ Event Broker, Version 2.1 broker, but with the following additions:

- Secure Sockets Layer (SSL) authentication
- Multicast
- HTTP tunnelling
- Connect via proxy

For detailed information about this additional function, see the WebSphere Business Integration Event Broker or WebSphere Business Integration Message Broker Information Center. The following sections explain how to configure a WebSphere MQ JMS client in order to use this function.

Secure Sockets Layer (SSL) authentication

You can use SSL authentication when a WebSphere MQ JMS client connects directly to a WebSphere Business Integration Event Broker or WebSphere Business Integration Message Broker broker. Only SSL authentication is supported for this type of connection. SSL cannot be used to encrypt or decrypt message data that flows between the WebSphere MQ JMS client and the broker or to perform integrity checks on the data.

Note the difference between this situation and that when a WebSphere MQ JMS client connects to a WebSphere MQ queue manager. In the latter case, the WebSphere MQ SSL support can be used to encrypt and decrypt message data that flows between the client and the queue manager and to perform integrity checks on the data, as well as providing authentication.

If you want to protect message data on a direct connection to a broker, you can use function in the broker instead. You can assign a quality of protection (QoP) value to each topic whose associated messages you want to protect. This allows you to select a different level of message protection for each topic.

If client authentication is required, a WebSphere MQ JMS client can use the same digital certificate for connecting directly to a broker as it does for connecting to a WebSphere MQ queue manager.

You can configure a WebSphere MQ JMS client to use SSL authentication in either of the following ways:

- In a WebSphere MQ JMS application, use the setDirectAuth() method of an MQConnectionFactory or MQTopicConnectionFactory object to set the direct authentication attribute to JMSC.MQJMS_DIRECTAUTH_CERTIFICATE.
- Use the WebSphere MQ JMS administration tool to set the DIRECTAUTH property to CERTIFICATE.

Note:

1. If the TRANSPORT property is set to DIRECT, then it is the DIRECTAUTH property, not the SSLCIPHERSUITE property, that determines whether SSL authentication is used.
2. If the DIRECTAUTH property is set to CERTIFICATE, the SSLPEERNAME and SSLCRL properties are used to perform the same checks as those performed when a WebSphere MQ JMS client connects to a WebSphere MQ queue manager using the WebSphere MQ SSL support.
3. The Java Secure Socket Extension (JSSE) keystore and truststore configurations determine which client certificate is used for authentication, and whether a server certificate is trusted, in the same way that they do when a WebSphere MQ JMS client connects to a WebSphere MQ queue manager using the WebSphere MQ SSL support.

Multicast

You can configure a WebSphere MQ JMS client multicast connection to a broker in either of the following ways:

- In a WebSphere MQ JMS application, use the setMulticast() method of an MQConnectionFactory, MQTopicConnectionFactory, or MQTopic object to set the multicast attribute.
- Use the WebSphere MQ JMS administration tool to set the MULTICAST property.

The TRANSPORT property must be set to DIRECT before the MULTICAST property has any effect.

WebSphere MQ includes support for subscribing to multicast-enabled topics using a direct connection to a broker where the multicast protocol defined on the broker is set to PGM/IP or UDP-encapsulated PGM. When PGM/IP is used, the client requires a native library to be present on the system path. It is installed into the directory specified by MQ_JAVA_LIB_PATH

This library has different names on different platforms, as follows:

- AIX - libPgmIpLayer.so (32-bit only)
- HP-UX 11i v1- libPgmIpLayer.sl (32-bit only)
- Linux (x86 platform) - libPgmIpLayer.so
- Linux (zSeries platform) - libPgmIpLayer.so (32-bit only)
- Solaris SPARC - libPgmIpLayer.so (32-bit only)
- Windows - PgmIpLayer.dll
- z/OS - libPgmIpLayer.so

PGM/IP can be used only on the platforms listed above. An error message, MQJMS_DIR_PGM_LIB_NOT_FOUND, is thrown if the library is not present or can not be found.

HTTP tunnelling

A WebSphere MQ JMS client can connect to a broker using HTTP tunnelling. HTTP tunnelling is suitable for applets because the Java 2 Security Manager normally rejects any attempt by an applet to connect directly to the broker. Using HTTP tunnelling, which exploits the built in support in Web browsers, a WebSphere MQ JMS client can connect to the broker using the HTTP protocol as though connecting to a Web site.

You can configure a WebSphere MQ JMS client to use HTTP tunnelling in either of the following ways:

- In a WebSphere MQ JMS application, use the setTransportType() method of an MQConnectionFactory object to set the transport type attribute to JMSC.MQJMS_TP_DIRECT_HTTP.
- Use the WebSphere MQ JMS administration tool to set the TRANSPORT property to DIRECTHTTP.

SSL authentication cannot be used with HTTP tunnelling.

Connect via proxy

A WebSphere MQ JMS client can connect to a broker through a proxy server. The client connects directly to the proxy server and uses the Internet protocol defined in RFC 2817 to ask the proxy server to forward the connection request to the broker. This option does not work for applets because the Java 2 Security Manager normally rejects any attempt by an applet to connect directly to a proxy server.

You can configure a WebSphere MQ JMS client to connect to a broker through a proxy server in either of the following ways:

- In a WebSphere MQ JMS application, use the setProxyHostName() and setProxyPort() methods of an MQConnectionFactory or MQTopicConnectionFactory object to set the proxy host name and proxy port attributes.
- Use the WebSphere MQ JMS administration tool to set the PROXYHOSTNAME and PROXYPORT properties.

If the TRANSPORT property is set to DIRECT, the type of connection to the broker depends on the PROXYHOSTNAME property according to the following rules:

- If the PROXYHOSTNAME property is set to the empty string, the WebSphere MQ JMS client connects directly to the broker using the HOSTNAME and PORT properties to locate the broker.
- If the PROXYHOSTNAME property is set to a value other than the empty string, the WebSphere MQ JMS client connects to the broker through the proxy server identified by the PROXYHOSTNAME and PROXYPORT properties.

Appendix D. SSL CipherSpecs and CipherSuites

Table 35 lists the CipherSpecs supported by WebSphere MQ and their equivalent CipherSuites. The table also indicates whether a WebSphere MQ Java application can establish a connection to a queue manager if a CipherSpec is specified at the server end of the MQI channel and the equivalent CipherSuite is specified at the client end.

For each combination of CipherSpec and CipherSuite, whether a WebSphere MQ base Java application can connect to a queue manager depends on the value of the sslFipsRequired field in the MQEnvironment class, or on the value of the environment property MQC.SSL_FIPS_REQUIRED_PROPERTY. Similarly, whether a WebSphere MQ JMS application can connect to a queue manager depends on the value of the SSLFIPSREQUIRED property of the ConnectionFactory object.

At the server end of an MQI channel, the name of a CipherSpec can be specified as the value of the SSLCIPH parameter on a DEFINE CHANNEL CHLTYPE(SVRCONN) command. At the client end of an MQI channel, the name of a CipherSuite can be specified in the following ways:

- A WebSphere MQ base Java application can set the sslCipherSuite field in the MQEnvironment class, or set the environment property MQC.SSL_CIPHER_SUITE_PROPERTY.
- A WebSphere MQ JMS application can call the setSSLCipherSuite() method of a ConnectionFactory object.
- Using the WebSphere MQ JMS administration tool, you can set the SSLCIPHERSUITE property of a ConnectionFactory object.

Table 35. CipherSpecs supported by WebSphere MQ and their equivalent CipherSuites

CipherSpec	Equivalent CipherSuite	Connection possible if SFIPS[1] is set to NO?	Connection possible if SFIPS[1] is set to YES?
NULL_MD5	SSL_RSA_WITH_NULL_MD5	Yes	No
NULL_SHA	SSL_RSA_WITH_NULL_SHA	Yes	No
RC4_MD5_EXPORT	SSL_RSA_EXPORT_WITH_RC4_40_MD5	Yes	No
RC4_MD5_US	SSL_RSA_WITH_RC4_128_MD5	Yes	No
RC4_SHA_US	SSL_RSA_WITH_RC4_128_SHA	Yes	No
RC2_MD5_EXPORT	SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5	Yes	No
DES_SHA_EXPORT	SSL_RSA_WITH_DES_CBC_SHA	Yes	No
RC4_56_SHA_EXPORT1024	SSL_RSA_EXPORT1024_WITH_RC4_56_SHA	No	No
DES_SHA_EXPORT1024	SSL_RSA_EXPORT1024_WITH_DES_CBC_SHA	No	No
TRIPLE_DES_SHA_US	SSL_RSA_WITH_3DES_EDE_CBC_SHA	Yes	No
TLS_RSA_WITH_AES_128_CBC_SHA	SSL_RSA_WITH_AES_128_CBC_SHA	No	Yes
TLS_RSA_WITH_AES_256_CBC_SHA	SSL_RSA_WITH_AES_256_CBC_SHA	No	Yes
AES_SHA_US[2]			
TLS_RSA_WITH_DES_CBC_SHA	SSL_RSA_WITH_DES_CBC_SHA	No	Yes
TLS_RSA_WITH_3DES_EDE_CBC_SHA	SSL_RSA_WITH_3DES_EDE_CBC_SHA	No	Yes

Table 35. CipherSpecs supported by WebSphere MQ and their equivalent CipherSuites (continued)

CipherSpec	Equivalent CipherSuite	Connection possible if SFIPS[1] is set to NO?	Connection possible if SFIPS[1] is set to YES?
FIPS_WITH_DES_CBC_SHA	SSL_RSA_FIPS_WITH_DES_CBC_SHA	Yes	No
FIPS_WITH_3DES_EDE_CBC_SHA	SSL_RSA_FIPS_WITH_3DES_EDE_CBC_SHA	Yes	No

Notes:

1. When using the WebSphere MQ JMS administration tool, SFIPS is the short name of the ConnectionFactory property SSLFIPSREQUIRED. In a WebSphere MQ base Java application, setting the sslFipsRequired field in the MQEnvironment class to `false` is equivalent to setting SSLFIPSREQUIRED to NO, and setting the sslFipsRequired field to `true` is equivalent to setting SSLFIPSREQUIRED to YES. Alternatively, a WebSphere MQ base Java application can set the environment property MQC.SSL_FIPS_REQUIRED_PROPERTY.
2. This CipherSpec has no equivalent CipherSuite.

Appendix E. Support for OSGi

OSGi provides a general purpose, secure, and managed Java framework, which supports the deployment of applications that come in the form of bundles. OSGi compliant devices can download and install bundles, and remove them when they are no longer required. The framework manages the installation and update of bundles in a dynamic and scalable fashion.

