Migration Guide

iPlanet Message Queue for Java™

Version 2.0

May 2001

Contents

List of Tables

Migrating from JMQ 1.1 to iMQ 2.0

This document provides the information you need to make a smooth transition between Java Message Queue (JMQ) 1.1 and iPlanet Message Queue for Java (iMQ) 2.0. It consists of the following sections:

Summary: Major Improvements in iMQ 2.0

iMQ 2.0 is a newly architected version of the iMQ product that provides significant advantages over JMQ 1.1 in the following areas:

Better Performance The redesign of the iMQ 2.0 Message Service provides significant improvement in message delivery throughput and overall performance.

Portability iMQ 2.0 has been re-written in Java, rather than in C. This provides Java-based portability to the iMQ product. (iMQ 2.0 can be installed on NT, Solaris, and Linux operating systems and has been tested and certified on these platforms.)

Administration iMQ 2.0 includes new command line and graphical administration tools with capabilities that allow iMQ to be a truly *administered* messaging system. An administrator can manage the iMQ Message Service, destinations, and user security, thereby controlling the operation of the messaging system so as to optimize performance and reliability.

Compatibility Issues

The iMQ Message Service has been totally rewritten, and iMQ administered objects have been changed. This leads to the following compatibilities and incompatibilities between JMQ 1.1 and iMQ 2.0.

Compatibilities

A provider-independent JMQ 1.1 client application is compatible with iMQ 2.0. This means that provider-independent client code can run with the iMQ 2.0 client runtime, and can thereby connect to an iMQ 2.0 Message Service to send and receive messages.

Incompatibilities

An iMQ 2.0 *broker* (called a *router* in JMQ 1.1) cannot connect to a JMQ 1.1 router and *vise versa*. All JMQ 1.1 persistent data is incompatible with iMQ 2.0 persistent data.

Also, the iMQ 2.0 client runtime is not compatible with JMQ 1.1 routers, and, similarly, the JMQ 1.1 client runtime is not compatible with iMQ 2.0 brokers. This means that client applications running with the JMQ 1.1 client runtime must now use the iMQ 2.0 client runtime to communicate with iMQ 2.0 brokers.

In addition, a provider-specific JMQ 1.1 client application (that is, one which contains code that instantiates administered objects) cannot run with the iMQ 2.0 client runtime. This means that the code instantiating administered objects must be re-written to be compatible with the iMQ 2.0 client runtime (see "Migrating JMQ 1.1 Client Applications to iMQ 2.0" on page 21).

How iMQ 2.0 Differs From JMQ 1.1

iMQ 2.0 is a very different product from JMQ 1.1 as detailed in the following sections. For the most part these differences fall into the following categories:

- iMQ Message Service implementation
- iMQ client runtime
- administered objects
- administration tools

iMQ Message Service Implementation

Message routing and delivery is performed in iMQ 2.0 by one or more brokers (called routers in JMQ 1.1). The iMQ 2.0 broker has many new features and is implemented quite differently than the JMQ 1.1 router.

Environment Variables

iMQ defines two environment variables: `JMQ_HOME` and `JMQ_VARHOME`. On Solaris, Linux, and Windows iMQ computes these variables automatically: On Windows, the installer sets `JMQ_HOME` to the iMQ installation directory and `JMQ_VARHOME` to `JMQ_HOME\var`. On Solaris and Linux, `JMQ_HOME` is set to `/opt/SUNWjmq and JMQ_VARHOME` is `/var/JMQ_HOME`.

JMQ 1.1 Transient files are stored in the `JMQ_VARHOME/res` directory

iMQ 2.0 Transient files are stored in `JMQ_VARHOME` (and not in the `JMQ_VARHOME/res` directory).

Broker Java Runtime Environment (JRE) Support

JMQ 1.1 The broker (router) was C-based and did not use a Java Runtime. Client applications were supported on JRE 1.2 (which was not bundled with the product).

iMQ 2.0 The broker is supported on JRE 1.3 (which is now bundled with the product).

Broker Startup

JMQ 1.1 The router was started using the `irouter` command with a set of command-line options.

iMQ 2.0 The broker is started using the `jmqbroker` command, which has a very different set of command line options (see "Starting the Broker" in Chapter 4, of the *iMQ Administrator's Guide*). Use `jmqbroker -h` to list these options.