WebSphere MQ Java includes the following three OSGi bundles. The bundles are in the java/lib/OSGI subdirectory of your WebSphere MQ installation, or the Java\lib\OSGI folder on Windows. *version* is the WebSphere MQ version number, for example 6.0.2.0.

com.ibm.mq.osgi.client_*version*.**jar**
: Provides versions of WebSphere MQ base Java and WebSphere MQ JMS that are able to communicate with WebSphere MQ using a TCP/IP connection

com.ibm.mq.osgi.directip_*version*.**jar**
: Contains JAR files to allow the client bundle, com.ibm.mq.osgi.client_*version*.jar, to create a direct connection to a broker

com.ibm.mq.osgi.prereq_*version*.**jar**
: Contains prerequisite JAR files required by the client bundle

These bundles have been written to the OSGi Release 4 specification. They do not work in an OSGi Release 3 environment.

You must set your system path or library path correctly so that the OSGi runtime environment can find any required DLL files or shared libraries.

If you use the OSGi bundles for WebSphere MQ Java, temporary topics do not work. In addition, channel exit classes written in Java are not supported because of an inherent problem in loading classes in a multiple class loader environment such as OSGi. A user bundle can be aware of the WebSphere MQ Java bundles, but the WebSphere MQ Java bundles are not aware of any user bundle. As a result, the class loader used in a WebSphere MQ Java bundle cannot load a channel exit class that is in a user bundle.

For more information about OSGi, see the OSGi Alliance Web site at http://www.osgi.org.

Appendix F. The WebSphere MQ resource adapter

The J2EE Connector Architecture (JCA) provides a standard way of connecting applications running in a J2EE environment to an Enterprise Information System (EIS) such as WebSphere MQ or DB2. The WebSphere MQ resource adapter implements the JCA 1.5 interfaces, and allows JMS applications and message driven beans (MDBs), running in an application server, to access the resources of a WebSphere MQ queue manager. The resource adapter supports both the point-to-point domain and the publish/subscribe domain.

The WebSphere MQ resource adapter supports two types of communication between an application and a queue manager:

Outbound communication
An application starts a connection to a queue manager, and then sends JMS messages to JMS destinations and receives JMS messages from JMS destinations in a synchronous manner.

Inbound communication
A JMS message arriving at a JMS destination is delivered to an MDB, which processes the message asynchronously.

The WebSphere MQ resource adapter is supported on all WebSphere MQ Version 6.0 platforms except z/OS. You can install it on any application server that is certified as compliant with the J2EE 1.4 specification. Using the resource adapter, an application can connect to a WebSphere MQ Version 6.0 queue manager in either client mode or bindings mode, or to a WebSphere MQ Version 5.3 queue manager in client mode only.

This appendix contains the following sections:

Other required documentation

Every application server provides its own set of administration interfaces. Some application servers provide graphical user interfaces to define JCA resources, but others require the administrator to write XML deployment plans. It is therefore beyond the scope of this documentation to provide information about how to configure the WebSphere MQ resource adapter for each application server. This documentation focuses only on what you need to configure, and you must refer to your application server's own documentation for information about how to configure a JCA resource adapter.

To understand this documentation, you must be familiar with JMS and WebSphere MQ JMS, as described in the chapters from Chapter 10, "Writing WebSphere MQ

JMS applications," on page 313 through to Chapter 14, "WebSphere MQ JMS Application Server Facilities," on page 399. Many of the properties used to configure the WebSphere MQ resource adapter are equivalent to properties of WebSphere MQ JMS objects and have the same function.

Installation of the WebSphere MQ resource adapter

The WebSphere MQ resource adapter is supplied as a resource archive (RAR) file called wmq.jmsra.rar. This file is installed with WebSphere MQ Java in the directory shown in Table 36.

Table 36. The directory containing wmq.jmsra.rar for each platform

Platform	Directory
AIX	/usr/mqm/java/lib/jca
HP-UX, Linux, and Solaris	/opt/mqm/java/lib/jca
i5/OS	/QIBM/ProdData/mqm/java/lib/jca
Windows	*install_dir*\Java\lib\jca
Note: *install_dir* is the directory where you installed the WebSphere MQ server or WebSphere MQ client. The default directory is C:\Program Files\IBM\WebSphere MQ, but you might have chosen a different directory.	

The RAR file contains the following Java archive (JAR) files:

- com.ibm.mq.jar, which contains WebSphere MQ base Java
- com.ibm.mqjms.jar, which contains WebSphere MQ JMS
- mqconnector.jar, which contains the WebSphere MQ implementation of the JCA interfaces
- dhbcore.jar, which is an additional library required by WebSphere MQ JMS

You must install the WebSphere MQ resource adapter RAR file in your application server, but the way you do this depends on the application server. See the documentation for your application server for information about how to install a resource adapter RAR file.

For non-transacted client connections, no other files are required. For bindings connections, WebSphere MQ Java must be installed. On UNIX systems, you must also ensure that the Java Native Interface (JNI) libraries are in the system path. See Table 6 on page 10 for the location of the WebSphere MQ Java libraries, which include the JNI libraries. On Windows, the WebSphere MQ Java libraries are added to the system path automatically during installation of WebSphere MQ Java.

Distributed transactions are supported by default in bindings mode but, in client mode, they are supported only in the following cases:

- If you are using WebSphere Application Server, Version 6.0
- For any other application server, if the extended transactional client JAR file, com.ibm.mqetclient.jar, is in the class path

Table 37 on page 651 summarizes the support for non-transacted and transacted connections. For an explanation of client and bindings modes, see "Connection options" on page 3.

Table 37. Support for non-transacted and transacted connections

Type of connection	Non-transacted connections	Transacted connections
Client mode	Supported by default	Supported if you are using WebSphere Application Server, Version 6.0 or if com.ibm.mqetclient.jar is in the class path
Bindings mode	Supported if the JNI libraries are in the system path	Supported if the JNI libraries are in the system path

The WebSphere MQ resource adapter and the version of WebSphere MQ Java used by the resource adapter must be at the same release level.

WebSphere Application Server, Version 6.0 and the WebSphere MQ resource adapter

WebSphere Application Server, Version 6.0 contains a version of WebSphere MQ Java that provides all the function of the WebSphere MQ resource adapter. A WebSphere MQ JMS application running in WebSphere Application Server, Version 6.0 does not therefore need the WebSphere MQ resource adapter in order to access the resources of a WebSphere MQ queue manager.

However, if you do use the WebSphere MQ resource adapter instead, you must set the WebSphere Application Server environment variable MQ_INSTALL_ROOT to the fully qualified path name of the directory where you installed WebSphere MQ. An application running in WebSphere Application Server then uses the version of WebSphere MQ JMS that is supplied with WebSphere MQ and is compatible with the WebSphere MQ resource adapter. For example, on HP-UX, Linux, or Solaris, the value of MQ_INSTALL_ROOT must be /opt/mqm instead of the default value ${WAS_LIBS_DIR}/WMQ.

Configuration of the WebSphere MQ resource adapter

To configure the WebSphere MQ resource adapter, define JCA resources in the following categories:

- The properties of the ResourceAdapter object, which represent the global properties of the resource adapter, such as the level of diagnostic tracing. These properties are described in “Configuration of the ResourceAdapter object” on page 652.
- The properties of an ActivationSpec object, which determine how an MDB is activated for inbound communication. These properties are described in “Configuration for inbound communication” on page 655.
- The properties of a ConnectionFactory object, which the application server uses to create a JMS ConnectionFactory object for outbound communication. These properties are described in “Configuration for outbound communication” on page 661.
- The properties of an administered object, which the application server uses to create a JMS Queue object or JMS Topic object for outbound communication. These properties are also described in “Configuration for outbound communication” on page 661.

The WebSphere MQ resource adapter RAR file contains a file called META-INF/ra.xml, which contains a deployment descriptor for the resource

adapter. This deployment descriptor is defined by the XML schema at http://java.sun.com/xml/ns/j2ee/connector_1_5.xsd and contains information about the resource adapter and the services that it provides. An application server might also require a deployment plan for the resource adapter. This deployment plan is specific to the application server. For example, WebSphere Application Server Community Edition requires a deployment plan called geronimo-ra.xml.

If you are using Secure Sockets Layer (SSL), you must specify the locations of the key store file and trust store file as JVM system properties, as in the following example:

```
java ... -Djavax.net.ssl.keyStore=key_store_location
         -Djavax.net.ssl.trustStore=trust_store_location
         -Djavax.net.ssl.keyStorePassword=key_store_password
```

These properties cannot be properties of an ActivationSpec or ConnectionFactory object, and you cannot specify more than one key store for an application server. The properties apply to the whole JVM, and might therefore affect the application server if other applications, running in the application server, are using SSL connections. The application server might also reset these properties to different values. For more information about using SSL with WebSphere MQ JMS, see "Using Secure Sockets Layer (SSL)" on page 373.

An installation verification test (IVT) program is supplied with the WebSphere MQ resource adapter, but you must configure the resource adapter before you can run the program. For information about what you need to configure in order to run the IVT program, see "The installation verification test (IVT) program" on page 667.

Configuration of the ResourceAdapter object

The ResourceAdapter object encapsulates the global properties of the WebSphere MQ resource adapter. The object has two sets of properties:

- Properties associated with diagnostic tracing
- Properties associated with the connection pool managed by the resource adapter

The way you define these properties depends on the administration interfaces provided by your application server.

Table 38 lists the properties of the ResourceAdapter object that are associated with diagnostic tracing.

Table 38. Properties of the ResourceAdapter object that are associated with diagnostic tracing

Name of property	Type	Default value	Description
traceEnabled	String	false	A flag to enable or disable diagnostic tracing. If the value is false, tracing is turned off. If the value is true, a trace is sent to the location specified by the traceDestination property.
traceDestination	String	wmq_jca.trc	The location to where a diagnostic trace is sent. If the value is System.err, the trace is directed to the system error stream instead of a file. Similarly, if the value is System.out, the trace is directed to the system output stream.

Table 38. Properties of the ResourceAdapter object that are associated with diagnostic tracing (continued)

Name of property	Type	Default value	Description
traceLevel	String	3	The level of detail in a diagnostic trace. The value can be in the range 0, which produces no trace, to 10, which provides the most detail. See Table 39 for a description of each level.
timestampsEnabled	String	true	A flag to enable or disable time stamps in a diagnostic trace. If the value is true, time stamps are added. If the value is false, time stamps are not added. An application server might add time stamps to a stream automatically. In this case, set the value of the property to false to avoid the duplication of time stamps.
logWriterEnabled	String	true	A flag to enable or disable the sending of a diagnostic trace to a LogWriter object provided by the application server. If the value is true, the trace is sent to a LogWriter object instead of the location specified by the traceDestination property. If the value is false, any LogWriter object provided by the application server is not used.

Table 39 describes the levels of detail for diagnostic tracing.

Table 39. The levels of detail for diagnostic tracing

Level number	Level of detail
0	No trace.
1	The trace contains error messages.
3	The trace contains error and warning messages.
6	The trace contains error, warning, and information messages.
8	The trace contains error, warning, and information messages, and entry and exit information for methods.
9	The trace contains error, warning, and information messages, entry and exit information for methods, and diagnostic data.
10	The trace contains all trace information.
Note: Any level that is not included in this table is equivalent to the next lowest level. For example, specifying a trace level of 4 is equivalent to specifying a trace level of 3. However, the levels that are not included might be used in future releases of the WebSphere MQ resource adapter, so it is better to avoid using these levels.	

If diagnostic tracing is turned off, error and warning messages are written to the system error stream. If diagnostic tracing is turned on, error messages are written to the system error stream and to the trace destination, but warning messages are written only to the trace destination. However, the trace contains warning messages only if the trace level is 3 or higher.

The resource adapter manages an internal connection pool of JMS connections that are used to deliver messages to MDBs. Table 40 on page 654 lists the properties of the ResourceAdapter object that are associated with the connection pool.

Table 40. Properties of the ResourceAdapter object that are associated with the connection pool

Name of property	Type	Default value	Description
maxConnections	String	10	The maximum number of connections to a WebSphere MQ queue manager.
connectionConcurrency	String	5	The maximum number of MDBs that can be supplied by each connection.
reconnectionRetryCount	String	5	The maximum number of attempts made by the resource adapter to reconnect to a WebSphere MQ queue manager if a connection fails.
reconnectionRetryInterval	String	300 000	The time, in milliseconds, that the resource adapter waits before making another attempt to reconnect to a WebSphere MQ queue manager.

When an MDB is deployed in the application server, the resource adapter attempts to use an existing JMS connection from the connection pool. Each connection can supply more than one MDB up to the maximum specified by the connectionConcurrency property. If there are no connections in the pool, or if all the connections are fully utilized, a new connection is created provided the maximum number of connections specified by the maxConnections property is not exceeded. The maximum number of MDBs that can be deployed is therefore equal to the product of the maxConnections and connectionConcurrency properties, which is 50 by default. If the number of deployed MDBs reaches the maximum, any attempt to deploy another MDB fails. If an MDB is stopped, its connection can be used by another MDB.

If MDBs are likely to receive a high volume of messages, you might need to reduce the value of the connectionConcurrency property. If you need to limit the number of connections, because of restrictions imposed by a firewall for example, you might need to increase the value of the connectionConcurrency property. In general, if many MDBs are to be deployed, increase the value of the maxConnections property.

The reconnectionRetryCount and reconnectionRetryInterval properties govern the behavior of the resource adapter when connections to a WebSphere MQ queue manager fail, because of a network failure for example. When a connection fails, the resource adapter suspends the delivery of messages to all MDBs supplied by that connection for an interval specified by the reconnectionRetryInterval property. The resource adapter then attempts to reconnect to the queue manager. If the attempt fails, the resource adapter makes further attempts to reconnect at intervals specified by the reconnectionRetryInterval property until the limit imposed by the reconnectionRetryCount property is reached. If all attempts fail, delivery is stopped permanently until the MDBs are restarted manually.

In general, the ResourceAdapter object requires no administration. However, to enable diagnostic tracing on a UNIX system for example, you can set the following properties:

```
traceEnabled:       true
traceDestination:   /tmp/wmq_jca.trace
traceLevel:         10
```

These properties have no effect if the resource adapter has not been started, which is the case, for example, when applications using WebSphere MQ resources are running only in the client container. In this situation, you can set the properties for diagnostic tracing as Java Virtual Machine (JVM) system properties. You can do this by using the -D flag on the **java** command, as in the following example:

```
java ... -DtraceEnabled=true -DtraceDestination=System.err -DtraceLevel=6
```

You do not need to define all the properties of the ResourceAdapter object. Any properties left unspecified take their default values. In a managed environment, it is better not to mix the two ways of specifying properties. If you do mix them, the JVM system properties take precedence over the properties of the ResourceAdapter object.

Configuration for inbound communication

To configure inbound communication, define the properties of one or more ActivationSpec objects. The properties of an ActivationSpec object determine how an MDB receives JMS messages from a WebSphere MQ queue. The transactional behavior of the MDB is defined in its deployment descriptor.

An ActivationSpec object has two sets of properties:

- Properties that are used to create a JMS connection to a WebSphere MQ queue manager
- Properties that are used to create a JMS connection consumer that delivers messages asynchronously as they arrive on a specified queue

The way in which you define the properties of an ActivationSpec object depends on the administration interfaces provided by your application server.

Table 41 lists the properties of an ActivationSpec object that are used to create a JMS connection to a WebSphere MQ queue manager.

Table 41. Properties of an ActivationSpec object that are used to create a JMS connection

Name of property	Type	Valid values (default value in bold)	Description
brokerCCDurSubQueue	String	• **SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE** • A queue name	The name of the queue from which a connection consumer receives durable subscription messages
brokerCCSubQueue	String	• **SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE** • A queue name	The name of the queue from which a connection consumer receives nondurable subscription messages
brokerControlQueue	String	• **SYSTEM.BROKER.CONTROL.QUEUE** • A queue name	The name of the broker control queue
brokerQueueManager	String	• "" **(empty string)** • A queue manager name	The name of the queue manager on which the broker is running
brokerSubQueue	String	• **SYSTEM.JMS.ND.SUBSCRIBER.QUEUE** • A queue name	The name of the queue from which a nondurable message consumer receives messages
brokerVersion	String	• **1** • 2	The version of the broker being used

Table 41. Properties of an ActivationSpec object that are used to create a JMS connection (continued)

Name of property	Type	Valid values (default value in bold)	Description
ccdtURL	String	• **null** • A uniform resource locator (URL)	A URL that identifies the name and location of the file containing the client channel definition table and specifies how the file can be accessed
CCSID	String	• **819** • A coded character set identifier supported by the Java virtual machine (JVM)	The coded character set identifier for a connection
channel	String	• **SYSTEM.DEF.SVRCONN** • The name of an MQI channel	The name of the MQI channel to use
cleanupInterval	int	• **3 600 000** • A positive integer	The interval, in milliseconds, between background runs of the publish/subscribe cleanup utility
cleanupLevel	String	• **SAFE** • NONE • STRONG • FORCE • NONDUR	The cleanup level for a broker based subscription store
clientID	String	• **null** • A client identifier	The client identifier for a connection
failIfQuiesce	boolean	• **true** • false	Whether calls to certain methods fail if the queue manager is in a quiescing state
hostName	String	• **localhost** • A host name • An IP address	The host name or IP address of the system on which the queue manager resides
localAddress	String	• **null** • A string in the format: `[host_name][(low_port[,high_port])]` where *host_name* is a host name or IP address, *low_port* and *high_port* are TCP port numbers, and brackets denote an optional component	For a connection to a queue manager, this property specifies either or both of the following: • The local network interface to be used • The local port, or range of local ports, to be used
messageSelection	String	• **CLIENT** • BROKER	Determines whether message selection is done by WebSphere MQ JMS or by the broker. Message selection by the broker is not supported when brokerVersion has the value 1.
password	String	• **null** • A password	The default password to use when creating a connection to the queue manager
port	int	• **1414** • A TCP port number	The port on which the queue manager listens
queueManager	String	• "" **(empty string)** • A queue manager name	The name of the queue manager to connect to

Table 41. Properties of an ActivationSpec object that are used to create a JMS connection (continued)

Name of property	Type	Valid values (default value in bold)	Description
receiveExit	String	• **null** • A string comprising one or more items separated by commas, where each item is the fully qualified name of a class that implements the WebSphere MQ base Java interface, MQReceiveExit	Identifies a channel receive exit program, or a sequence of receive exit programs to be run in succession
receiveExitInit	String	• **null** • A string comprising one or more items of user data separated by commas	The user data that is passed to channel receive exit programs when they are called
securityExit	String	• **null** • The fully qualified name of a class that implements the WebSphere MQ base Java interface, MQSecurityExit	Identifies a channel security exit program
securityExitInit	String	• **null** • A string of user data	The user data that is passed to a channel security exit program when it is called
sendExit	String	• **null** • A string comprising one or more items separated by commas, where each item is the fully qualified name of a class that implements the WebSphere MQ base Java interface, MQSendExit	Identifies a channel send exit program, or a sequence of send exit programs to be run in succession
sendExitInit	String	• **null** • A string comprising one or more items of user data separated by commas	The user data that is passed to channel send exit programs when they are called
sslCertStores	String	• **null** • A string of one or more LDAP URLs separated by blanks. Each LDAP URL has the format: `ldap://host_name[:port]` where *host_name* is a host name or IP address, *port* is a TCP port number, and brackets denote an optional component.	The Lightweight Directory Access Protocol (LDAP) servers that hold certificate revocation lists (CRLs) for use on an SSL connection
sslCipherSuite	String	• **null** • The name of a CipherSuite	The CipherSuite to use for an SSL connection
sslFipsRequired[1]	boolean	• **false** • true	Whether an SSL connection must use a CipherSuite that is supported by the IBM Java JSSE FIPS provider (IBMJSSEFIPS)
sslPeerName	String	• **null** • A template for distinguished names	For an SSL connection, a template that is used to check the distinguished name in the digital certificate provided by the queue manager
sslResetCount	int	• **0** • An integer in the range 0 to 999 999 999	The total number bytes sent and received by an SSL connection before the secret keys used by SSL are renegotiated

Table 41. Properties of an ActivationSpec object that are used to create a JMS connection (continued)

Name of property	Type	Valid values (default value in bold)	Description
subscriptionStore	String	• **MIGRATE** • QUEUE • BROKER	Determines where WebSphere MQ JMS stores persistent data about active subscriptions
transportType	String	• **CLIENT** • BINDINGS	Whether a connection to a queue manager uses client mode or bindings mode
username	String	• **null** • A user name	The default user name to use when creating a connection to a queue manager
Note: 1. For important information about using the sslFipsRequired property, see "Limitations of the WebSphere MQ resource adapter" on page 671.			

Table 42 lists the properties of an ActivationSpec object that are used to create a JMS connection consumer.