Broker Configuration

JMQ 1.1 The router was configured using command line options, with a `-w` option for storing the current configuration.

iMQ 2.0 The broker is configured using an instance configuration file that is read at startup (see "Configuration Files" in Chapter 4 of the *iMQ Administrator's Guide*). The configuration property settings in this file can be overridden by iMQ Broker command line options (see "Starting the Broker" in Chapter 4 of the *iMQ Administrator's Guide*). Some configuration properties can be overridden *after* startup (using the iMQ Command utility and iMQ Admin tool). Broker configuration properties affect broker resources (such as ingoing and outgoing threads, memory usage, and caching) and behaviors (such as logging, security, and persistence).

Connection Services

JMQ 1.1 The router supported a single TCP-based connection service on port `25374`. You could explicitly change that port using the `irouter -n` command.

iMQ 2.0 The broker supports multiple connection services, including TCP, SSL, and HTTP (see "Connection Services" in Chapter 2 of the *iMQ Administrator's Guide*). In addition it has a Port Mapper component on port `7676` (by default) that maps dynamically allocated prots to the different connection services. You can explicitly change the Port Mapper port using the `jmqbroker -port` command.

Connection Service Management

JMQ 1.1 A connection service was started when you started the router and shut down when you shut down the router. There was no administration tool for pausing or resuming connection services while the router was running.

iMQ 2.0 The iMQ Command utility (`jmqcmd`) and a GUI Administration Console can pause and resume individual connection services without shutting down the broker (see "Managing Connection Services" in Chapter 5 of the *iMQ Administrator's Guide*).

Persistence

JMQ 1.1 The router wrote persistent information to a flat-file store only.

iMQ 2.0 The broker can write persistent information to a flat-file store or to any database with a JDBC-compliant driver. In addition, iMQ 2.0 uses a new file format for storing persistent messages that is incompatible with JMQ 1.1.

Security

JMQ 1.1 Security was not a production grade capability. A separate process, outside the router, was used for authentication. The `jmqadmin` tool was used to start and stop the authentication server and to specify user names and passwords. Authorization and SSL encryption were not supported.

iMQ 2.0 The broker performs both authentication and authorization internally and supports SSL encryption of messages (using a private/public key system). Authentication is performed by a broker component that requests a username and password from the client during connection setup, and then checks them against an existing user repository or a repository created and populated by an administrator (either LDAP-based or a file-based repository) using the iMQ User Manager utility (`jmqusermgr`). Authorization is performed using an access control properties file, which an administrator must set up, to control access by individual users or groups to connection services and destinations (see "Authorizing Users: the Access Control Properties File" in Chapter 6 of the *iMQ Administrator's Guide*).

Logger

JMQ 1.1 JMQ had a command line utility (`irmon`) for displaying router debugging and network statistics.

iMQ 2.0 The broker generates error information and network metrics, and can be configured to write that information to different output channels: to log files or to the console (see "Logger" in Chapter 2 of the *iMQ Administrator's Guide*).

Destination Management

An iMQ Message Service can auto-create destinations when a destination administered object is passed as a parameter in the create `MessageProducer` or create `MessageConsumer` operations, or otherwise referenced.

JMQ 1.1 There was no administration tool for creating, destroying, or managing destinations or durable subscriptions. You could only create or destroy a queue (using `jmqadmin`).

iMQ 2.0 An administrator can explicitly create destinations (both queues and topics) and set their properties (see "Managing Destinations" in Chapter 5 of the *iMQ Administrator's Guide*) using the iMQ Command utility (`jmqcmd`) or the iMQ Administration Console, and can turn off auto-create to give the administrator more control over destination resources (see "Physical Destinations" in Chapter 2 of the *iMQ Administrator's Guide*). In addition, queues can have one of three delivery policies: single, failover, and round robin (see "Physical Destinations" in Chapter 2 of the *iMQ Administrator's Guide*). Also, there are new restrictions on destination names: they must be legal Java language identifiers.

Multi-broker Configurations

JMQ 1.1 JMQ supported fully connected multi-router configurations, but required all routers to be running for persistent information to be propagated.

iMQ 2.0 Multi-broker configurations are much more functional. They can be centrally configured and they provide for propagation and synchronization of all persistent information among the brokers in a cluster, even when brokers are offline for significant periods of time, or newly added to a running cluster (see "Multi-Broker Configurations (Clusters)" in Chapter 2 of the *iMQ Administrator's Guide*).

iMQ Client Runtime

The iMQ 2.0 client runtime has a number of enhancements for adjusting throughput and performance for message delivery. These include an automatic reconnect capability, parameters for adjusting the flow of messages from broker to client, and options for relaxing the mechanisms that guarantee reliable delivery (see "iMQ Client Runtime Configurable Properties" in Chapter 4 of the *iMQ Developer's Guide*).