Table 42. Properties of an ActivationSpec object that are used to create a JMS connection consumer

Name of property	Type	Valid values (default value in bold)	Description
destination	String	A destination name	The destination from which to receive messages. The useJNDI property determines how the value of this property is interpreted.
destinationType	String	• javax.jms.Queue • javax.jms.Topic	The type of destination, a queue or a topic
maxMessages	int	• **1** • A positive integer	The maximum number of messages that can be assigned to a server session at one time
maxPoolSize	int	• **10** • A positive integer	The maximum number of server sessions in the server session pool used by the connection consumer
messageSelector	String	• **null** • An SQL92 message selector expression	A message selector expression specifying which messages are to be delivered
poolTimeout	int	• **300 000** • A positive integer	The period of time, in milliseconds, that an unused server session is held open in the server session pool before being closed due to inactivity
startTimeout	int	• **10 000** • A positive integer	The period of time, in milliseconds, within which delivery of a message to an MDB must start after the work to deliver the message has been scheduled. If this period of time elapses, the message is rolled back onto the queue.
subscriptionDurability	String	• **NonDurable** - A nondurable subscription is used to deliver messages to an MDB subscribing to the topic. • Durable - A durable subscription is used to deliver messages to an MDB subscribing to the topic.	Whether a durable or nondurable subscription is used to deliver messages to an MDB subscribing to the topic

Table 42. Properties of an ActivationSpec object that are used to create a JMS connection consumer (continued)

Name of property	Type	Valid values (default value in bold)	Description
subscriptionName	String	• "" **(empty string)** • A subscription name	The name of the durable subscription
useJNDI	boolean	• **false** - The property called destination is interpreted as the name of a WebSphere MQ queue or a topic. • true - The property called destination is interpreted as the name of a javax.jms.Queue object or javax.jms.Topic object in the application server's JNDI namespace.	Determines how the value of the property called destination is interpreted

The ActivationSpec properties called destination and destinationType must be defined explicitly. All the other properties are optional.

An ActivationSpec object can have conflicting properties. For example, you can specify SSL properties for a connection in bindings mode. In this case, the behavior is determined by the transport type and the messaging domain, which is either point-to-point or publish/subscribe as determined by the destinationType property. Any properties that are not applicable to the specified transport type or messaging domain are ignored.

If you define a property that requires other properties to be defined, but you do not define these other properties, the ActivationSpec object throws an InvalidPropertyException exception when its validate() method is called during the deployment of an MDB. The exception is reported to the administrator of the application server in a manner that depends on the application server. For example, if you set the subscriptionDurability property to Durable, indicating that you want use durable subscriptions, you must also define the subscriptionName property.

If the properties called ccdtURL and channel are both defined, an InvalidPropertyException exception is thrown. However, if you define the ccdtURL property only, leaving the property called channel with its default value of SYSTEM.DEF.SVRCONN, no exception is thrown, and the client channel definition table identified by the ccdtURL property is used to start a JMS connection.

Most of the properties of an ActivationSpec object are equivalent to properties of WebSphere MQ JMS objects or parameters of WebSphere MQ JMS methods. However, three tuning properties, and one usability property, have no equivalents in WebSphere MQ JMS:

startTimeout
The time, in milliseconds, that the work manager of the application server waits for resources to become available after the resource adapter schedules a Work object to deliver a message to an MDB. If this period of time elapses before delivery of the message starts, the Work object times out, the message is rolled back onto the queue, and the resource adapter can then make another attempt to deliver the message. A warning is written to diagnostic trace, if enabled, but this does not otherwise affect the process of delivering messages. You might expect this condition to occur only at times when the application server is experiencing a very high load.

If the condition occurs regularly, consider increasing the value of this property to give the work manager longer to schedule message delivery.

maxPoolSize
The maximum number of server sessions in the server session pool used by a connection consumer. The connection consumer uses a server session to deliver a message to an MDB. A larger pool size allows more messages to be delivered concurrently in high volume situations, but uses more resources of the application server. If many MDBs are to be deployed, consider making the pool size smaller in order to maintain the load on the application server at a manageable level. Note that each connection consumer uses its own server session pool, so that this property does not define the total number of server sessions available to all connection consumers.

poolTimeout
The period of time, in milliseconds, that an unused server session is held open in the server session pool before being closed due to inactivity. A transient increase in the message workload causes additional server sessions to be created in order to distribute the load but, after the message workload returns to normal, the additional server sessions remain in the pool and are not used.

Every time a server session is used, it is marked with a timestamp. Periodically a scavenger thread checks that each server session has been used within the period specified by this property. If a server session has not been used, it is closed and removed from the server session pool. A server session might not be closed immediately after the specified period has elapsed, this property represents the minimum period of inactivity before removal.

useJNDI
For a description of this property, see Table 42 on page 658.

To deploy an MDB, first define the properties of an ActivationSpec object, specifying the properties that the MDB requires. The following example is a typical set of properties that you might define explicitly:

```
channel:          SYSTEM.DEF.SVRCONN
destination:      SYSTEM.DEFAULT.LOCAL.QUEUE
destinationType:  javax.jms.Queue
hostName:         192.168.0.42
messageSelector:  color='red'
port:             1414
queueManager:     ExampleQM
transportType:    CLIENT
```

The application server uses the properties to create an ActivationSpec object, which is then associated with an MDB. The properties of the ActivationSpec object determine how messages are delivered to the MDB. Deployment of the MDB fails if the MDB requires distributed transactions but the resource adapter does not support distributed transactions. For information about how to install the resource adapter so that distributed transactions are supported, see “Installation of the WebSphere MQ resource adapter” on page 650.

If more than one MDB is receiving messages from the same destination, then a message sent in the point-to-point domain is received by only one MDB, even if other MDBs are eligible to receive the message. In particular, if two MDBs are using different message selectors, and an incoming message matches both message selectors, only one of the MDBs receives the message. The MDB chosen to receive a

message is undefined, and you cannot rely on a specific MDB receiving the message. Messages sent in the publish/subscribe domain are received by all eligible MDBs.

Poison messages

In some circumstances, a message delivered to an MDB might be rolled back onto a WebSphere MQ queue. This can happen, for example, if a message is delivered within a unit of work that is subsequently rolled back. A message that is rolled back is generally delivered again, but a badly formatted message might repeatedly cause an MDB to fail and therefore cannot be delivered. Such a message is called a *poison message*. You can configure WebSphere MQ so that WebSphere MQ JMS automatically transfers a poison message to another queue for further investigation or discards the message. For information about how to configure WebSphere MQ in this way, see "Handling poison messages" on page 402.

Configuration for outbound communication

When using outbound communication, an application running in an application server starts a connection to a queue manager, and then sends messages to its queues and receives messages from its queues in a synchronous manner. For example, the following servlet method, doGet(), uses outbound communication:

```
protected void doGet(HttpServletRequest request, HttpServletResponse response)
     throws ServletException, IOException {

...

// Look up ConnectionFactory and Queue objects from the JNDI namespace

     InitialContext ic = new InitialContext();
     ConnectionFactory cf = (javax.jms.ConnectionFactory) ic.lookup("myCF");
     Queue q = (javax.jms.Queue) ic.lookup("myQueue");

// Create and start a connection

     Connection c = cf.createConnection();
     c.start();

// Create a session and message producer

     Session s = c.createSession(false, Session.AUTO_ACKNOWLEDGE);
     MessageProducer pr = s.createProducer(q);

// Create and send a message

     Message m = s.createTextMessage("Hello, World!");
     pr.send(m);

// Create a message consumer and receive the message just sent

     MessageConsumer co = s.createConsumer(q);
     Message mr = co.receive(5000);

// Close the connection

      c.close();
}
```

When the servlet receives an HTTP GET request, it retrieves a ConnectionFactory object and a Queue object from the JNDI namespace, and uses the objects to send a message to a WebSphere MQ queue. The servlet then receives the message that it has just sent.

Configuration

To configure outbound communication, define JCA resources in the following categories:

- The properties of a ConnectionFactory object, which the application server uses to create a JMS ConnectionFactory object.
- The properties of an administered object, which the application server uses to create a JMS Queue object or JMS Topic object.

The way you define these properties depends on the administration interfaces provided by your application server. ConnectionFactory, Queue, and Topic objects created by the application server are bound into a JNDI namespace from where they can be retrieved by an application.

Typically, you define one ConnectionFactory object for each queue manager that applications might need to connect to, one Queue object for each queue that applications might need to access in the point-to-point domain, and one Topic object for each topic that applications might want to publish or subscribe to. A ConnectionFactory object can be domain independent. Alternatively, it can be domain specific, a QueueConnectionFactory object for the point-to-point domain or a TopicConnectionFactory object for the publish/subscribe domain.

Table 43 lists the properties of a ConnectionFactory object.

Table 43. Properties of a ConnectionFactory object

Name of property	Type	Valid values (default value in bold)	Description
brokerCCSubQueue	String	• **SYSTEM.JMS.ND.CC.SUBSCRIBER.QUEUE** • A queue name	The name of the queue from which a connection consumer receives nondurable subscription messages
brokerControlQueue	String	• **SYSTEM.BROKER.CONTROL.QUEUE** • A queue name	The name of the broker control queue
brokerPubQueue	String	• **SYSTEM.BROKER.DEFAULT.STREAM** • A queue name	The name of the queue where published messages are sent (the stream queue)
brokerQueueManager	String	• "" **(empty string)** • A queue manager name	The name of the queue manager on which the broker is running
brokerSubQueue	String	• **SYSTEM.JMS.ND.SUBSCRIBER.QUEUE** • A queue name	The name of the queue from which a nondurable message consumer receives messages
brokerVersion	String	• **1** • 2	The version of the broker being used
ccdtURL	String	• **null** • A uniform resource locator (URL)	A URL that identifies the name and location of the file containing the client channel definition table and specifies how the file can be accessed
CCSID	String	• **819** • A coded character set identifier supported by the Java virtual machine (JVM)	The coded character set identifier for a connection
channel	String	• **SYSTEM.DEF.SVRCONN** • The name of an MQI channel	The name of the MQI channel to use
cleanupInterval	int	• **3 600 000** • A positive integer	The interval, in milliseconds, between background runs of the publish/subscribe cleanup utility

Table 43. Properties of a ConnectionFactory object (continued)

Name of property	Type	Valid values (default value in bold)	Description
cleanupLevel	String	• **SAFE** • NONE • STRONG • FORCE • NONDUR	The cleanup level for a broker based subscription store
clientID	String	• **null** • A client identifier	The client identifier for a connection
failIfQuiesce	boolean	• **true** • false	Whether calls to certain methods fail if the queue manager is in a quiescing state
hostName	String	• **localhost** • A host name • An IP address	The host name or IP address of the system on which the queue manager resides
localAddress	String	• **null** • A string in the format: `[host_name][(low_port[,high_port])]` where *host_name* is a host name or IP address, *low_port* and *high_port* are TCP port numbers, and brackets denote an optional component	For a connection to a queue manager, this property specifies either or both of the following: • The local network interface to be used • The local port, or range of local ports, to be used
messageSelection	String	• **CLIENT** • BROKER	Determines whether message selection is done by WebSphere MQ JMS or by the broker. Message selection by the broker is not supported when brokerVersion has the value 1.
password	String	• **null** • A password	The default password to use when creating a connection to the queue manager
port	int	• **1414** • A TCP port number	The port on which the queue manager listens
pubAckInterval	int	• **25** • A positive integer	The number of messages published by a publisher before WebSphere MQ JMS requests an acknowledgement from the broker.
queueManager	String	• `""` **(empty string)** • A queue manager name	The name of the queue manager to connect to
receiveExit	String	• **null** • A string comprising one or more items separated by commas, where each item is the fully qualified name of a class that implements the WebSphere MQ base Java interface, MQReceiveExit	Identifies a channel receive exit program, or a sequence of receive exit programs to be run in succession
receiveExitInit	String	• **null** • A string comprising one or more items of user data separated by commas	The user data that is passed to channel receive exit programs when they are called
securityExit	String	• **null** • The fully qualified name of a class that implements the WebSphere MQ base Java interface, MQSecurityExit	Identifies a channel security exit program

Table 43. Properties of a ConnectionFactory object (continued)

Name of property	Type	Valid values (default value in bold)	Description
securityExitInit	String	• **null** • A string of user data	The user data that is passed to a channel security exit program when it is called
sendExit	String	• **null** • A string comprising one or more items separated by commas, where each item is the fully qualified name of a class that implements the WebSphere MQ base Java interface, MQSendExit	Identifies a channel send exit program, or a sequence of send exit programs to be run in succession
sendExitInit	String	• **null** • A string comprising one or more items of user data separated by commas	The user data that is passed to channel send exit programs when they are called
sslCertStores	String	• **null** • A string of one or more LDAP URLs separated by blanks. Each LDAP URL has the format: `ldap://host_name[:port]` where *host_name* is a host name or IP address, *port* is a TCP port number, and brackets denote an optional component.	The Lightweight Directory Access Protocol (LDAP) servers that hold certificate revocation lists (CRLs) for use on an SSL connection
sslCipherSuite	String	• **null** • The name of a CipherSuite	The CipherSuite to use for an SSL connection
sslFipsRequired[1]	boolean	• **false** • true	Whether an SSL connection must use a CipherSuite that is supported by the IBM Java JSSE FIPS provider (IBMJSSEFIPS)
sslPeerName	String	• **null** • A template for distinguished names	For an SSL connection, a template that is used to check the distinguished name in the digital certificate provided by the queue manager
sslResetCount	int	• **0** • An integer in the range 0 to 999 999 999	The total number bytes sent and received by an SSL connection before the secret keys used by SSL are renegotiated
subscriptionStore	String	• **MIGRATE** • QUEUE • BROKER	Determines where WebSphere MQ JMS stores persistent data about active subscriptions
targetClientMatching	boolean	• **true** • false	Whether a reply message, sent to the queue identified by the JMSReplyTo header field of an incoming message, has an MQRFH2 header only if the incoming message has an MQRFH2 header

Table 43. Properties of a ConnectionFactory object (continued)

Name of property	Type	Valid values (default value in bold)	Description
tempQPrefix	String	• *""* **(empty string)** • A prefix that can be used to form the name of a WebSphere MQ dynamic queue. The rules for forming the prefix are the same as those for forming the contents of the *DynamicQName* field in a WebSphere MQ object descriptor, structure MQOD, but the last non blank character must be an asterisk (*). If the value of the property is the empty string, WebSphere MQ JMS uses the value AMQ.* when creating a dynamic queue.	The prefix that is used to form the name of a WebSphere MQ dynamic queue.
transportType	String	• **CLIENT** • BINDINGS	Whether a connection to a queue manager uses client mode or bindings mode
username	String	• **null** • A user name	The default user name to use when creating a connection to a queue manager
Note: 1. For important information about using the sslFipsRequired property, see "Limitations of the WebSphere MQ resource adapter" on page 671.			

The following example shows a typical set of properties of a ConnectionFactory object:

```
channel:        SYSTEM.DEF.SVRCONN
hostName:       192.168.0.42
port:           1414
queueManager:   ExampleQM
transportType:  CLIENT
```

Table 44 lists the properties that are common to a Queue object and a Topic object.

Table 44. Properties that are common to a Queue object and a Topic object

Name of property	Type	Valid values (default value in bold)	Description
CCSID	String	• **1208** • A coded character set identifier supported by the Java virtual machine (JVM)	The coded character set identifier for the destination
encoding	String	• **NATIVE** • A string of three characters: – The first character specifies the representation of binary integers: - *N* denotes normal encoding. - *R* denotes reverse encoding. – The second character specifies the representation of packed decimal integers: - *N* denotes normal encoding. - *R* denotes reverse encoding. – The third character specifies the representation of floating point numbers: - *N* denotes standard IEEE encoding. - *R* denotes reverse IEEE encoding. - *3* denotes zSeries encoding. NATIVE is equivalent to the string NNN.	The representation of binary integers, packed decimal integers, and floating point numbers for the destination.

Table 44. Properties that are common to a Queue object and a Topic object (continued)

Name of property	Type	Valid values (default value in bold)	Description
expiry	String	• **APP** - The expiry time of a message is determined by the message producer. • UNLIM - A message never expires. • A positive integer representing the expiry time of a message in milliseconds.	The expiry time of a message sent to the destination
failIfQuiesce	String	• **true** • false	Whether an attempt to access the destination fails if the queue manager is in a quiescing state
persistence	String	• **APP** - The persistence of a message is determined by the message producer. • QDEF - The persistence of a message is determined by the *DefPersistence* attribute of the WebSphere MQ queue. • PERS - A message is persistent. • NON - A message is nonpersistent. • HIGH - The persistence of a message is determined by the *NonPersistentMessageClass* attribute of the WebSphere MQ queue according to the explanation in "JMS persistent messages" on page 365.	The persistence of a message sent to the destination
priority	String	• **APP** - The priority of a message is determined by the message producer. • QDEF - The priority of a message is determined by the *DefPriority* attribute of the WebSphere MQ queue. • An integer in the range 0, lowest priority, to 9, highest priority.	The priority of a message sent to the destination
targetClient	String	• **JMS** - The target of a message is a JMS application. • MQ - The target of a message is a non-JMS WebSphere MQ application.	Whether the target of a message sent to the destination is a JMS application. A message whose target is a JMS application contains an MQRFH2 header.

Table 45 lists the properties that are specific to a Queue object.

Table 45. Properties that are specific to a Queue object

Name of property	Type	Valid values (default value in bold)	Description
baseQueueManagerName	String	• "" **(empty string)** • A queue manager name	The name of the queue manager that owns the underlying WebSphere MQ queue
baseQueueName	String	• "" **(empty string)** • A queue name	The name of the underlying WebSphere MQ queue

Table 46 lists the properties that are specific to a Topic object.

Table 46. Properties that are specific to a Topic object

Name of property	Type	Valid values (default value in bold)	Description
baseTopicName	String	• "" **(empty string)** • A topic name	The name of the underlying topic

Table 46. Properties that are specific to a Topic object (continued)

Name of property	Type	Valid values (default value in bold)	Description
brokerCCDurSubQueue	String	• **SYSTEM.JMS.D.CC.SUBSCRIBER.QUEUE** • A queue name	The name of the queue from which a connection consumer receives durable subscription messages
brokerDurSubQueue	String	• **SYSTEM.JMS.D.SUBSCRIBER.QUEUE** • A queue name	The name of the queue from which a durable topic subscriber receives messages
brokerPubQueue	String	• **Not set** • A queue name	The name of the queue where published messages are sent (the stream queue). The value of this property overrides the value of the brokerPubQueue property of the ConnectionFactory object. However, if you do not set the value of this property, the value of the brokerPubQueue property of the ConnectionFactory object is used instead.
brokerPubQueueManager	String	• "" **(empty string)** • A queue manager name	The name of the queue manager that owns the queue where messages published on the topic are sent
brokerVersion	String	• **Not set** • 1 • 2	The version of the broker being used. The value of this property overrides the value of the brokerVersion property of the ConnectionFactory object. However, if you do not set the value of this property, the value of the brokerVersion property of the ConnectionFactory object is used instead.

The following example shows a set of properties of a Queue object:

```
expiry:                 UNLIM
persistence:            QDEF
baseQueueManagerName:   ExampleQM
baseQueueName:          SYSTEM.DEFAULT.LOCAL.QUEUE
```

The following example shows a set of properties of a Topic object:

```
expiry:                 UNLIM
persistence:            NON
baseTopicName:          myTestTopic
```

The installation verification test (IVT) program

The installation verification test (IVT) program is supplied as an enterprise archive (EAR) file called wmq.jmsra.ivt.ear. This file is installed with WebSphere MQ Java in the same directory as the WebSphere MQ resource adapter RAR file, wmq.jmsra.rar. For information about where these files are installed, see "Installation of the WebSphere MQ resource adapter" on page 650.

The installation verification test (IVT) program

You must deploy the IVT program on your application server. The IVT program runs as a servlet and tests that a message can be sent to, and received from, a WebSphere MQ JMS Queue or Topic object. Optionally, you can use the IVT program to verify that the WebSphere MQ resource adapter has been correctly configured to support distributed transactions.

Before you can run the IVT program, you must define the properties of a ConnectionFactory object and a Queue or Topic object as JCA resources, and ensure that your application server creates JMS objects from these definitions and binds them into a JNDI namespace. You can choose the properties of the objects, but the following set of properties is a simple example:

ConnectionFactory object

```
channel:              SYSTEM.DEF.SVRCONN
hostName:             192.168.0.42
port:                 1414
queueManager:         ExampleQM
transportType:        CLIENT
```

Queue object

```
baseQueueManagerName: ExampleQM
baseQueueName:        SYSTEM.DEFAULT.LOCAL.QUEUE
```

By default, the IVT program expects a ConnectionFactory object to be bound in the JNDI namespace with the name IVTCF and a Queue object to be bound with the name IVTQueue. You can use different names, and you can use a Topic object instead of a Queue object as a destination. But if you do, you must enter the names of the objects on the initial page of the IVT program.