Client Runtime JRE Support

JMQ 1.1 The client runtime was supported on JRE 1.1.8, 1.2.2 (which was not bundled with the product).

iMQ 2.0 The client runtime is supported on JRE 1.1.8, 1.2.2, and 1.3 (which is now bundled with the product).

Client Application Command Line Options

JMQ 1.1 The client application command-line options were the following:

- `-s <router host>`
- `-n <subnet number>`
- `-t <ack timeout>`

iMQ 2.0 The JMQ 1.1 client application command-line options are no longer valid. The new way of specifying configurable options in iMQ 2.0 is:

`java[-D`*OptionName*`=`*OptionValue*`]...` *Client_Application_Name*

where `-D`*OptionName*`=`*OptionValue* can be repeated for as many options as need to be configured. (See the JavaDocs for a table of options and values.)

Administered Objects

The 2.0 connection factory anddestination administered objects are not compatible with 1.1 administered objects. The configurable properties of 2.0 administered objects are specified as attributes whose values are specified through `setProperties()` methods.

The iMQ 2.0 connection factory has many more properties (attributes) than the JMQ 1.1 connection factory, including those used to configure auto-reconnect behavior, connections using SSL and HTTP protocols, and message delivery throughput and performance (see "iMQ Client Runtime Configurable Properties" in Chapter 4 of the *iMQ Developer's Guide*).

Administered object constructors have also changed to reflect their new structure. These changes in the object constructors require a change in how you instantiate and configure connection factory anddestination administered objects in provider-specific client application code (see "Migrating JMQ 1.1 Client Applications to iMQ 2.0" on page 21).

Administration Tools

This section compares the JMQ 1.1 and iMQ 2.0 administration tools. In both versions, these tools consist of several command line utilities plus a graphical user interface (GUI) administration tool.

Command Line Administration Utilities

JMQ 1.1 provided three command line utilities, shown in the following table:

Table 1 JMQ 1.1 Command Line Utilities

JMQ 1.1 Utility	Description
`ircmd`	Utility to: • query router about it's current connections and interests • reconfigure the connections between routers • verify connections to routers • control network print output • display various status information
`irmon`	Utility that displays router debugging and network output.
`jmqconfig`	Utility used for creating, managing, and configuring JMQ administered objects

In iMQ 2.0, none of the above utilities are supported. They have been replaced by the following iMQ 2.0 command line utilities:

Table 2 iMQ 2.0 Command Line Utilities

iMQ 2.0 Utility	Description	See...
iMQ Command (`jmqcmd`)	Utility for performing diverse administrative tasks on a broker: • control broker and its connection services • manage broker resources • create, update, and delete physical destinations	Chapter 5, "Broker and Application Management" in the *iMQ Administrator's Guide*
iMQ Object Manager (`jmqobjmgr`)	Utility used for creating, managing, and configuring iMQ administered objects. (This is a replacement for `jmqconfig`.)	Chapter 7, "Managing Administered Objects" in the *iMQ Administrator's Guide*
iMQ User Manager (`jmqusermgr`)	Utility for populating the flat-file user repository (user names and passwords) used to authenticate and authorize users.	"Authenticating Users" in Chapter 6, of the *iMQ Administrator's Guide*
iMQ Key Tool (`jmqkeytool`)	Utility used for generating a self-signed certificate used for SSL authentication.	Chapter 6, "Security Management" in the *iMQ Administrator's Guide*
iMQ Database Manager (`jmqdbmgr`)	Utility used for creating a JDBC-compliant database used for persistent storage.	Appendix A, "Setting Up Plugged-in Persistence" in the *iMQ Administrator's Guide*

As you can see by comparing Table 1 to Table 2:

- The functionality of `ircmd` has been largely eliminated.
- The functionality of `irmon` is now controlled through the brokers internal logging (see "Logger" on page 12).
- The `jmqconfig` utility has been replaced by iMQ Object Manager (`jmqobjmgr`). Note that the use of the `jndi.properties` file in JMQ 1.1 to hold commonly used property values in the `jmqconfig` command has been replaced in iMQ 2.0 by the use of a command file, specified by a `-i` option of the `jmqobjmgr` command (see "Using Input Files" in Chapter 7 of the *iMQ Administrator's Guide*). This command file can contain more than just JNDI related property values.
- New capabilities and tools have been added for managing brokers and destinations, user repositories, and persistent data stores.