After you have deployed the IVT program, and the application server has created the JMS objects and bound them into the JNDI namespace, you can start the IVT program by entering a URL in the following format into your Web browser:

`http://app_server_host:port/WMQ_IVT/`

where *app_server_host* is the IP address or host name of the system on which your application server is running, and *port* is the number of the TCP port on which the application server is listening. Here is an example:

`http://localhost:9080/WMQ_IVT/`

Figure 6 on page 669 shows the initial page of the IVT program.

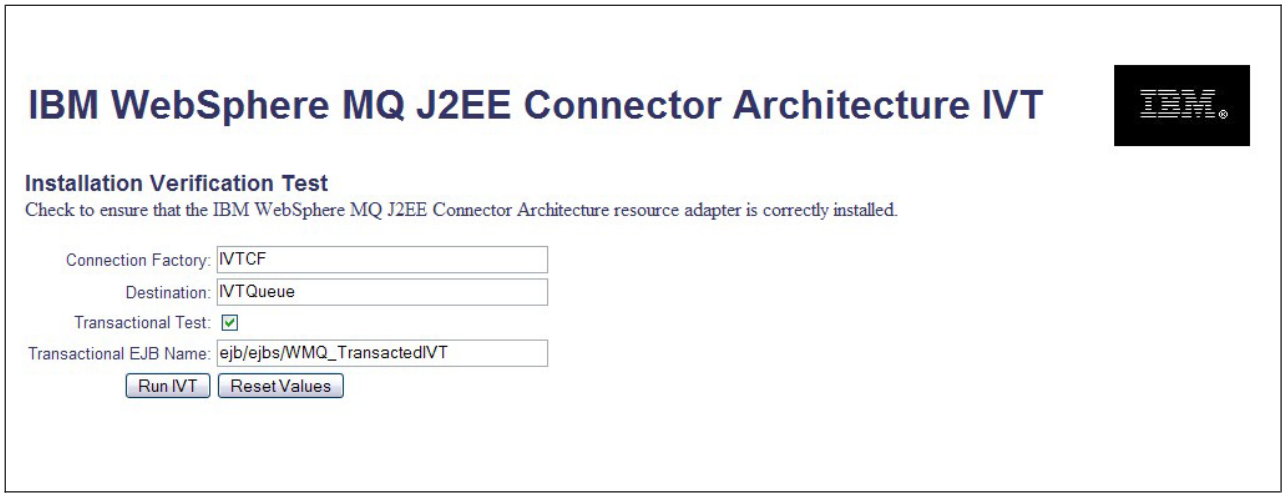

Figure 6. The initial page of the IVT program

On the initial page, the **Connection Factory** field already contains the name `IVTCF` and the **Destination** field already contains the name `IVTQueue`. If you want to run the IVT program using a ConnnectionFactory object and a Queue or Topic object that are bound in the JNDI namespace with different names, you must enter the JNDI names of the objects into these fields, replacing the existing contents.

If you want to verify that the WebSphere MQ resource adapter can support distributed transactions, select the **Transactional Test** check box.

The **Transactional EJB Name** field specifies the JNDI name of the enterprise Java bean (EJB) to be used for the transactional test. By default, the IVT program expects the EJB to be bound with the name ejb/ejbs/WMQ_TransactedIVT, and this name is the initial value of the field. However, application servers use different naming conventions, and this name might not match the JNDI name used by your application server. The IVT program attempts to use some common variations of the default name but, if none of these variations are valid, the IVT program fails with a javax.naming.NameNotFoundException exception. If this happens, you must set this field to the JNDI name used by your application server, replacing the existing contents of the field. To help you work out the JNDI name used by your application server, the file ejb-jar.xml contains the following definition for the EJB:

```
<display-name>WMQ_TransactedIVT</display-name>
<enterprise-beans>
  <session id="WMQ_TransactedIVT">
    <ejb-name>WMQ_TransactedIVT</ejb-name>
    <home>ejbs.WMQ_TransactedIVTHome</home>
    <remote>ejbs.WMQ_TransactedIVT</remote>
    <ejb-class>ejbs.WMQ_TransactedIVTBean</ejb-class>
    <session-type>Stateless</session-type>
    <transaction-type>Container</transaction-type>
  </session>
</enterprise-beans>
```

To run the test, click **Run IVT**. Figure 7 on page 670 shows the page that is displayed if the IVT is successful.

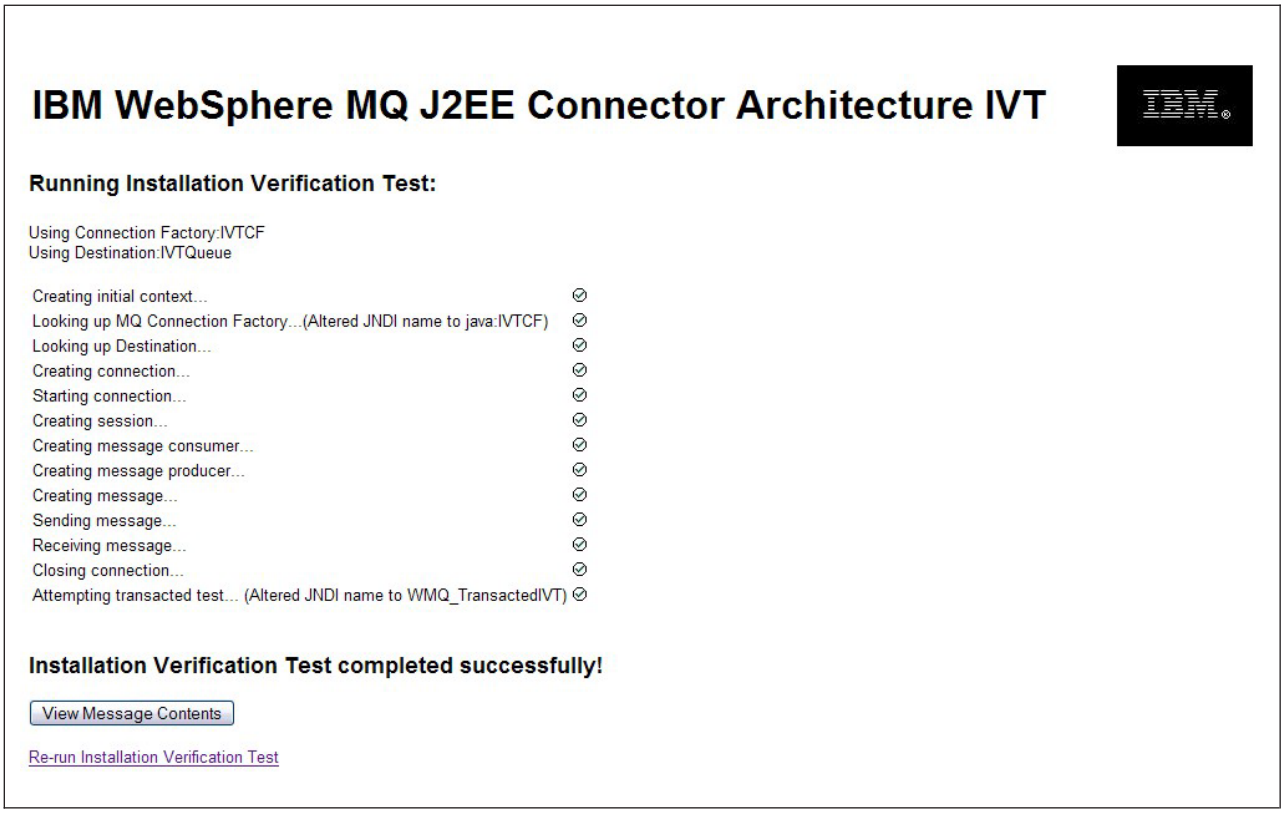

Figure 7. Page showing the results of a successful IVT

If the IVT fails, a page similar to that shown in Figure 8 on page 671 is displayed. To obtain further information about the cause of the failure, click **View Stack Trace**.

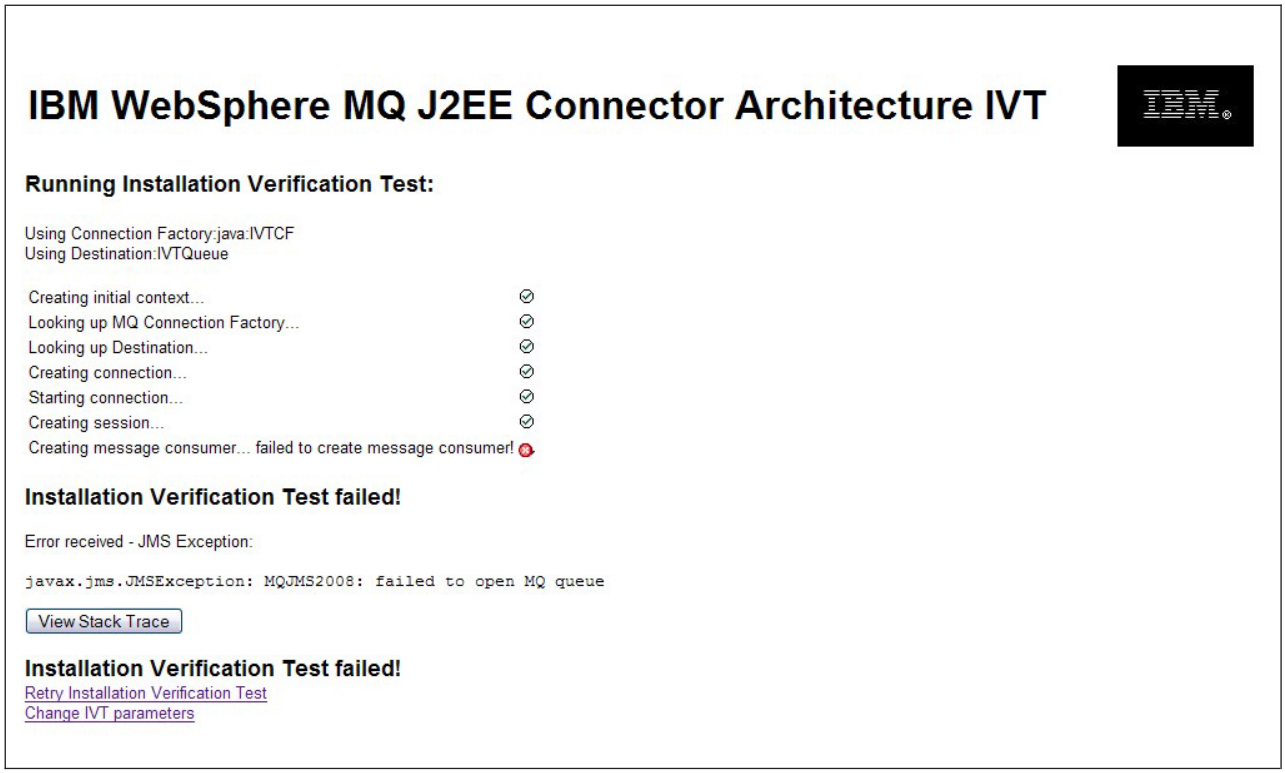

Figure 8. Page showing the results of an IVT that failed

Limitations of the WebSphere MQ resource adapter

The WebSphere MQ resource adapter has the following limitations:

- The WebSphere MQ resource adapter is supported on all WebSphere MQ Version 6.0 platforms, except z/OS.
- The WebSphere MQ resource adapter does not support direct connections to a broker. It supports only connections to a WebSphere MQ queue manager in client or bindings mode.
- The WebSphere MQ resource adapter does not support channel exit programs that are written in languages other than Java.
- While an application server is running, the value of the sslFipsRequired property must be true for all JCA resources or false for all JCA resources. This is a requirement even if the JCA resources are not used concurrently. If the sslFipsRequired property has different values for different JCA resources, WebSphere MQ issues the reason code MQRC_UNSUPPORTED_CIPHER_SUITE, even if an SSL connection is not being used.
- You cannot specify more than one key store for an application server. If connections are made to more than one queue manager, all the connections must use the same key store.

Problem determination

When using the WebSphere MQ resource adapter, most errors cause exceptions to be thrown, and these exceptions are reported to the user in a manner that depends on the application server. The resource adapter makes extensive use of linked exceptions to report problems. Typically, the first exception in a chain is a high

level description of the error, and subsequent exceptions in the chain provide the more detailed information that is required to diagnose the problem.

For example, if the IVT program fails to obtain a connection to a WebSphere MQ queue manager, the following exception might be thrown:

```
javax.jms.JMSException: MQJCA0001: An exception occurred in the JMS layer.
                                   See the linked exception for details.
```

Linked to this exception is a second exception:

```
javax.jms.JMSException: MQJMS2005: failed to create an MQQueueManager for
                                   'localhost:ExampleQM'
```

This exception is thrown by WebSphere MQ JMS and has a further linked exception:

```
com.ibm.mq.MQException: MQJE001: An MQException occurred: Completion Code 2,
                                 Reason 2059
```

This final exception indicates the source of the problem. Reason code 2059 is MQRC_Q_MGR_NOT_AVAILABLE, which indicates that the queue manager specified in the definition of the ConnectionFactory object might not have been started.

If the information provided by exceptions is not sufficient to diagnose a problem, you might need to request a diagnostic trace. For information about how to enable diagnostic tracing, see “Configuration of the WebSphere MQ resource adapter” on page 651.

Configuration problems commonly occur in the following areas:

- “Problems in deploying the resource adapter”
- “Problems in deploying MDBs”
- “Problems in creating connections for outbound communication” on page 673

Problems in deploying the resource adapter

Failures in deploying the resource adapter are generally caused by not configuring JCA resources correctly. For example, a property of the ResourceAdapter object might not be specified correctly, or the deployment plan required by the application server might not be written correctly. Failures might also occur when the application server attempts to create objects from the definitions of JCA resources and bind the objects into the JNDI namespace, but certain properties are not specified correctly or the format of a resource definition is incorrect.

The resource adapter can also fail to deploy because it loaded incorrect versions of JCA or WebSphere MQ JMS classes from JAR files in the class path. This kind of failure can commonly occur on a system where WebSphere MQ is already installed. On such a system, the application server might find existing copies of the WebSphere MQ Java JAR files and load classes from them in preference to the classes supplied in the WebSphere MQ resource adapter RAR file. If the extended transactional client JAR file, com.ibm.mqetclient.jar, cannot be loaded when the resource adapter is deployed, a warning is written to the diagnostic trace, if enabled, but this does not cause deployment to fail.

Problems in deploying MDBs

Failures might occur when the application server attempts to start message delivery to an MDB. This kind of failure is typically caused by an error in the

definition of the associated ActivationSpec object, or because the resources referenced in the definition are not available. For example, the queue manager might not be running, or a specified queue might not exist.

An ActivationSpec object attempts to validate its properties when the MDB is deployed, and deployment fails if the ActivationSpec object has any properties that are mutually exclusive or does not have all the required properties. However, not all problems associated with the properties of the ActivationSpec object can be detected at this time.

Deployment might also fail if an MDB is transacted and the connection is in client mode, but distributed transactions are not available because the extended transactional client JAR file, com.ibm.mqetclient.jar, is not in the class path.

Failures to start message delivery are reported to the user in a manner that depends on the application server. Typically, these failures are reported in the logs and diagnostic trace of the application server. If enabled, the diagnostic trace of the WebSphere MQ resource adapter also records these failures.

Problems in creating connections for outbound communication

Failures in outbound communication commonly occur when an application attempts to look up and use a ConnectionFactory object in a JNDI namespace. A JNDI exception is thrown if the ConnectionFactory object cannot be found in the namespace. A ConnectionFactory object might not be found for the following reasons:

- The application specified an incorrect name for the ConnectionFactory object.
- The application server was not able to create the ConnectionFactory object and bind it into the namespace. In this case, the startup logs of the application server usually contain information about the failure.

If the application successfully retrieves the ConnectionFactory object from the JNDI namespace, an exception might still be thrown when the application calls the ConnectionFactory.createConnection() method. An exception in this context indicates that it is not possible to create a connection to a WebSphere MQ queue manager. Here are some common reasons why an exception might be thrown:

- The queue manager is not available, or cannot be found using the properties of the ConnectionFactory object. For example, the queue manager is not running, or the specified host name, IP address, or port number of the queue manager is incorrect.
- The user is not authorized to connect to the queue manager. For a client connection, if the createConnection() call does not specify a user name, and the application server supplies no user identity information, the JVM process ID is passed to the queue manager as the user name. For the connection to succeed, this process ID must be a valid user name in the system on which the queue manager is running.
- The application is a transacted EJB and therefore the connection must be transacted, but distributed transactions are not available because the extended transactional client JAR file, com.ibm.mqetclient.jar, is not in the class path.
- The ConnectionFactory object has a property called ccdtURL and a property called channel. These properties are mutually exclusive.
- On an SSL connection, the SSL related properties, or the SSL related attributes in the server connection channel definition, have not been specified correctly.

- The sslFipsRequired property has different values for different JCA resources. For more information about this limitation, see "Limitations of the WebSphere MQ resource adapter" on page 671.

The WebSphere MQ resource adapter error and warning messages

Each WebSphere MQ resource adapter error or warning message has an associated message ID with one of the following formats:

MQJCA0*nnn*
: A generic error message

MQJCA1*nnn*
: An error message that applies to the resource adapter or outbound communication

MQJCA2*nnn*
: An error message that applies to inbound communication

MQJCA4*nnn*
: A warning message

where *nnn* is three digits.

Error messages are returned to applications and written to the application server's logs. They are also written to the diagnostic trace of the WebSphere MQ resource adapter, provided diagnostic tracing is turned on.

Warning messages are not returned to applications, but are written to the application server's logs and to the diagnostic trace, provided diagnostic tracing is turned on and the trace level is 3 or higher. For information about the properties that control diagnostic tracing, see "Configuration of the ResourceAdapter object" on page 652.

WebSphere MQ resource adapter error messages

Table 47 describes the WebSphere MQ resource adapter error messages.

Table 47. WebSphere MQ resource adapter error messages

Message ID	Message	Explanation	Action
MQJCA0000	An unknown error occurred.	The resource adapter failed to map an internal reason code to an error message.	This is an internal error. A linked exception or diagnostic trace might provide additional information.
MQJCA0001	An exception occurred in the JMS layer. See the linked exception for details.	A call to WebSphere MQ JMS caused an exception to be thrown.	See the linked exception for details of the failure.
MQJCA0002	An exception occurred in the WebSphere MQ layer. See the linked exception for details.	A call to WebSphere MQ base Java caused an exception to be thrown.	See the linked exception for details of the failure.

Table 47. WebSphere MQ resource adapter error messages (continued)

Message ID	Message	Explanation	Action
MQJCA0003	A JNDI naming exception was thrown. See the linked exception for details.	An attempt to look up a JMS destination in the JNDI namespace failed. Either the namespace cannot be accessed, or a destination with the supplied name is not bound in the namespace. This exception occurs only if the value of the useJNDI property is true.	See the linked exception for details of the failure. Check that the JNDI namespace is available, that a destination with the correct name is bound in the JNDI namespace, and that the value of the property called destination is correctly defined.
MQJCA1001	The MDB cannot be deployed. Inbound messaging is unavailable.	An attempt was made to deploy a MessageEndpoint object when inbound messaging was not available.	Find the earlier exception that indicates why inbound messaging is not available. Use the information to enable inbound messaging.
MQJCA1002	Unknown ActivationSpec implementation supplied to the resource adapter.	The application server attempted to use the WebSphere MQ resource adapter to deploy an ActivationSpec object that is not a WebSphere MQ ActivationSpec object.	This is an application server error. For information about how to diagnose and correct the error, see the documentation for the application server.
MQJCA1003	The message endpoint has no onMessage() method. Deployment failed.	The application server attempted to use the WebSphere MQ resource adapter to deploy a message endpoint that does not implement the JMS MessageListener interface.	This is an application server error. For information about how to diagnose and correct the error, see the documentation for the application server.
MQJCA1004	Distributed transactions are unavailable.	An attempt was made to use distributed transactions in an environment where they are not available.	Make sure that the WebSphere MQ extended transactional client is installed if required, or use a bindings connection.
MQJCA1005	The JCA classes cannot be loaded. Deployment failed.	The JCA classes were not found in the class path.	Make sure that the application server can find the necessary JCA JAR files in the class path.
MQJCA1006	An incorrect version of the JCA classes was found. Deployment failed.	The JCA classes were found in the class path but were not the correct version.	This error can occur if the application server finds connector.jar in the class path. This JAR file is required by WebSphere MQ Java in a J2SE environment, but is not required in a J2EE 1.4 environment.
MQJCA1007	The WebSphere MQ Java classes cannot be loaded. Deployment failed.	The WebSphere MQ Java classes were not found in the class path.	These classes are included in the RAR file of the WebSphere MQ resource adapter. Make sure that the application server can find the classes.
MQJCA1008	An incorrect version of the WebSphere MQ Java classes was found. Deployment failed.	The WebSphere MQ Java classes were found in the class path but were not the correct version.	This error can occur if the application server finds an earlier version of the WebSphere MQ Java classes in the class path. The error can also occur if you do not configure WebSphere Application Server to use the correct version of the WebSphere MQ Java classes.