GUI Administration Tools

JMQ 1.1 provided a GUI administration tool (started by the `jmqadmin` command), which allowed an administrator to:

- manage the `jmqadmin` console logs
- delete queues and their persistent messages
- enable the authentication of users and specify users and passwords

iMQ 2.0 provides a new version of the GUI administration tool, the iMQ Administration Console, (also started by the `jmqadmin` command), which allows an administrator to:

- perform the broker administration provided by iMQ Command (`jmqcmd`): start, pause, shutdown broker or individual connection services
- perform the destination management provided by iMQ Command (`jmqcmd`)
- perform the administered object management provided by iMQ ObJect Manager (`jmqobjmgr`)

Converting JMQ 1.1 Administered Objects to iMQ 2.0

iMQ 2.0 changes both the structure and properties of administered objects. Hence, the iMQ 2.0 client runtime cannot use JMQ 1.1 administered objects without their being converted to iMQ 2.0 administered objects. This conversion can be done permanently, by the iMQ Object Manager utility, or temporarily, by the iMQ client runtime.

Permanent Conversion Using the iMQ Object Manager Utility (jmqobjmgr)

The utility recognizes both iMQ 2.0 and JMQ 1.1 administered objects. For example, the `list` subcommand displays JMQ 1.1 administered objects, if present, and the `query` subcommand displays attributes of a JMQ 1.1 administered object. You can use the `update` subcommand to modify the attributes of a JMQ 1.1 administered object, converting it to an iMQ 2.0 administered object. JMQ 1.1 object attributes that map directly to iMQ 2.0 attributes are converted automatically, and attributes that are new in iMQ 2.0 are assigned default values. Once administered objects have been converted to iMQ 2.0, they will no longer be usable by JMQ 1.1 clients.

Temporary Conversion Using the iMQ Client Runtime

In a JNDI lookup, the iMQ client runtime will transparently convert JMQ 1.1 administered objects that were stored in a JNDI object store (using the JMQ 1.1 `jmqconfig` utility). JMQ 1.1 object attributes that map directly to iMQ 2.0 attributes are converted automatically, and attributes that are new in iMQ 2.0 are assigned default values. Administered objects in the JNDI object store are not modified and will continue to be usable by JMQ 1.1 clients.

jmqconfig and jmqobjmgr Compared

You can also convert JMQ 1.1 administered objects to iMQ 2.0 administered objects by simply creating replacement administered objects using iMQ 2.0 tools. As a guide to this process, Table 3 through Table 6 show example usages of JMQ 1.1 and iMQ 2.0 command line utilities to perform equivalent administered object operations:

Table 3 Adding a Topic

JMQ 1.1 jmqconfig

```
jmqconfig -a
-t t
-n "cn=myTopic"
-o "name=This is my Test Topic"
-i "com.sun.jndi.ldap.LdapCtxFactory"
-u "ldap://mydomain.com:389/o=JMQ"
```

iMQ 2.0 Object Manager (jmqobjmgr)

```
jmqobjmgr add
-t t
-l "cn=myTopic"
-o "JMQDestinationName=This_is_my_Test_Topic"
-j "java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory"
-j "java.naming.provider.url=ldap://mydomain.com:389/o=JMQ"
```

Table 4 Adding a Queue Connection Factory

JMQ 1.1 jmqconfig

```
jmqconfig -a
-t qf
-n "cn=myQCF"
-o "host=foo"
-o "subnet=5"
-i "com.sun.jndi.ldap.LdapCtxFactory"
-u "ldap://mydomain.com:389/o=JMQ"
```

iMQ 2.0 Object Manager (jmqobjmgr)

```
jmqobjmgr add
-t qf
-l "cn=myQCF"
-o "JMQBrokerHostName=foo"
-j "java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory"
-j "java.naming.provider.url=ldap://mydomain.com:389/o=JMQ"
```

Table 5 Deleting a Topic

JMQ 1.1 jmqconfig

```
jmqconfig -d
-f
-n "cn=myTopic"
-i "com.sun.jndi.ldap.LdapCtxFactory"
-u "ldap://mydomain.com:389/o=JMQ"
```

iMQ 2.0 Object Manager (jmqobjmgr)

```
jmqobjmgr delete
-f
-l "cn=myTopic"
-j "java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory"
-j "java.naming.provider.url=ldap://mydomain.com:389/o=JMQ"
```