Table 47. WebSphere MQ resource adapter error messages (continued)

Message ID	Message	Explanation	Action
MQJCA1011	Failed to allocate a JMS connection.	An internal error caused an attempt to allocate a connection to fail.	See the linked exception for details of the failure.
MQJCA1012	Failed to create a JMS connection factory.	A JCA ManagedConnectionFactory object was not able to create a WebSphere MQ JMS ConnectionFactory object.	Check the properties of the ConnectionFactory object.
MQJCA1013	Failed to obtain an XAResource object.	An attempt to access the XAResource object of an XASession object failed.	This is an internal error. A linked exception or diagnostic trace might provide additional information.
MQJCA1014	Transaction begin failed.	An attempt to begin a transaction failed.	This is an internal error. A linked exception or diagnostic trace might provide additional information.
MQJCA1015	Transaction commit failed.	An attempt to commit a transaction failed.	This is an internal error. A linked exception or diagnostic trace might provide additional information.
MQJCA1016	Transaction rollback failed.	An attempt to roll back a transaction failed.	This is an internal error. A linked exception or diagnostic trace might provide additional information.
MQJCA1018	Only one session per connection allowed.	The application attempted to create more than one JMS session on the same JMS connection. This exception occurs only if the application is running in a managed environment.	Modify the application so that it creates only one JMS session on a JMS connection.
MQJCA1019	Connection closed.	The application attempted to use a JMS connection after it had closed the connection.	Modify the application so that it closes the JMS connection only after it has finished using the connection.
MQJCA1020	Session closed.	The application attempted to use a JMS session after it had closed the session.	Modify the application so that it closes the JMS session only after it has finished using the session.
MQJCA1021	Message producer closed.	The application attempted to use a JMS message producer after it had closed the message producer.	Modify the application so that it closes the JMS message producer only after it has finished using the message producer.
MQJCA1022	Message consumer closed.	The application attempted to use a JMS message consumer after it had closed the message consumer.	Modify the application so that it closes the JMS message consumer only after it has finished using the message consumer.
MQJCA1023	Failed to allocate a JMS session.	An attempt to create a JMS session failed.	This is an internal error. A linked exception or diagnostic trace might provide additional information.

Table 47. WebSphere MQ resource adapter error messages (continued)

Message ID	Message	Explanation	Action
MQJCA1024	Session must not have a message listener.	An application attempted to set a message listener for a JMS session. This exception occurs only if the application is running in a managed environment.	Modify the application so that it does use a message listener.
MQJCA1025	Message consumer must not have a message listener.	An application attempted to set a message listener for a JMS message consumer. This exception occurs only if the application is running in a managed environment.	Modify the application so that it does use a message listener.
MQJCA1026	An operation on a domain specific object was not valid.	A JMS application attempted to perform an operation on a domain specific object, but the operation is valid only on cross domain objects or in the other messaging domain.	Make sure that the JMS objects used by your application are relevant to the required messaging domain.
MQJCA1027	Only channel exits written in Java are supported.	A securityExit, sendExit, or receiveExit property referred to a channel exit program that is not written in Java.	Make sure that the definitions of all securityExit, sendExit, and receiveExit properties refer to channel exit programs written only in Java.
MQJCA1028	Re-authentication is not supported.	The application server attempted to re-authenticate a JMS connection, but the WebSphere MQ resource adapter does not support re-authentication.	In the supplied ra.xml file, the property called reauthentication-support has the value false. Make sure that you have not changed the value of this property. If the property still has the value false, then this error is an application server error.
MQJCA2001	Value of the following ActivationSpec property is not valid: {0}	A property of an ActivationSpec object has a value that is not valid.	Supply a valid value for the property.
MQJCA2002	destination property must be defined.	The ActivationSpec property called destination is not defined.	Define the property called destination.
MQJCA2003	destinationType property must be javax.jms.Queue or javax.jms.Topic.	The destinationType property of an ActivationSpec object has a value that is not valid.	Set the value of the destinationType property to `javax.jms.Queue` or `javax.jms.Topic`.
MQJCA2004	subscriptionName property must be defined for a durable subscription.	An attempt was made to use a durable subscription, but the subscriptionName property is not defined.	Define the subscriptionName property.
MQJCA2005	destinationType property must be javax.jms.Topic for durable subscriptions.	An attempt was made to use durable subscriptions, but the value of the destinationType property is javax.jms.Queue.	Set the value of the destinationType property to `javax.jms.Topic`.
MQJCA2006	brokerCCDurSubQueue property must be defined for durable subscriptions.	An attempt was made to use durable subscriptions, but the brokerCCDurSubQueue property is not defined.	Define the brokerCCDurSubQueue property.

WebSphere MQ resource adapter warning messages

Table 48 describes the WebSphere MQ resource adapter warning messages.

Table 48. WebSphere MQ resource adapter warning messages

Message ID	Message	Explanation	Action
MQJCA4000	An unknown warning occurred.	The resource adapter failed to map an internal reason code to a warning message.	This is an internal error.
MQJCA4001	Unknown exception message ID: {0}	The exception builder was asked to build an exception but does not understand the requested type.	This is an internal error.
MQJCA4002	Failed to find javax.jms.MessageListener class. Inbound messaging unavailable. Check the class path.	The javax.jms.MessageListener class was not found in the class path. Inbound messaging is not available, although outbound messaging might still work.	A problem with the class path has prevented inbound messaging from starting. Check the class path.
MQJCA4003	A recoverable exception occurred in the JMS layer. See the linked exception for details.	A call to a JMS method threw an exception, but this failure does not directly affect the operation of the resource adapter.	Investigate the cause of the error. The exception might indicate configuration problems, or more serious issues with the resource adapter or WebSphere MQ queue manager.
MQJCA4004	Message delivery to an MDB failed. See the linked exception for details.	An attempt to deliver a message to a Message Endpoint failed. The message has been rolled back onto the queue.	You can ignore a single warning if a subsequent attempt to deliver the message is successful. Repeated warnings might indicate problems with the resource adapter or the WebSphere MQ queue manager, or that a corrupt message is on the queue.
MQJCA4005	Distributed transactions are not available in client mode.	The extended transactional client JAR file, com.ibm.mqetclient.jar, was not be found in the class path.	If the extended transactional client is not required, no action is necessary.
MQJCA4011	An exception was thrown by ManagedConnection.destroy().	An exception was thrown while a managed connection was closing.	The error does not affect the operation of the resource adapter because the managed connection was no longer being used. However, the error might indicate configuration problems, or more serious issues with the resource adapter or WebSphere MQ queue manager.
MQJCA4013	A connection to a queue manager failed. Check the queue manager error logs for details.	A connection to a WebSphere MQ queue manager was broken.	If the reconnectionRetryCount property is not zero, the resource adapter attempts to reconnect to the queue manager. Until the connection is restarted, the delivery of messages to any MDBs supplied by the connection is suspended.

Table 48. WebSphere MQ resource adapter warning messages (continued)

Message ID	Message	Explanation	Action
MQJCA4014	Failed to reconnect one or more MDBs after a connection failure.	The connection supplying one or more MDBs failed, and the resource adapter was not able to reconnect within the number of attempts specified by the reconnectionRetryCount property.	Make sure that the WebSphere MQ queue manager is running, and that any other required components such as a listener are also running. Examine the application server logs to determine which MDBs have failed and restart the MDBs manually.

Appendix G. Notices

This information was developed for products and services offered in the United States. IBM may not offer the products, services, or features discussed in this information in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this information. The furnishing of this information does not give you any license to these patents. You can send license inquiries, in writing, to:

- IBM Director of Licensing
- IBM Corporation
- North Castle Drive
- Armonk, NY 10504-1785
- U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

- IBM World Trade Asia Corporation
- Licensing
- 2-31 Roppongi 3-chome, Minato-ku
- Tokyo 106, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law:
INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the information. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this information at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Licensees of this program who wish to have information about it for the purpose of enabling: (i) the exchange of information between independently created programs and other programs (including this one) and (ii) the mutual use of the information which has been exchanged, should contact:
- IBM United Kingdom Laboratories,
- Mail Point 151,
- Hursley Park,
- Winchester,
- Hampshire,
- England
- SO21 2JN.

Such information may be available, subject to appropriate terms and conditions, including in some cases, payment of a fee.

The licensed program described in this information and all licensed material available for it are provided by IBM under terms of the IBM Customer Agreement, IBM International Programming License Agreement, or any equivalent agreement between us.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs. You may copy, modify, and distribute these sample programs in any form without payment to IBM for the purposes of developing, using, marketing, or distributing application programs conforming to IBM's application programming interfaces.

Trademarks

The following terms are trademarks of International Business Machines Corporation in the United States, or other countries, or both:

AIX	CICS	CICS/VSE
DB2	Everyplace	i5/OS
IBM	IMS	iSeries
Language Environment	Lotus Notes	MQSeries
MVS	MVS/ESA	OS/2
OS/390	OS/400	POWER
S/390	System/370	System/390
WebSphere	z/OS	zSeries

Java and all Java based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Windows and Windows NT are trademarks of Microsoft Corporation in the United States, other countries, or both.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Linux is a trademark of Linus Torvalds in the United States, other countries, or both.

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

Index

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

W

Z

Sending your comments to IBM

If you especially like or dislike anything about this book, please use one of the methods listed below to send your comments to IBM.

Feel free to comment on what you regard as specific errors or omissions, and on the accuracy, organization, subject matter, or completeness of this book.

Please limit your comments to the information in this book and the way in which the information is presented.

To make comments about the functions of IBM products or systems, talk to your IBM representative or to your IBM authorized remarketer.

When you send comments to IBM, you grant IBM a nonexclusive right to use or distribute your comments in any way it believes appropriate, without incurring any obligation to you.

You can send your comments to IBM in any of the following ways:

- By mail, to this address:

  User Technologies Department (MP095)
  IBM United Kingdom Laboratories
  Hursley Park
  WINCHESTER,
  Hampshire
  SO21 2JN
  United Kingdom
- By fax:
  - From outside the U.K., after your international access code use 44-1962-816151
  - From within the U.K., use 01962-816151
- Electronically, use the appropriate network ID:
  - IBM Mail Exchange: GBIBM2Q9 at IBMMAIL
  - IBMLink™: HURSLEY(IDRCF)
  - Internet: idrcf@hursley.ibm.com

Whichever method you use, ensure that you include:

- The publication title and order number
- The topic to which your comment applies
- Your name and address/telephone number/fax number/network ID.

SC34-6591-02

Spine information:

WebSphere MQ Using Java Version 6.0