Table 6 Querying a Topic

JMQ 1.1 jmqconfig

```
jmqconfig -q
-n "cn=myTopic"
-i "com.sun.jndi.ldap.LdapCtxFactory"
-u "ldap://mydomain.com:389/o=JMQ"
-s simple
-p "uid=foo,ou=JMQConfig,o=JMQ"
-c foo
```

iMQ 2.0 Object Manager (jmqobjmgr)

```
jmqobjmgr query
-l "cn=myTopic"
-j "java.naming.factory.initial=com.sun.jndi.ldap.LdapCtxFactory"
-j "java.naming.provider.url=ldap://mydomain.com:389/o=JMQ"
-j "java.naming.security.authentication=simple"
-j "java.naming.security.principal=uid=foo,ou=JMQConfig,o=JMQ"
-j "java.naming.security.credentials=foo"
```

Migrating JMQ 1.1 Client Applications to iMQ 2.0

You only need to migrate a client application if it has provider-specific code that instantiates iMQ administered objects. If the client code uses a JNDI lookup of administered objects, you *do not* have to change the application to run with the iMQ 2.0 client runtime.

Because of changes in iMQ 2.0 of the structure and configuration properties of administered objects, administered object constructors have changed.

ConnectionFactory Administered Objects

JMQ 1.1 Constructors

The following JMQ 1.1 ConnectionFactory administered object constructors have been removed:

- `new com.sun.messaging.TopicConnectionFactory(Object config);`
- `new com.sun.messaging.TopicConnectionFactory(Object config, int port);`
- `new com.sun.messaging.TopicConnectionFactory(int port);`
- `new com.sun.messaging.QueueConnectionFactory(Object config);`
- `new com.sun.messaging.QueueConnectionFactory(Object config, int port);`
- `new com.sun.messaging.QueueConnectionFactory(int port);`

iMQ 2.0 Constructors

The following iMQ ConnectionFactory administered object constructors are still valid and can be used as before in iMQ Client Applications. However, default values may be different as indicated below:

- `new com.sun.messaging.TopicConnectionFactory();`

  Instantiates a `TopicConnectionFactory` with the default configuration that creates Topic Connections to an iMQ 2.0 broker running on `localhost` using the default port `7676`.

- `new com.sun.messaging.QueueConnectionFactory();`

  Instantiates a `QueueConnectionFactory` with the default configuration that creates Queue Connections to an iMQ 2.0 broker running on `localhost` using the default port `7676` and a TCP-based connection service.

To directly instantiate iMQ 2.0 connection factory objects and change their default configuration, you should perform the following steps in your code.

Code Example 1 Instantiating iMQ 2.0 ConnectionFactory Objects

```
com.sun.messaging.TopicConnectionFactory tcf =
        new com.sun.messaging.TopicConnectionFactory();
try {
   tcf.setProperty("JMQBrokerHostName", "new_hostname");
   tcf.setProperty("JMQBrokerHostPort", "7878");
} catch (JMSException je) {
   //If bad configuration property name used OR tcf
      was set to ReadOnly
}
TopicConnection tc =
(TopicConnection)tcf.createTopicConnection();
```

Destination Administered Objects

JMQ 1.1 Constructors

The following JMQ 1.1 Destination administered object constructors have been modified to throw a `JMSException` if the `name` parameter passed is an invalid JMQ Destination name.

- `new com.sun.messaging.Topic(String name);`
- `new com.sun.messaging.Queue(String name);`

iMQ 2.0 Constructors

The following iMQ Destination administered object constructors have been added, with default values as indicated below:

- `new com.sun.messaging.Topic();`

  Instantiates a `Topic` with the default destination name of `Untitled_Destination_Object`.

- `new com.sun.messaging.Queue();`

  Instantiates a `Queue` with the default destination name of `Untitled_Destination_Object`.

To directly instantiate iMQ 2.0 destination administered objects and change their default destination names, you should perform the following steps in your code.

Code Example 2 Instantiating iMQ 2.0 Destination Objects

```
com.sun.messaging.Queue myQ = new com.sun.messaging.Queue();
try {
    myQ.setProperty("JMQDestinationName", "new_queue_name");
} catch (JMSException je) {
// If bad configuration property name used OR myQ was set to
// ReadOnly
}
//Use this to instantiate a Message Producer or Consumer
QueueSender qs = qSession.createSender((Queue)myQ);
...
```